YAYASAN PRIMA AGUS TEKNIK

Desain Manajemen Pola
dengan
Kubernetes lewat Docker, CoreOS Linux dan Platform Lainnya

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

Desain Manajemen Pola

dengan

Kubernetes lewat Docker, CoreOS Linux dan Platform Lainnya

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8642-74-8 (PDF)

9 786238 642748

Desain Manajemen Pola dengan Kubernetes lewat Docker, CoreOS Linux dan platform lainnya

Penulis :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom

ISBN : 978-623-8642-74-8

Editor :
Dr. Ir. Agus Wibowo, M.Kom, M.Si, MM.

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Buku ini dimulai dengan pengenalan tentang instalasi Kubernetes di berbagai platform, dimulai dengan *Amazon Web Services* (AWS) pada Bab 1. Di sini, pembaca akan mempelajari cara menginstal Kubernetes dan membuat serta menjalankan layanan di dalam kluster multi-simpul dengan konfigurasi minimal. Pada Bab 2, fokus beralih ke instalasi Kubernetes di CoreOS, yang akan digunakan sebagai platform utama dalam sebagian besar bab. Proses dimulai dengan pembuatan instans AWS EC2 menggunakan Amazon Linux AMI yang sudah dilengkapi dengan AWS *Command Line Interface* (CLI), diikuti dengan instalasi Kube-aws dan pembuatan kluster Kubernetes menggunakan CloudFormation.

Bab 3 memperkenalkan penggunaan *Google Cloud Platform* (GCP) untuk Kubernetes, di mana pembaca akan membuat proyek dan instans VM, kemudian mengonfigurasi kluster Kubernetes dan menguji aplikasi sampel. Bab 4 membahas penggunaan beberapa zona dalam AWS CloudFormation untuk kluster Kubernetes, sedangkan Bab 5 memperkenalkan Tectonic Console untuk manajemen aplikasi Kubernetes yang diterapkan pada CoreOS. Bab 6 membahas penggunaan volume, dengan fokus pada dua jenis volume, yaitu awsElasticBlockStore dan gitRepo.

Pada Bab 7, pembaca akan mempelajari cara membuat layanan sampel untuk tiga jenis layanan yang didukung Kubernetes: ClusterIP, NodePort, dan LoadBalancer. Bab 8 membahas pembaruan bergulir, yaitu mekanisme untuk memperbarui pengontrol replikasi ke citra atau spesifikasi yang lebih baru saat aplikasi masih berjalan. Bab 9 mengulas kebijakan penjadwalan Kubernetes, termasuk penggunaan NodeSelector dan pengaturan afinitas node. Bab 10 menyajikan cara mengalokasikan sumber daya komputasi seperti CPU dan memori, serta mengatur permintaan dan batasan sumber daya untuk memastikan kualitas layanan.

Bab 11 membahas ConfigMaps, yang digunakan untuk mengonfigurasi variabel lingkungan dan argumen perintah dalam pod atau pengontrol replikasi. Bab 12 menjelaskan pengaturan kuota sumber daya dalam namespace untuk membatasi penggunaan sumber daya sesuai dengan kebutuhan tim atau fase aplikasi seperti pengembangan, pengujian, dan produksi. Bab 13 membahas penskalaan otomatis, yang memungkinkan penskalaan jumlah pod dalam kluster secara otomatis sesuai dengan fluktuasi beban kerja. Pada Bab 14, pencatatan log diperkenalkan dengan menggunakan alat seperti Elasticsearch, Fluentd, dan Kibana untuk pencatatan tingkat kluster.

Bab 15 mengulas penggunaan OpenShift, platform PaaS untuk Kubernetes, untuk membuat kluster Kubernetes induk dengan ketersediaan tinggi menggunakan Ansible. Di Bab 16, buku ini mengajarkan cara mengembangkan situs web dengan ketersediaan tinggi menggunakan AWS Route 53 untuk failover DNS, memastikan situs tetap tersedia meskipun terjadi kegagalan. Selamat membaca.

Semarang, Januari 2025
Penulis

Dr. Joseph Teguh Santoso, S. Kom., M.Kom.

DAFTAR ISI

BAB 1
MANAJEMEN KUBERNETES DENGAN DOCKER DAN COREOS

Pendahuluan

Docker tersedia sebagai sumber terbuka pada Maret 2013 dan telah menjadi platform kontainerisasi yang paling umum digunakan. Kubernetes menjadi sumber terbuka pada Juni 2014 dan telah menjadi pengelola klaster kontainer yang paling banyak digunakan. Versi stabil pertama CoreOS Linux tersedia pada Juli 2014 dan sejak itu telah menjadi sistem operasi yang paling umum digunakan untuk kontainer. Buku pertama saya, Kubernetes Microservices with Docker (Apress, 2016), merupakan pengantar untuk membuat layanan mikro dengan Kubernetes dan Docker. Buku ini, Kubernetes Management Design Patterns, membawa manajemen klaster kontainer ke tingkat berikutnya dan membahas semua atau sebagian besar aspek pengelolaan dan konfigurasi Kubernetes pada CoreOS dan penerapan pola desain yang sesuai seperti ConfigMaps, penskalaan otomatis, kuota sumber daya, dan ketersediaan tinggi.

Kubernetes adalah pengelola klaster untuk kontainer Docker dan rkt, tetapi buku ini membahas Kubernetes dalam konteks Docker saja. Pengelola kluster untuk kontainer Docker diperlukan karena mesin Docker sendiri tidak memiliki beberapa fungsi, seperti kemampuan untuk menskalakan kluster kontainer, menjadwalkan pod pada node, atau memasang jenis penyimpanan tertentu (seperti AWS Volume atau repo Github) sebagai volume. Docker Engine 1.12 mengintegrasikan pengelola kluster Docker Swarm dan Docker Swarm mengatasi beberapa keterbatasan Docker sebelumnya dengan menyediakan replikasi, penyeimbangan beban, toleransi kesalahan, dan penemuan layanan, tetapi Kubernetes menyediakan beberapa fitur yang sesuai untuk mengembangkan aplikasi berorientasi objek.

Abstraksi Pod adalah unit atomik penyebaran di Kubernetes. Pod dapat terdiri dari satu atau beberapa kontainer. Penempatan kontainer secara bersama memiliki beberapa keuntungan karena kontainer dalam Pod berbagi jaringan dan sistem berkas yang sama dan berjalan pada node yang sama. Docker Swarm tidak mendukung penskalaan otomatis secara langsung. Meskipun Docker Swarm merupakan Docker asli, Kubernetes lebih siap produksi karena telah digunakan dalam produksi di Google selama lebih dari 15 tahun.

1.1 POLA DESAIN KUBERNETES

> *Pola desain perangkat lunak adalah solusi umum yang dapat digunakan kembali untuk masalah yang umum terjadi dalam konteks tertentu dalam desain perangkat lunak.*
>
> *Wikipedia*

Gambar Docker mencakup instruksi untuk mengemas semua perangkat lunak dan dependensi yang diperlukan, mengatur variabel lingkungan, dan menjalankan perintah, dan merupakan enkapsulasi perangkat lunak yang dapat digunakan kembali untuk desain modular. Unit atom dari layanan kontainer modular di Kubernetes adalah pod, yang merupakan sekelompok kontainer dengan sistem berkas dan jaringan yang sama. Abstraksi pod

Kubernetes memungkinkan pola desain untuk aplikasi yang dikontainerisasi mirip dengan pola desain berorientasi objek.

Pod, layanan, pengontrol replikasi, penyebaran, dan ConfigMap semuanya adalah jenis objek Kubernetes. Lebih jauh, karena kontainer berinteraksi satu sama lain melalui HTTP, memanfaatkan format data yang umum tersedia seperti JSON, pola desain Kubernetes bersifat independen terhadap bahasa dan platform. Kontainer memberikan beberapa manfaat yang sama seperti objek perangkat lunak seperti modularitas atau pengemasan, abstraksi, dan penggunaan kembali. Kubernetes telah menjelaskan tiga kelas atau jenis pola.

- Pola desain manajemen
- Pola yang melibatkan beberapa kontainer yang bekerja sama dan berjalan pada node yang sama
- Pola yang melibatkan kontainer yang berjalan melintasi beberapa node Berikut ini adalah beberapa manfaat kontainer modular:
- Batas kontainer adalah batas enkapsulasi atau abstraksi yang dapat digunakan untuk membangun komponen modular yang dapat digunakan kembali.
- Kontainer yang dapat digunakan kembali dapat digunakan bersama di antara berbagai aplikasi dan tim pengembang tangkas.
- Kontainer mempercepat pengembangan aplikasi.
- Kontainer cocok untuk pengembangan tim tangkas.
- Kontainer dapat digunakan untuk merangkum desain atau implementasi terbaik.
- Kontainer menyediakan pemisahan perhatian

Pola desain diperkenalkan dalam publikasi Pola Desain untuk Sistem Terdistribusi Berbasis Kontainer, oleh Brendan Burns dan David Oppenheimer (https://www.usenix.org/node/196347). Dalam buku ini, kami akan menggunakan beberapa pola desain ini dan lainnya.

Arsitektur Kubernetes

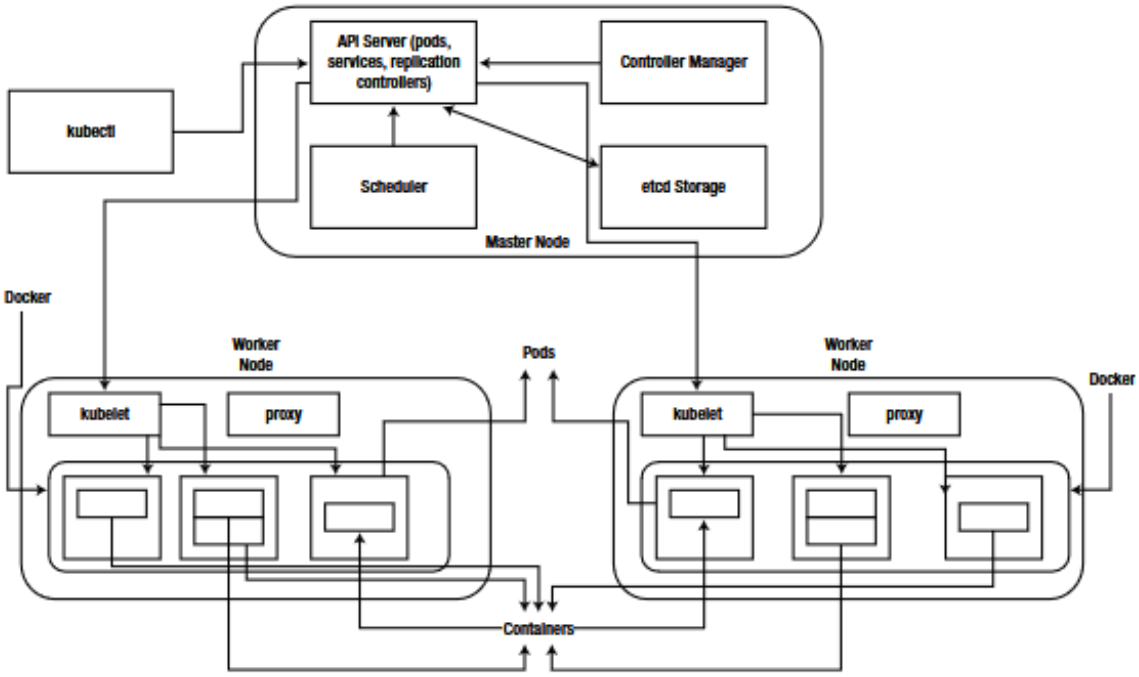
Kluster Kubernetes terdiri dari satu simpul master (kecuali jika digunakan simpul master dengan ketersediaan tinggi, yang bukan merupakan default) dan satu atau beberapa simpul pekerja dengan Docker terpasang pada setiap simpul. Komponen berikut berjalan pada setiap simpul master:

1. etcd untuk menyimpan status persisten dari master termasuk semua data konfigurasi. Kluster etcd dengan ketersediaan tinggi juga dapat digunakan.
2. Server API untuk menyediakan API REST Kubernetes bagi objek Kubernetes (pod, layanan, pengontrol replikasi, dan lainnya).
3. Penjadwal untuk mengikat pod yang tidak ditetapkan pada node.
4. Manajer pengontrol melakukan semua operasi tingkat kluster seperti membuat dan memperbarui titik akhir layanan, menemukan, mengelola, dan memantau node. Pengontrol replikasi digunakan untuk menskalakan pod dalam kluster.

Komponen berikut berjalan pada setiap simpul pekerja:

a. Kubelet untuk mengelola pod (termasuk kontainer), citra Docker, dan volume. Kubelet dikelola dari Server API pada simpul master.

b. Kube-proxy adalah proxy jaringan dan penyeimbang beban untuk menyediakan layanan. Arsitektur Kubernetes ditunjukkan pada Gambar 1.1.



Gambar 1.1 Arsitektur Kubernetes

Mengapa CoreOS?

CoreOS adalah OS Linux yang paling banyak digunakan yang dirancang untuk kontainer, bukan hanya kontainer Docker tetapi juga kontainer rkt (implementasi spesifikasi APP Container). Docker dan rkt sudah terpasang di CoreOS sejak awal. CoreOS mendukung sebagian besar penyedia cloud termasuk Amazon Web Services (AWS) Elastic Compute Cloud (EC2), Google Cloud Platform, dan platform virtualisasi seperti VMWare dan VirtualBox. CoreOS menyediakan Cloud-Config untuk mengonfigurasi secara deklaratif untuk item OS seperti konfigurasi jaringan (flannel), penyimpanan (etcd), dan akun pengguna.

CoreOS menyediakan infrastruktur tingkat produksi untuk aplikasi yang dikontainerisasi termasuk otomatisasi, keamanan, dan skalabilitas. CoreOS telah memimpin dorongan untuk standar industri kontainer dan bahkan mendirikan appc. CoreOS tidak hanya sistem operasi yang paling banyak digunakan untuk kontainer tetapi juga registri kontainer paling canggih, Quay. CoreOS menyediakan keamanan server dengan Distributed Trusted Computing. CoreOS juga menyediakan Tectonic Enterprise untuk beban kerja tingkat perusahaan tanpa overhead operasional dan kluster Kubernetes yang siap pakai serta dasbor yang mudah digunakan.

1.2 KUBERNETES DI AWS

Kubernetes adalah pengelola kluster untuk kontainer Docker (dan rkt). Pendahuluan menguraikan arsitektur dasar dan hubungannya dengan CoreOS dan Amazon Web Services (AWS). Dalam bab ini, kita akan membuat kluster dasar tanpa konfigurasi. Kubernetes Microservices with Docker (Apress, 2016) mencakup penginstalan Kubernetes pada kluster simpul tunggal dan multisimpul.

Masalah

Menginstal Kubernetes dengan menginstal komponen individualnya (Docker, Flannel, Kubelet, dan Service Proxy) secara terpisah merupakan proses yang rumit yang memerlukan banyak perintah untuk dijalankan dan file untuk dikonfigurasi.

Solusi

AWS menyediakan alat lama yang disebut kube-up.sh untuk membuat kluster Kubernetes tanpa memerlukan konfigurasi apa pun. Yang diperlukan hanyalah akun AWS, AWS Command Line Interface (CLI), dan akses ke API AWS. Kubernetes dan alat lain seperti Elasticsearch (digunakan untuk mengindeks dan menyimpan log), Heapster (digunakan untuk menganalisis penggunaan sumber daya komputasi), Kibana (dasbor GUI yang digunakan untuk melihat log), KubeDNS (digunakan untuk menyelesaikan nama DNS untuk layanan), Kubernetes-dashboard, Grafana (digunakan untuk visualisasi metrik), dan InfluxDB semuanya diinstal dengan satu perintah.

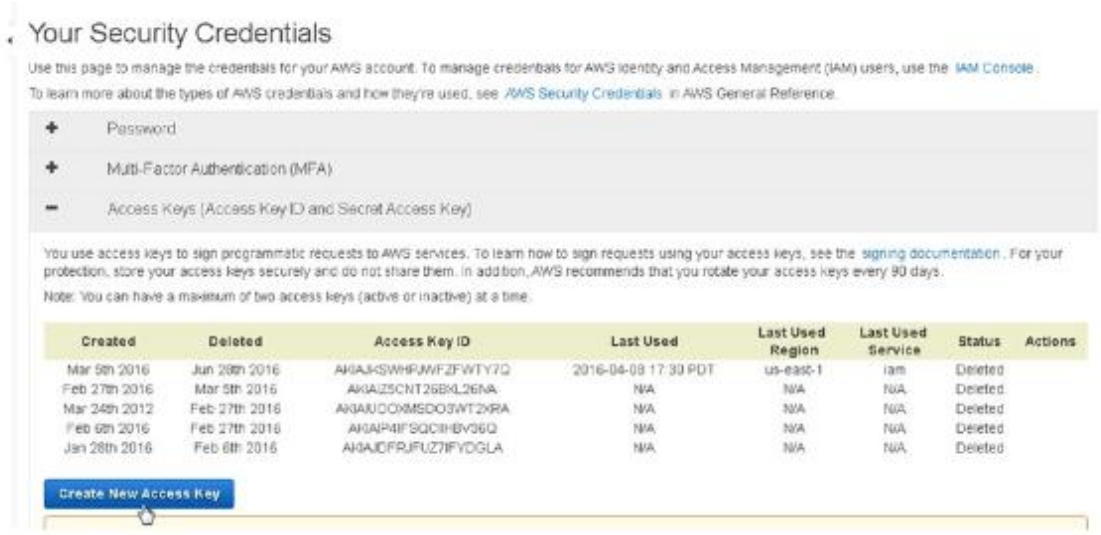
Gambaran Umum

Dalam bab ini, kita akan membuat kluster multi-simpul (terdiri dari satu master dan beberapa minion) di Amazon *Elastic Compute Cloud* (EC2) menggunakan AWS Command Line Interface. Tahapannya adalah sebagai berikut:

1. Menetapkan Lingkungan
2. Memulai Kluster
3. Menguji Kluster
4. Mengonfigurasi Kluster
5. Menghentikan Kluster

Mengatur Lingkungan

Karena kita menggunakan Amazon EC2, akun AWS diperlukan. Selain itu, untuk mengonfigurasi AWS, Anda perlu memperoleh kredensial keamanan. Pilih Kredensial Keamanan untuk akun pengguna. Di layar Kredensial Keamanan Anda, pilih simpul Kunci Akses dan klik Buat Kunci Akses Baru seperti yang ditunjukkan pada Gambar 1.2 untuk membuat kunci akses baru.



Gambar 1.2 Membuat Kunci Akses Baru

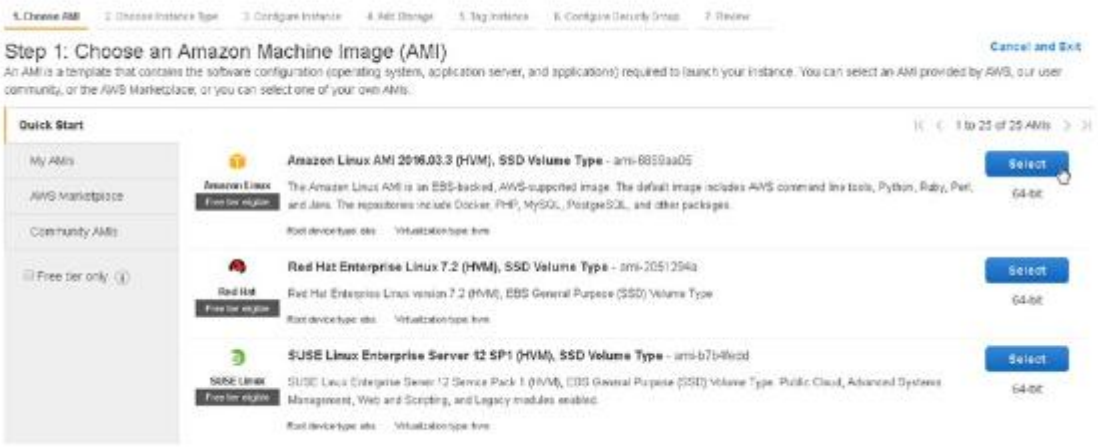
Kunci akses keamanan baru dibuat dan Access Key ID dan Secret Access Key dicantumkan. Salin Access Key ID dan Secret Access Key yang akan digunakan nanti untuk mengonfigurasi AWS. Access Key ID dan Secret Access Key akan berbeda untuk setiap pengguna.

```
AWS_ACCESS_KEY_ID AKIAISQVxxxxxxxxxxxxxxx
AWS_SECRET_ACCESS_KEY VuJD5gDxxxxxxxxxxxxxxxxxxxxx
```

Karena AWS Command Line Interface diperlukan, buat instans EC2 dari Amazon Linux Amazon Machine Image (AMI), yang telah terinstal AWS CLI. Klik Launch Instance seperti yang ditunjukkan pada Gambar 1.2 untuk membuat instans baru.

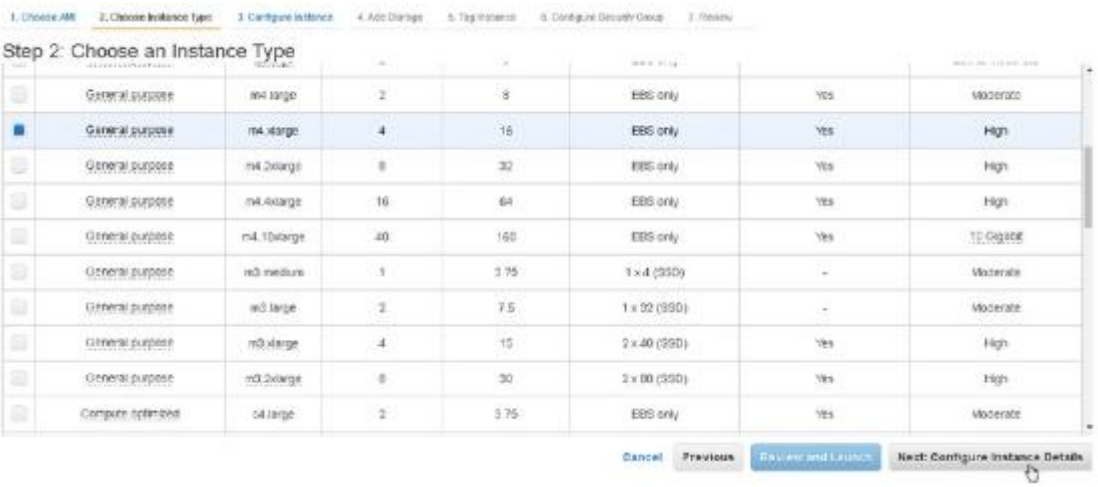
Gambar 1.3 Meluncurkan instans EC2

Pada layar berikutnya, pilih Amazon Linux AMI (64 bit) seperti yang ditunjukkan pada Gambar 1.4.



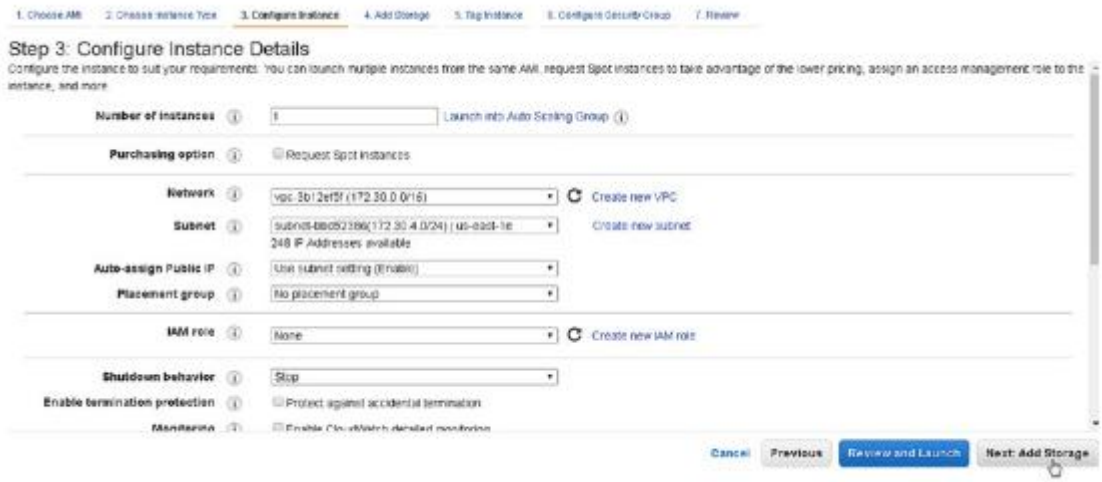
Gambar 1.4 Memilih Amazon Linux AMI

Untuk Jenis Instans, pilih ukuran Instans yang relatif besar (m4.xlarge) seperti yang ditunjukkan pada Gambar 1.5, karena ukuran mikro default (Tingkat Gratis) mungkin tidak menyediakan memori atau ruang disk yang cukup untuk menginstal Kubernetes. Beberapa jenis instans seperti m4.xlarge mungkin hanya diluncurkan di virtual private cloud (VPC). Jika Anda sudah siap, klik Berikutnya: Konfigurasikan Detail Instans.



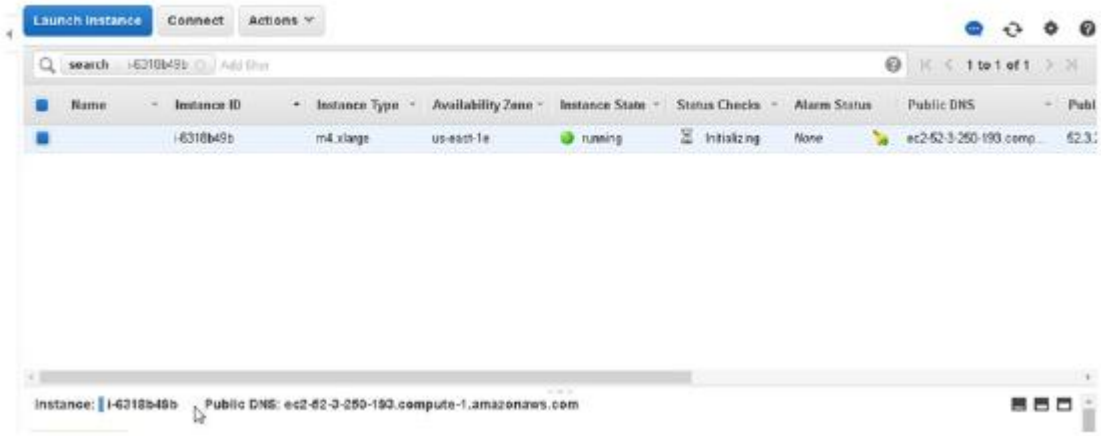
Gambar 1.5 Memilih Jenis Instans

Tentukan detail instans seperti Network VPC dan Subnet seperti yang ditunjukkan pada Gambar 1.6. Setelah selesai, klik Next: Add Storage.



Gambar 1.6 Mengonfigurasi Detail Instans

Instans EC2 baru dibuat. Dapatkan DNS Publik untuk instans seperti yang ditunjukkan pada Gambar 1.7.

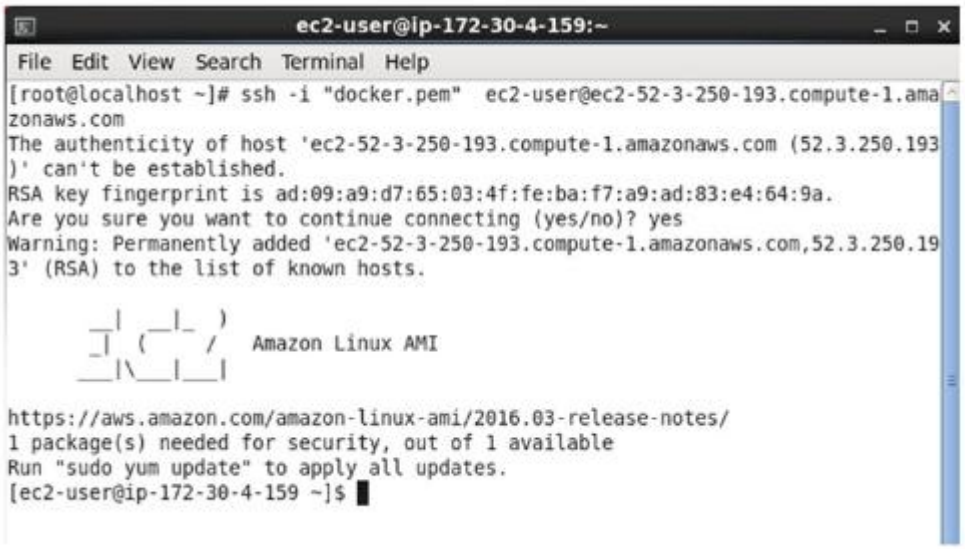


Gambar 1.7 DNS Publik

Dengan menggunakan kunci pribadi yang ditentukan saat instans dibuat, log masuk SSH ke instans:

```
ssh -i "docker.pem" ec2-user@ec2-52-3-250-193.compute-
1.amazonaws.com
```

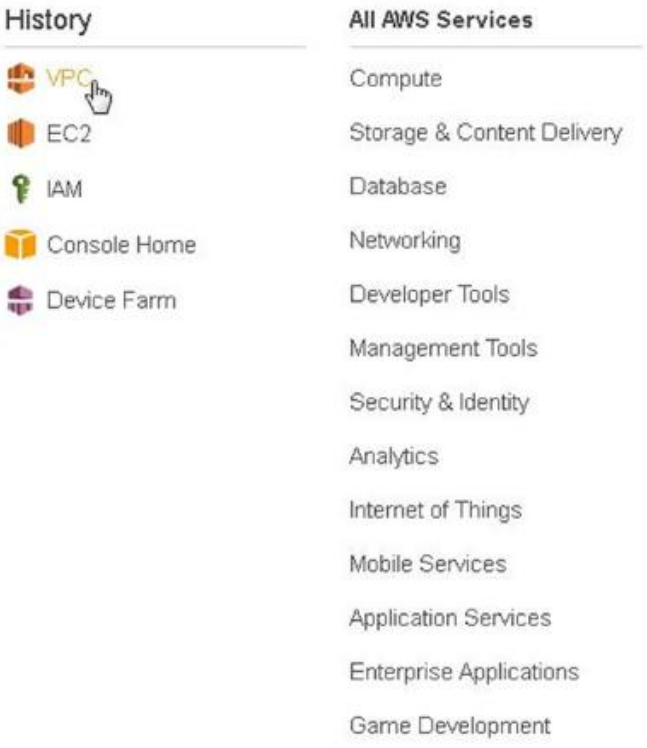
Prompt perintah Amazon Linux ditampilkan seperti yang ditunjukkan pada Gambar 1.8.



Gambar 1.8 Prompt Perintah Amazon Linux AMI

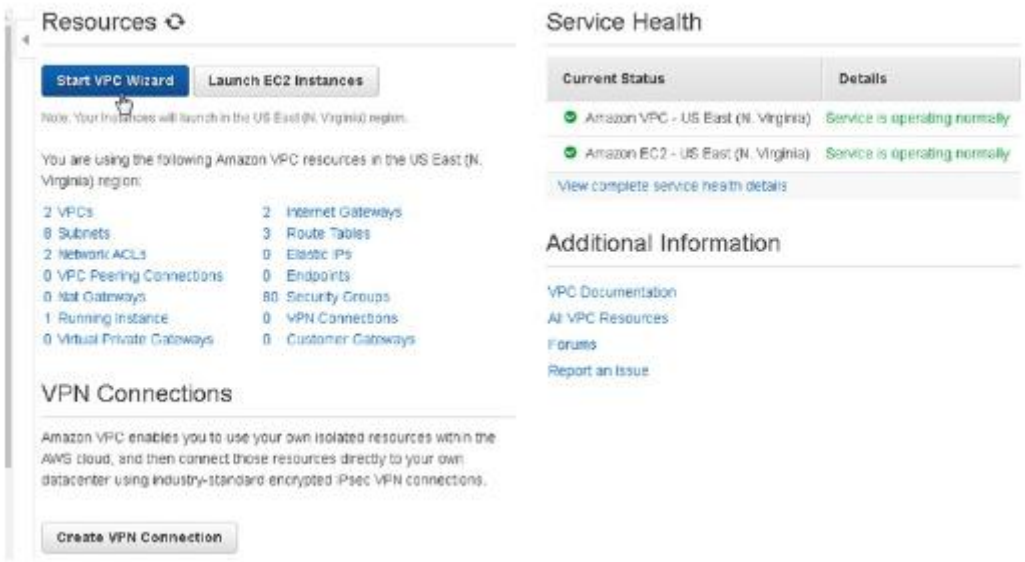
1.3 MENGONFIGURASI AWS

Saat klaster Kubernetes dimulai di AWS EC2, VPC baru dibuat untuk node master dan minion. Jumlah VPC yang dapat dibuat di akun AWS memiliki batas, yang dapat bervariasi untuk setiap pengguna. Sebelum memulai klaster, hapus VPC yang tidak digunakan sehingga batas tidak tercapai saat VPC baru dibuat. Untuk memulai, pilih VPC di Layanan AWS seperti yang ditunjukkan pada Gambar 1.9.



Gambar 1.9 Memilih Konsol VPC

Klik Start VPC Wizard seperti yang ditunjukkan pada Gambar 1.10 untuk membuat daftar dan menghapus VPC jika diperlukan.



Gambar 1.10 Memulai VPC Wizard

VPC yang sudah tersedia tercantum seperti yang ditunjukkan pada Gambar 1.11.

Gambar 1.11 VPC Yang Tersedia

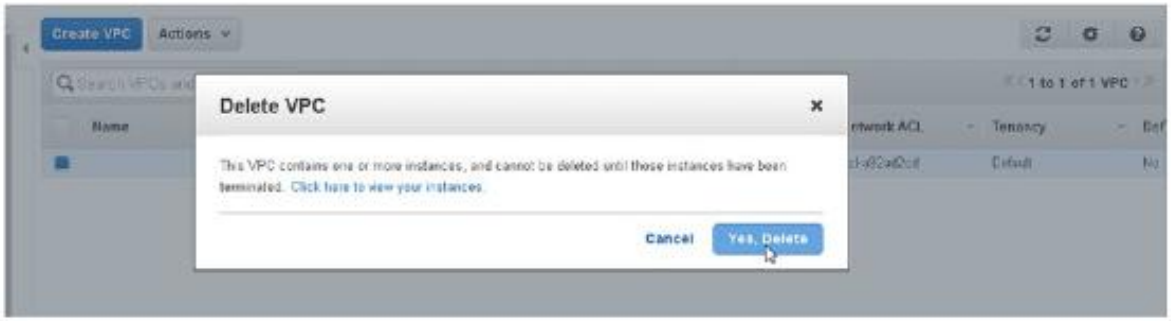
Untuk menghapus VPC, pilih VPC dan klik Tindakan ➤ Hapus VPC, seperti yang ditunjukkan pada Gambar 1.12.

Gambar 1.12 Memilih Tindakan ➤ Hapus VPC

Pada layar konfirmasi yang muncul, klik Ya, Hapus. Jika VPC tidak dikaitkan dengan instans mana pun, VPC akan mulai dihapus seperti yang ditunjukkan pada Gambar 1.13.

Gambar 1.13 Menghapus VPC

Jika VPC dikaitkan dengan instans mana pun, maka VPC tersebut tidak dapat dihapus dan tombol Ya, Hapus tidak tersedia, seperti yang ditunjukkan pada Gambar 1.14.



Gambar 1.14 Pesan Untuk VPC Yang Tidak Dapat Dihapus

Berikutnya, konfigurasikan AWS pada instans Amazon Linux menggunakan perintah berikut:

```
aws configure
```

Saat diminta, tentukan AWS Access Key ID dan AWS Access Key. Tentukan juga nama wilayah default (us-east-1) dan format output default (json) seperti yang ditunjukkan pada Gambar 1.15.

```
[ec2-user@ip-172-30-4-159 ~]$ aws configure
AWS Access Key ID [None]: AKIAISQVST
AWS Secret Access Key [None]: VuJD5gD
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-30-4-159 ~]$
```

Gambar 1.15 Mengonfigurasi AWS

1.4 MEMULAI KLUSTER KUBERNETES

Sekarang setelah Anda mengonfigurasi AWS, jalankan perintah berikut untuk menginstal Kubernetes:

```
export KUBERNETES_PROVIDER=aws; wget -q -O - https://get.k8s.io
| bash
```

Perintah ini memulai proses instalasi Kubernetes seperti yang ditunjukkan pada Gambar 1.15.

```
[ec2-user@ip-172-30-4-159 ~]$ aws configure
AWS Access Key ID [None]: AKIAISQVST
AWS Secret Access Key [None]: VuJD5g
Default region name [None]: us-east-1
Default output format [None]: json
[ec2-user@ip-172-30-4-159 ~]$ export KUBERNETES_PROVIDER=aws; wget -q -O - https
://get.k8s.io | bash
```

Gambar 1.16 Menginstal Kubernetes

```
Sleeping for 3 seconds...
Waiting for instance i-04978291 to be running (currently pending)
Sleeping for 3 seconds...
Waiting for instance i-04978291 to be running (currently pending)
Sleeping for 3 seconds...
 [master running]
Attaching IP 50.112.79.71 to instance i-04978291
Attaching persistent data volume (vol-c026ee75) to master
2016-06-28T18:42:25.708Z        /dev/sdb        i-04978291      attaching       v
ol-c026ee75
cluster "aws_kubernetes" set.
user "aws_kubernetes" set.
context "aws_kubernetes" set.
switched to context "aws_kubernetes".
user "aws_kubernetes-basic-auth" set.
Wrote config for aws_kubernetes to /home/ec2-user/.kube/config
Creating minion configuration
Creating autoscaling group
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 0 minions started; waiting
 4 minions started; ready
Waiting for cluster initialization.

  This will continually check to see if the API for kubernetes is reachable.
  This might loop forever if there was some uncaught error during start
  up.

.....
```

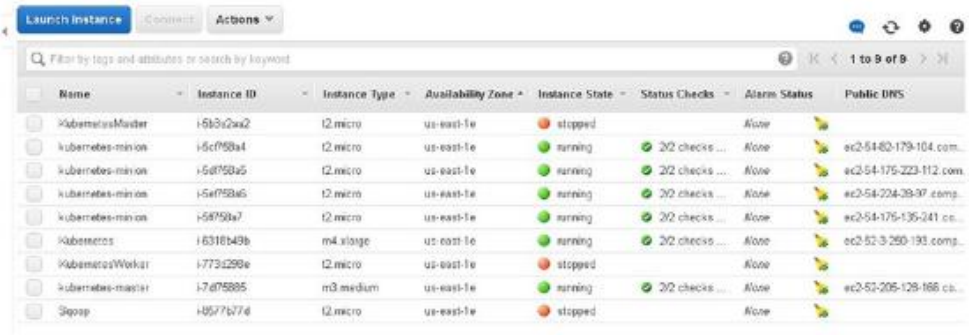
Gambar 1.17 Memulai Master Dan Minion

Perintah sebelumnya memanggil skrip cluster/kube-up.sh, yang selanjutnya memanggil skrip cluster/aws/util.sh menggunakan konfigurasi yang ditentukan dalam skrip cluster/aws/config-default.sh. Satu master dan empat minion dimulai pada Debian 8 (jessie) seperti yang ditunjukkan pada Gambar 1.17. Inisialisasi klaster dimulai kemudian. Klaster dimulai dan divalidasi, dan komponen yang diinstal dicantumkan. URL tempat Kubernetes master, Elasticsearch, Heapster, dan layanan lainnya berjalan dicantumkan seperti yang ditunjukkan pada Gambar 1.18. Jalur direktori tempat biner Kubernetes diinstal juga dicantumkan.

```
Flag --api-version has been deprecated, flag is no longer respected and will be
deleted in the next release
Validate output:
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
Cluster validation succeeded
Done, listing cluster services:

Kubernetes master is running at https://50.112.79.71
Elasticsearch is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-sy
stem/services/elasticsearch-logging
Heapster is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-system/
services/heapster
Kibana is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-system/se
rvices/kibana-logging
KubeDNS is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-system/s
ervices/kube-dns
kubernetes-dashboard is running at https://50.112.79.71/api/v1/proxy/namespaces/
kube-system/services/kubernetes-dashboard
Grafana is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-system/s
ervices/monitoring-grafana
InfluxDB is running at https://50.112.79.71/api/v1/proxy/namespaces/kube-system/
services/monitoring-influxdb

Kubernetes binaries at /home/ec2-user/kubernetes/cluster/
You may want to add this directory to your PATH in $HOME/.profile
Installation successful!
[ec2-user@ip-172-30-4-159 ~]$
```

Gambar 1.18 Kubernetes Dan Komponen Yang Dimulai

Satu node kubernetes-master dan empat node kubernetes-minion yang dimulai tercantum di konsol EC2 seperti yang ditunjukkan pada Gambar 1.19.



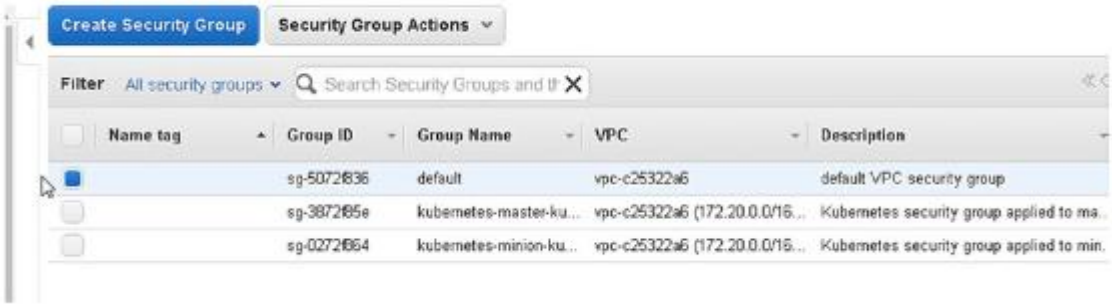
Gambar 1.19 Instans EC2 Kubernetes-Master Dan Kubernetes-Minion

Informasi klaster dapat diperoleh dengan perintah kubectl cluster-info, seperti yang ditunjukkan pada Gambar 1.20.

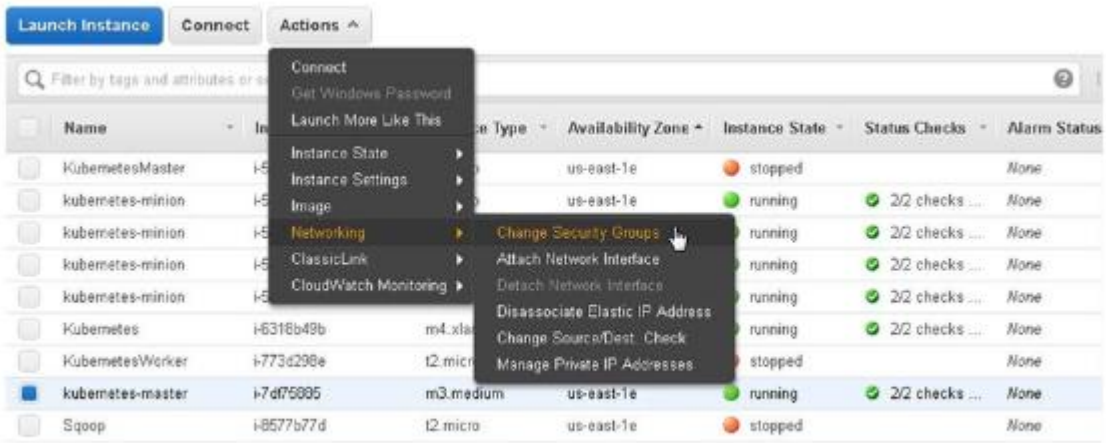
```
[ec2-user@ip-172-30-4-159 ~]$ kubectl cluster-info
Kubernetes master is running at https://52.205.128.166
Elasticsearch is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-
system/services/elasticsearch-logging
Heapster is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-syste
m/services/heapster
Kibana is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-system/
services/kibana-logging
KubeDNS is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-system
/services/kube-dns
kubernetes-dashboard is running at https://52.205.128.166/api/v1/proxy/namespace
s/kube-system/services/kubernetes-dashboard
Grafana is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-system
/services/monitoring-grafana
InfluxDB is running at https://52.205.128.166/api/v1/proxy/namespaces/kube-syste
m/services/monitoring-influxdb
[ec2-user@ip-172-30-4-159 ~]$
```

Gambar 1.20 Menjalankan perintah kubectl cluster-info

Instans yang berbeda perlu saling mengakses. Grup keamanan dibuat untuk setiap tipe instans, master dan minion, seperti yang ditunjukkan pada Gambar 1.21.

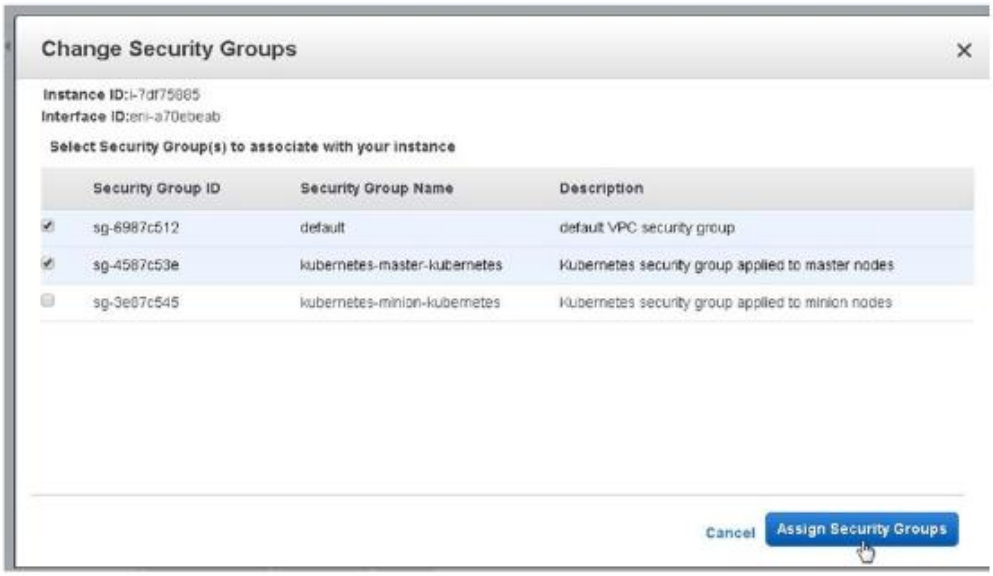


Gambar 1.21 Grup keamanan



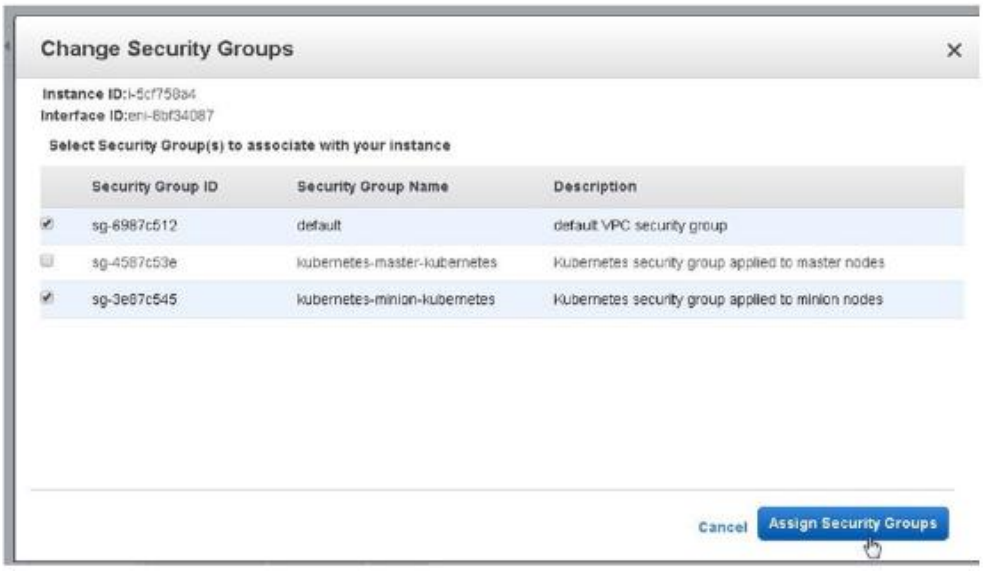
Gambar 1.22 Memilih Tindakan ➤ Jaringan ➤ Ubah Grup Keamanan

Untuk menambahkan semua lalu lintas antar instans, tambahkan grup keamanan default ke grup keamanan untuk master dan minion; grup keamanan default mengizinkan semua lalu lintas dari semua protokol dari semua sumber. Untuk menambahkan grup keamanan ke instans (misalnya kubernetes-master), pilih instans. Kemudian pilih Tindakan ➤ Jaringan ➤ Ubah Grup Keamanan seperti yang ditunjukkan pada Gambar 1.22. Di layar Ubah Grup Keamanan, pilih grup keamanan default selain grup keamanan yang ditetapkan ke node master dan klik Tetapkan Grup Keamanan seperti yang ditunjukkan pada Gambar 1.23.



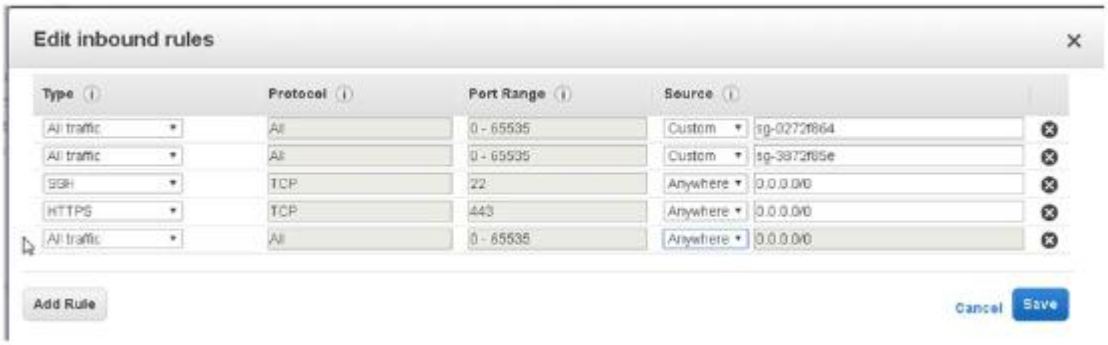
Gambar 1.23 Menetapkan grup keamanan untuk kubernetes-master

Demikian pula, untuk setiap node kubernetes-minion, tambahkan grup keamanan default dan klik Tetapkan Grup Keamanan seperti yang ditunjukkan pada Gambar 1.24.



Gambar 1.24 Menetapkan grup keamanan untuk kubernetes-minion

Atau, jika grup keamanan default dimodifikasi untuk tidak mengizinkan semua lalu lintas, grup keamanan yang ditetapkan ke kubernetes-master dan setiap grup keamanan kubernetes-minion harus menyertakan aturan masuk ke semua lalu lintas, seperti yang ditunjukkan pada Gambar 1.25.



Gambar 1.25 Grup Keamanan Dengan Aturan Masuk Untuk Mengizinkan Semua Lalu Lintas

1.5 MENGUJI KLUSTER

Selanjutnya, kita akan menguji kluster Kubernetes. Pertama, kita perlu menambahkan jalur direktori tempat biner Kubernetes diinstal ke variabel lingkungan PATH.

```
export PATH=/home/ec2-
user/kubernetes/platforms/linux/amd64:$PATH
```

Kemudian echo variabel lingkungan PATH seperti yang ditunjukkan pada Gambar 1.26.

```
[ec2-user@ip-172-30-4-159 kubernetes]$ export PATH=/home/ec2-user/kubernetes/pla
tforms/linux/amd64:$PATH
[ec2-user@ip-172-30-4-159 kubernetes]$ cd ~
[ec2-user@ip-172-30-4-159 ~]$ echo $PATH
/home/ec2-user/kubernetes/platforms/linux/amd64:/usr/local/bin:/bin:/usr/bin:/us
r/local/sbin:/usr/sbin:/sbin:/opt/aws/bin:/home/ec2-user/.local/bin:/home/ec2-us
er/bin
```

Gambar 1.26 Menetapkan variabel lingkungan PATH

- Untuk menguji kluster, jalankan citra Docker seperti citra nginx untuk membuat tiga replika pod: `kubectl run nginx --image=nginx --replicas=3 --port=80`
- Daftar pod: `kubectl get pods`
- Daftar deployment: `kubectl get deployments`
- Buat layanan bertipe LoadBalancer untuk deployment: `kubectl exposed deployment nginx --port=80 --type=LoadBalancer`
- Daftar layanan: `kubectl get services`
- Daftar pod di seluruh kluster: `kubectl get pods –o wide`

Jika Kubernetes telah terinstal dengan benar, semua perintah sebelumnya akan berjalan dengan benar dan menghasilkan output untuk menunjukkan bahwa kluster pod telah dibuat, seperti yang ditunjukkan pada Gambar 1.27.

```
[ec2-user@ip-172-30-4-159 ~]$ kubectl run nginx --image=nginx --replicas=3 --por
t=80
deployment "nginx" created
[ec2-user@ip-172-30-4-159 ~]$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-198147104-de6gq    1/1       Running   0          23s
nginx-198147104-q322k    1/1       Running   0          23s
nginx-198147104-u3sah    1/1       Running   0          23s
[ec2-user@ip-172-30-4-159 ~]$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     3         3         3            3           45s
[ec2-user@ip-172-30-4-159 ~]$ kubectl expose deployment nginx --port=80 --type=L
oadBalancer
service "nginx" exposed
[ec2-user@ip-172-30-4-159 ~]$ kubectl get services
NAME         CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1       <none>        443/TCP   55m
nginx        10.0.165.153                 80/TCP    22s
[ec2-user@ip-172-30-4-159 ~]$ kubectl get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       NODE
nginx-198147104-de6gq    1/1       Running   0          2m        ip-172-20-0-175
.us-west-2.compute.internal
nginx-198147104-q322k    1/1       Running   0          2m        ip-172-20-0-174
.us-west-2.compute.internal
nginx-198147104-u3sah    1/1       Running   0          2m        ip-172-20-0-175
.us-west-2.compute.internal
[ec2-user@ip-172-30-4-159 ~]$
```

Gambar 1.27 Membuat Kluster Pod Untuk Nginx

Mengonfigurasi Kluster

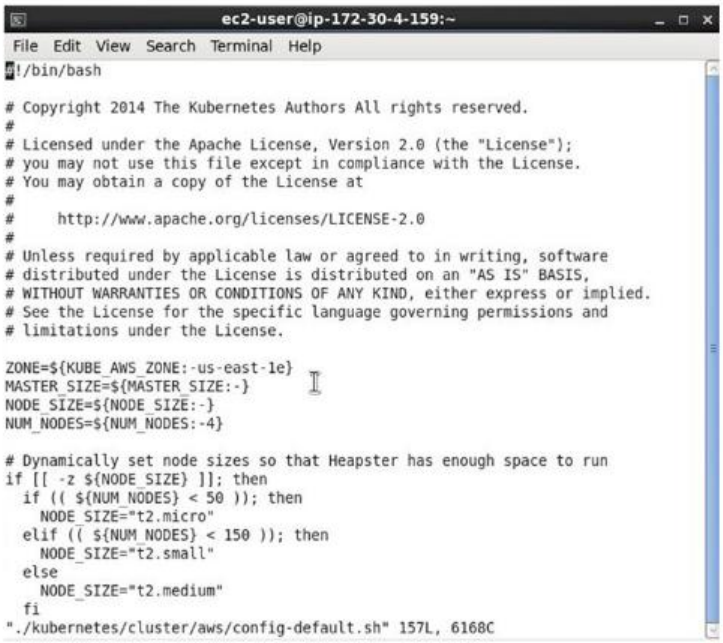
Pengaturan konfigurasi default yang digunakan untuk memulai kluster baru ditetapkan dalam file cluster/aws/config-default.sh. Konfigurasi default mencakup pengaturan untuk zona AWS, jumlah node, ukuran master, ukuran node, wilayah AWS S3, AWS S3 Bucket, dan awalan instans.

```
export KUBE_AWS_ZONE=eu-west-1c
export NUM_NODES=3
export MASTER_SIZE=m3.medium
export NODE_SIZE=m3.medium
export AWS_S3_REGION=eu-west-1
export AWS_S3_BUCKET=mycompany-kubernetes-artifacts
export INSTANCE_PREFIX=k8s
```

File config-default.sh dapat dibuka di editor vi:

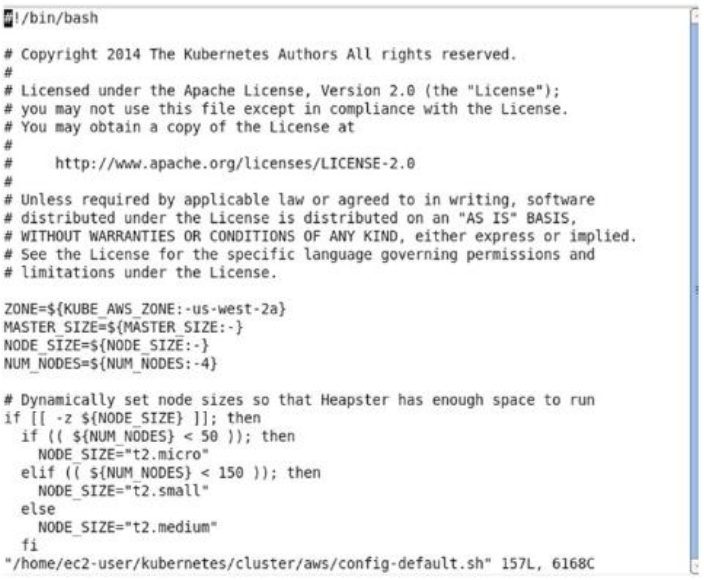
```
sudo vi /home/ec2-user/kubernetes/cluster/aws/config-default.sh
```

Pengaturan konfigurasi tercantum seperti yang ditunjukkan pada Gambar 1.28.



Gambar 1.28 Mencantumkan Pengaturan Konfigurasi Default

Sebagai contoh, ubah zona AWS dari us-east-1e ke us-west-2a seperti yang ditunjukkan pada Gambar 1.29.



Gambar 1.29 Memodifikasi Zona AWS

Matikan klaster setelah melakukan modifikasi apa pun:

```
/home/ec2-user/kubernetes/cluster/kube-down.sh
```

Mulai ulang klaster:

```
/home/ec2-user/kubernetes/cluster/kube-up.sh
```

Klaster harus dimulai di zona us-west-2a seperti yang ditunjukkan pada Gambar 1.30.

Gambar 1.30 Kluster Kubernetes Yang Dimulai Ulang Dengan Node Di Zona AWS Baru

Menghentikan Kluster

Untuk menghentikan kluster, jalankan perintah kube-down.sh:

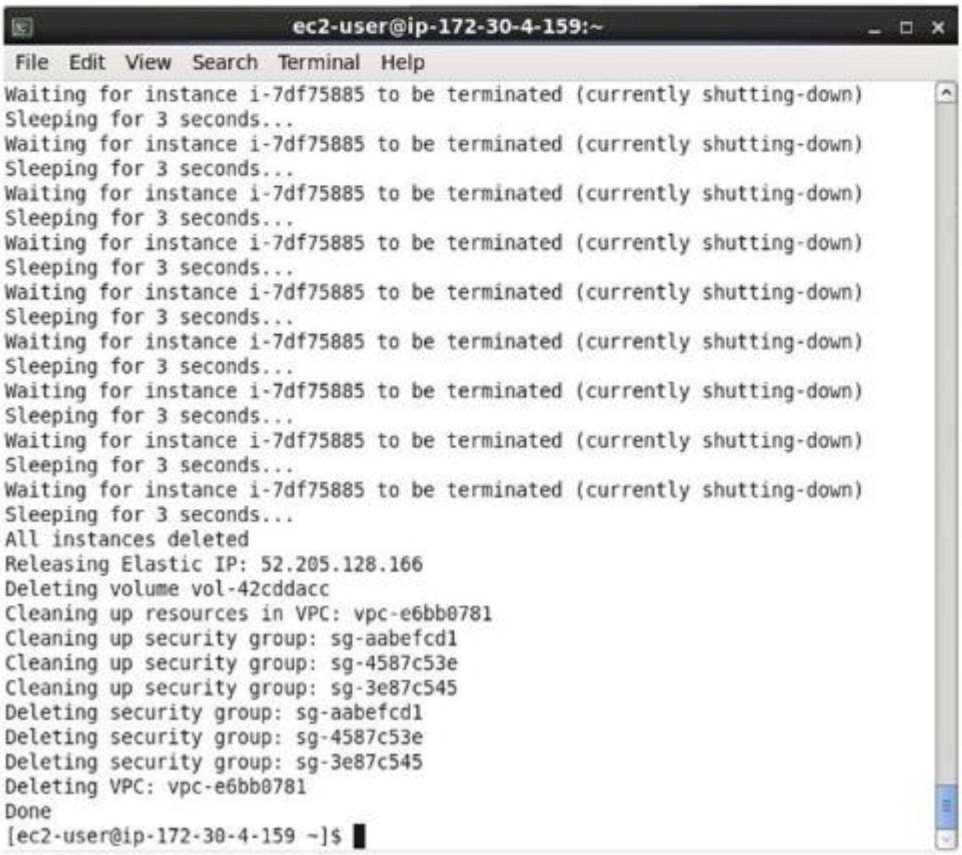
```
/home/ec2-user/kubernetes/cluster/kube-down.sh
```

Seperti yang ditunjukkan pada output pada Gambar 1.30, ELB di VPC dihapus, instans di VPC dihapus, dan grup penskalaan otomatis serta konfigurasi peluncuran otomatis dihapus.

```
[ec2-user@ip-172-30-4-159 ~]$ /home/ec2-user/kubernetes/cluster/kube-down.sh
Bringing down cluster using provider: aws
Deleting ELBs in: vpc-e6bb0781
Waiting for ELBs to be deleted
All ELBs deleted
Deleting instances in VPC: vpc-e6bb0781
Deleting auto-scaling group: kubernetes-minion-group-us-east-1e
Deleting auto-scaling launch configuration: kubernetes-minion-group-us-east-1e
Deleting auto-scaling group: kubernetes-minion-group-us-east-1e
Deleting auto-scaling group: kubernetes-minion-group-us-east-1e
Deleting auto-scaling group: kubernetes-minion-group-us-east-1e
Waiting for instances to be deleted
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
Sleeping for 3 seconds...
Waiting for instance i-7df75885 to be terminated (currently shutting-down)
```

Gambar 1.31 Menghentikan Kluster Kubernetes

Setelah semua instans dihapus, IP elastis dilepaskan, dan akhirnya grup keamanan dan VPC dihapus seperti yang ditunjukkan pada Gambar 1.32.



Gambar 1.32 Menghapus instans, volume, grup keamanan, dan VPC

Selanjutnya, klaster dapat dimulai ulang jika diperlukan.

```
/home/ec2-user/kubernetes/cluster/kube-up.sh
```

Ringkasan

Dalam bab ini, kami membahas cara menginstal Kubernetes di AWS. Amazon Linux AMI harus digunakan karena telah menginstal AWS CLI. Terlalu banyak VPC tidak boleh dibuat sebelum membuat klaster Kubernetes, karena VPC baru dibuat saat klaster dibuat dan memiliki terlalu banyak VPC sebelumnya dapat menyebabkan kuota VPC yang dialokasikan ke akun pengguna terlampaui. Kami membuat klaster Kubernetes yang terdiri dari satu master dan tiga minion. Dalam bab berikutnya, kami akan menginstal Kubernetes di CoreOS, OS Linux yang dirancang khusus untuk kontainer.

BAB 2
KUBERNETES PADA COREOS DI AWS

2.1 KLUSTER KUBERNETES DENGAN COREOS DAN AWS

Kubernetes biasanya digunakan dengan platform cloud, karena infrastruktur perangkat keras yang diperlukan untuk kluster Kubernetes multi-simpul paling baik disediakan di lingkungan cloud. Di Bab 1, kami menggunakan alat kube-up untuk menjalankan kluster tanpa memerlukan konfigurasi apa pun.

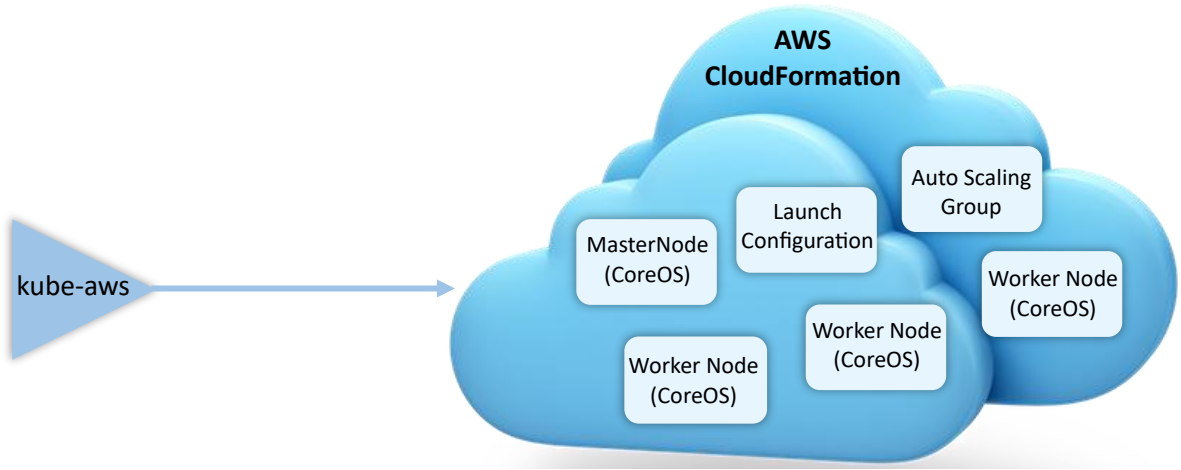
Masalah

Alat kube-up tidak membuat kluster yang siap produksi. Salah satu keterbatasan kube-up adalah tidak mendukung CoreOS. Docker harus diinstal, meskipun instalasi Docker telah dikonfigurasi sebelumnya.

Solusi

Docker diinstal secara langsung di CoreOS. Alat CoreOS kube-aws dapat digunakan untuk menjalankan kluster Kubernetes yang siap produksi pada simpul CoreOS di EC2 tanpa banyak konfigurasi. Alat kube-aws menggunakan AWS CloudFormation untuk membuat kluster instans EC2 yang menjalankan CoreOS. AWS CloudFormation adalah layanan untuk menyediakan sumber daya AWS seperti instans EC2, Auto Scaling Group, dan penyeimbang beban Elastic Load Balancing, semuanya menggunakan templat.

Menggunakan satu berkas konfigurasi kluster untuk menyediakan AWS CloudFormation bagi kluster Kubernetes merupakan pola desain manajemen. Templat kluster yang sama dapat digunakan kembali untuk menyediakan kluster Kubernetes lainnya. Gambar 2.1 menunjukkan AWS CloudFormation yang terdiri dari satu simpul Master Kubernetes dan tiga simpul pekerja Kubernetes, beserta satu Auto Scaling Group dan satu Konfigurasi Peluncuran.



Gambar 2.1 AWS Cloudformation Untuk Kluster Kubernetes

Gambaran Umum

AWS CloudFormation menyediakan kumpulan sumber daya AWS berdasarkan templat, yang menentukan sumber daya dan dependensi. Dalam bab ini, kami akan menerapkan AWS CloudFormation untuk kluster instans CoreOS yang menjalankan Kubernetes. Kami akan menggunakan AWS Launch Configuration dan Scaling Group untuk meluncurkan dan menskalakan CloudFormation secara otomatis. Alat CLI generator CloudFormation kube-aws digunakan untuk membuat tumpukan CloudFormation dari templat. Tahapan yang akan kami jelajahi adalah sebagai berikut:

- Menetapkan Lingkungan
- Mengonfigurasi Kredensial AWS
- Menginstal kube-aws
- Menetapkan Parameter Kluster
- Membuat Kunci KMS
- Menetapkan Nama DNS Eksternal
- Membuat CloudFormation Kluster
- Membuat Direktori Aset
- Menginisialisasi CloudFormation Kluster
- Merender Konten Direktori Aset
- Menyesuaikan Kluster
- Memvalidasi Tumpukan CloudFormation
- Meluncurkan Cluster CloudFormation
- Mengonfigurasi DNS
- Mengakses Cluster
- Menguji Cluster

Mengatur Lingkungan

Perangkat lunak berikut diperlukan untuk bab ini:

- AWS Command Line Interface (CLI)
- kube-aws CloudFormation Generator

Untuk mengatur lingkungan Anda, pertama-tama buat instans Amazon EC2 dari Amazon Linux AMI (ami-7172b611), yang telah terinstal AWS CLI. Ubah aturan Inbound/Outbound untuk mengizinkan semua lalu lintas untuk semua protokol dalam rentang port 0–65535 dari sumber mana pun dan ke tujuan mana pun. Dapatkan IP Publik instans EC2. SSH Masuk ke instans EC2:

```
ssh -i kubernetes-coreos.pem ec2-user@54.86.194.192
```

Prompt perintah Amazon Linux ditampilkan. Anda sekarang siap untuk memulai.

2.2 MENGONFIGURASI KREDENSIAL AWS

Kita perlu membuat satu set kredensial Keamanan AWS, yang akan kita gunakan untuk mengonfigurasi instans EC2 tempat tumpukan CloudFormation diluncurkan. Kredensial Keamanan AWS yang digunakan dalam Bab 1 dapat digunakan jika tidak dihapus. Untuk

membuat kredensial Keamanan AWS baru, klik Kredensial Keamanan untuk akun pengguna dan klik Buat Kunci Akses Baru untuk membuat kunci akses. Salin ID Kunci Akses dan kunci akses. Di instans Amazon Linux, jalankan perintah berikut untuk mengonfigurasi instans dengan kredensial AWS:

```
aws configure
```

Tentukan ID kunci akses dan kunci akses saat diminta. Tentukan nama wilayah default (us-east-1) dan format output (json).

Menginstal Kube-aws

Aplikasi CoreOS di GitHub dan dikemas ke dalam gambar AppC ditandatangani dengan Kunci Penandatanganan Aplikasi CoreOS. Agar Anda dapat mendistribusikan pekerjaan Anda sendiri, impor Kunci Penandatanganan Aplikasi CoreOS, seperti yang ditunjukkan di sini:

```
gpg2 --keyserver pgp.mit.edu --recv-key FC8A365E
```

Selanjutnya, validasi kunci:

```
gpg2 --fingerprint FC8A365E
```

Gambar 2.2 menunjukkan output dari perintah ini. Seperti yang dapat Anda lihat, sidik jari kunci adalah 18AD 5014 C99E F7E3 BA5F 6CE9 50BD D3E0 FC8A 365E, yang merupakan sidik jari kunci yang benar; nilainya adalah konstanta.

```
[ec2-user@ip-172-30-1-188 ~]$ gpg2 --keyserver pgp.mit.edu --recv-key FC8A365E
gpg: directory `/home/ec2-user/.gnupg' created
gpg: new configuration file `/home/ec2-user/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/ec2-user/.gnupg/gpg.conf' are not yet active dur
ing this run
gpg: keyring `/home/ec2-user/.gnupg/secring.gpg' created
gpg: keyring `/home/ec2-user/.gnupg/pubring.gpg' created
gpg: requesting key FC8A365E from hkp server pgp.mit.edu
gpg: /home/ec2-user/.gnupg/trustdb.gpg: trustdb created
gpg: key FC8A365E: public key "CoreOS Application Signing Key <security@coreos.c
om>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
[ec2-user@ip-172-30-1-188 ~]$ gpg2 --fingerprint FC8A365E
pub   4096R/FC8A365E 2016-03-02 [expires: 2021-03-01]
      Key fingerprint = 18AD 5014 C99E F7E3 BA5F  6CE9 50BD D3E0 FC8A 365E
uid       [ unknown] CoreOS Application Signing Key <security@coreos.com>
sub   2048R/3F1B2C87 2016-03-02 [expires: 2019-03-02]
sub   2048R/BEDDBA18 2016-03-08 [expires: 2019-03-08]
sub   2048R/7EF48FD3 2016-03-08 [expires: 2019-03-08]

[ec2-user@ip-172-30-1-188 ~]$
```

Gambar 2.2 Mengimpor Dan Memvalidasi Kunci Penandatanganan Aplikasi Coreos

Unduh tarball rilis terbaru dan tanda tangan terpisah (.sig) untuk kube-aws dari https://github.com/coreos/coreos-kubernetes/releases:

```
wget https://github.com/coreos/coreos-
kubernetes/releases/download/v0.7.1/kube-aws-linux-
amd64.tar.gz
wget https://github.com/coreos/coreos-
kubernetes/releases/download/v0.7.1/kube-aws-linux-
amd64.tar.gz.sig
```

Validasi tanda tangan GPG tarball.

```
gpg2 --verify kube-aws-linux-amd64.tar.gz.sig kube-aws-linux-
                        amd64.tar.gz
```

Sidik jari kunci utama harus berupa 18AD 5014 C99E F7E3 BA5F 6CE9 50BD D3E0 FC8A 365E, seperti yang ditunjukkan pada Gambar 2.3.

Gambar 2.3 Validasi Tanda Tangan GPG Tarball

1. Ekstrak biner dari berkas tar.gz: tar zxvf kube-aws-linux-amd64.tar.gz
2. Tambahkan kube-aws ke jalur: sudo mv linux-amd64/kube-aws /usr/local/bin

Generator CloudFormation kube-aws telah terinstal. Anda dapat menampilkan informasi tentang penggunaannya dengan perintah kube-aws –help.

Menyiapkan Parameter Klaster

Sebelum menginisialisasi dan meluncurkan klaster AWS CloudFormation, kita perlu membuat atau menentukan parameter klaster berikut:

- Pasangan kunci EC2
- Kunci KMS
- Nama DNS eksternal

Sebelum membuat pasangan kunci, kita perlu mengonfigurasi wilayah AWS; kita telah melakukannya dengan perintah aws configure. Jalankan perintah berikut untuk membuat

pasangan kunci yang disebut kubernetes-coreos dan simpan sebagai kubernetes-coreos.pem:

```
aws ec2 create-key-pair --key-name kubernetes-coreos --query
'KeyMaterial' --output text > kubernetes-coreos.pem
```

Ubah izin akses pasangan kunci menggunakan mode 400, yang menetapkan izin akses untuk dibaca oleh pemilik.

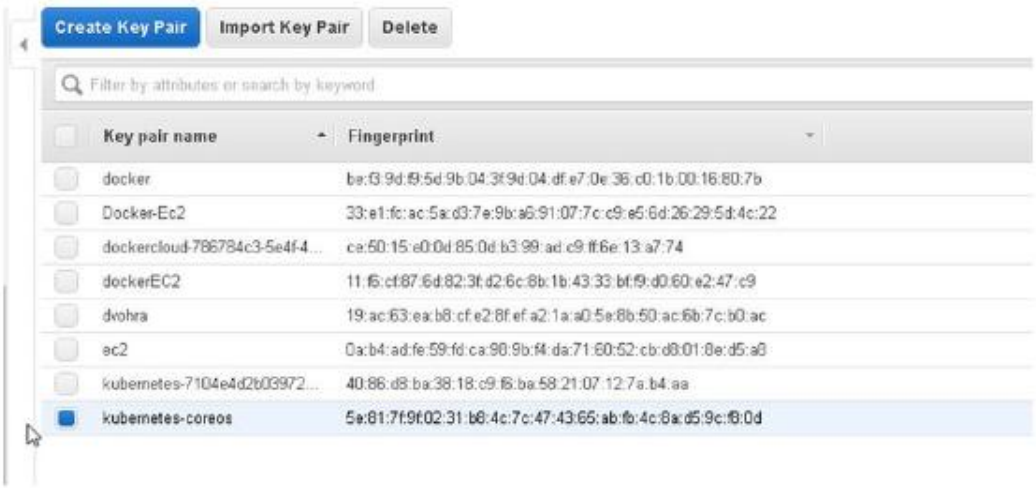
```
chmod 400 kubernetes-coreos.pem
```

Pasangan kunci dibuat dan izin akses ditetapkan seperti yang ditunjukkan pada Gambar 2-4.

```
[ec2-user@ip-172-30-1-188 ~]$ aws ec2 create-key-pair --key-name kubernetes-core
os --query 'KeyMaterial' --output text > kubernetes-coreos.pem
[ec2-user@ip-172-30-1-188 ~]$ chmod 400 kubernetes-coreos.pem
```

Gambar 2.4 Membuat Pasangan Kunci

Pada konsol AWS, pasangan kunci kubernetes-coreos harus dicantumkan, seperti yang ditunjukkan pada Gambar 2.5.



Gambar 2.5 Mencantumkan Pasangan Kunci Di Konsol EC2

Membuat Kunci KMS

Selanjutnya, buat kunci KMS, yang digunakan untuk mengenkripsi dan mendekripsi aset TLS kluster dan diidentifikasi oleh string Amazon Resource Name (ARN). Gunakan aws CLI untuk membuat kunci KMS untuk wilayah us-east-1.

```
aws kms --region=us-east-1 create-key --description="kube-aws
assets"
```

Kunci KMS dibuat seperti yang ditunjukkan pada Gambar 2-6. Salin string `KeyMetadata.Arn arn:aws:kms:us-east-1:672593526685:key/b7209ba2-cb87-4ccf-8401-5c6fd4fb9f9b` untuk digunakan nanti guna menginisialisasi kluster CloudFormation.

```
[ec2-user@ip-172-30-1-188 ~]$ aws kms --region=us-east-1 create-key --descriptio
n="kube-aws assets"
{
    "KeyMetadata": {
        "KeyId": "b7209ba2-cb87-4ccf-8401-5c6fd4fb9f9b",
        "Description": "kube-aws assets",
        "Enabled": true,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1467655082.654,
        "Arn": "arn:aws:kms:us-east-1:672593526685:key/b7209ba2-cb87-4ccf-8401-5
c6fd4fb9f9b",
        "AWSAccountId": "672593526685"
    }
}
[ec2-user@ip-172-30-1-188 ~]$
```

Gambar 2.6 Membuat Kunci KMS

Menyiapkan Nama DNS Eksternal

Berikutnya, Anda perlu mendaftarkan nama domain dengan pendaftar domain, karena kami akan menggunakan nama DNS eksternal domain tersebut untuk membuat API klaster dapat diakses. Kami telah menggunakan nama DNS eksternal NOSQLSEARCH.COM. Domain NOSQLSEARCH.COM tidak dapat digunakan untuk semua pengguna, dan pengguna yang berbeda perlu mendaftarkan nama domain yang berbeda dengan pendaftar domain. Atau, gunakan domain yang sudah terdaftar.

Membuat Klaster

Membuat klaster memerlukan prosedur berikut:

1. Buat direktori aset.
2. Inisialisasi tumpukan CloudFormation.
3. Render konten direktori aset.
4. Kustomisasi klaster secara opsional dalam file cluster.yaml.
5. Validasi tumpukan CloudFormation dan file data pengguna cloud-config.
6. Luncurkan tumpukan CloudFormation.

Kami akan membahas masing-masing tahap ini selanjutnya.

Membuat Direktori Aset

Buat direktori pada instans Amazon Linux EC2 untuk aset yang dihasilkan. Lalu cd (ubah direktori) ke direktori aset:

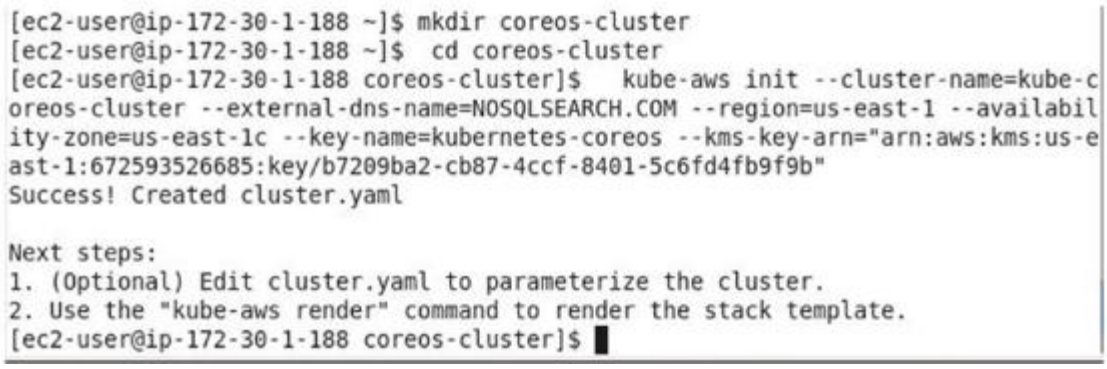
```
mkdir coreos-cluster
cd coreos-cluster
```

Menginisialisasi Cluster CloudFormation

Dengan menggunakan pasangan kunci Amazon EC2, string ARN Kunci KMS, dan nama DNS eksternal, inisialisasi tumpukan CloudFormation:

```
kube-aws init --cluster-name=kube-coreos-cluster
--external-dns-name=NOSQLSEARCH.COM
--region=us-east-1
--availability-zone=us-east-1c
--key-name=kubernetes-coreos
--kms-key-arn="arn:aws:kms:us-east-
1:672593526685:key/b7209ba2-cb87-4ccf-8401-5c6fd4fb9f9b "
```

Aset tumpukan CloudFormation dibuat; berkas konfigurasi utama adalah cluster.yaml, seperti yang ditunjukkan pada Gambar 2.7.



Gambar 2.7 Membuat Aset Tumpukan Cloudformation

2.3 MERENDER KONTEN DIREKTORI ASET

Selanjutnya, render (buat) aset kluster (templat dan kredensial), yang digunakan untuk membuat, memperbarui, dan berinteraksi dengan kluster Kubernetes.

```
kube-aws render
```

Templat CloudFormation stack-template.json dibuat (seperti yang ditunjukkan pada Gambar 2.8); templat ini akan digunakan untuk membuat kluster Kubernetes. Cluster.yaml, file userdata untuk pengontrol Kubernetes dan pekerja, serta stack-template.json dapat disesuaikan secara opsional.

```
[ec2-user@ip-172-30-1-188 coreos-cluster]$ kube-aws render
Success! Stack rendered to stack-template.json.

Next steps:
1. (Optional) Validate your changes to cluster.yaml with "kube-aws validate"
2. (Optional) Further customize the cluster by modifying stack-template.json or
files in ./userdata.
3. Start the cluster with "kube-aws up".
[ec2-user@ip-172-30-1-188 coreos-cluster]$
```

Gambar 2.8 Merender Aset Pengelompokan

Menyesuaikan Kluster

Menyesuaikan kluster bersifat opsional, dan tumpukan CloudFormation dapat diluncurkan dengan pengaturan default-nya. Di antara alasan untuk melakukan penyesuaian adalah untuk menggunakan wilayah penyedia cloud dan nama DNS eksternal yang berbeda dari yang ditentukan saat merender aset kluster dan untuk menggunakan pengaturan nondefault untuk parameter lainnya. Beberapa pengaturan konfigurasi di cluster.yaml dibahas dalam Tabel 2.1.

Tabel 2.1 Pengaturan Konfigurasi Cluster.yaml

Pengaturan konfigurasi	Keterangan	Nilai Default
nama cluster	Nama klaster Kubernetes. Jika lebih dari satu klaster akan di-deploy di akun AWS yang sama, nama klaster Kubernetes harus unik dalam akun AWS tersebut. Untuk klaster contoh, tetapkan ini ke kube-coreos-cluster.	
namaDNSeksternal	Nama DNS dapat dirutekan ke node pengontrol Kubernetes dari node pekerja dan klien eksternal. Konfigurasikan opsi createRecordSet dan hostedZone di bawah ini jika Anda ingin kube-aws membuat kumpulan rekaman Route53/zona yang dihosting untuk Anda. Jika tidak, deployer bertanggung jawab untuk membuat nama ini dapat dirutekan. Untuk contoh kluster, tetapkan ini ke NOSQLSEARCH.COM.	
saluran rilis	Saluran rilis CoreOS yang dapat	alfa

	digunakan. Opsi yang didukung saat ini: [alpha, beta]	
buatRecordSet	Tetapkan ke true jika Anda ingin kube-aws membuat Catatan A Route53 untuk Anda.	Salah
hostedZone	Nama zona yang dihosting untuk menambahkan externalDNSName, seperti “google.com”. Ini harus sudah ada; kube-aws tidak akan membuatnya untuk Anda.	“”
hostedZoneId	ID zona yang dihosting untuk menambahkan externalDNSName. Tentukan hostedZoneId atau hostedZone, tetapi jangan keduanya.	“”
namakunci	Nama pasangan kunci SSH yang sudah dimuat ke akun AWS yang digunakan untuk menyebarkan klaster ini. Untuk klaster contoh, atur ke kubernetes-coreos.	
wilayah	Wilayah untuk penyediaan kluster Kubernetes. Untuk kluster contoh, tetapkan ke us-east-1.	
zona ketersediaan	Zona ketersediaan untuk menyediakan kluster Kubernetes saat menempatkan node dalam satu zona ketersediaan (tidak terlalu tersedia) Beri komentar untuk pengaturan beberapa zona ketersediaan dan gunakan bagian subnet sebagai gantinya. Untuk kluster contoh, atur ke us-east-1c.	
controllerInstanceType	Jenis instans untuk node pengontrol.	m3.sedang
controllerRootVolumeSize	Ukuran disk (GiB) untuk node pengontrol.	30
workerCount	Jumlah node pekerja yang akan dibuat.	1
workerInstanceType	Jenis instans untuk node pekerja.	m3.sedang

Ukuran volume akar pekerja	Ukuran disk (GiB) untuk node pekerja.	30
vpcId	ID VPC yang ada untuk membuat subnet. Kosongkan untuk membuat VPC baru.	
routeTableId	ID tabel rute yang ada di VPC yang ada untuk melampirkan subnet. Kosongkan untuk menggunakan tabel rute utama VPC.	
vpcCIDR	CIDR untuk Kubernetes VPC. Jika vpcId ditentukan, harus sesuai dengan CIDR dari vpc yang ada.	“10.0.0.0/16”
instanceCIDR	CIDR untuk subnet Kubernetes saat menempatkan node dalam satu zona ketersediaan (tidak terlalu tersedia) Biarkan komentar untuk pengaturan beberapa zona ketersediaan dan gunakan bagian subnet sebagai gantinya.	“10.0.0.0/24”
subnet	Subnet Kubernetes dengan CIDR dan zona ketersediaannya. Membedakan zona ketersediaan untuk dua atau lebih subnet menghasilkan ketersediaan tinggi (kegagalan satu zona ketersediaan tidak akan mengakibatkan waktu henti langsung).	
controllerIP	Alamat IP untuk pengontrol di subnet Kubernetes. Bila kita memiliki dua atau lebih subnet, pengontrol ditempatkan di subnet pertama dan controllerIP harus disertakan dalam instanceCIDR dari subnet pertama. Konvensi ini akan berubah setelah CoreOS mendukung pengontrol H/A.	10.0.0.50
serviceCIDR	CIDR untuk semua alamat IP layanan.	“10.3.0.0/24”

podCIDR	CIDR untuk semua alamat IP pod.	“10.2.0.0/16”
Layanan IP dns	Alamat IP layanan dns Kubernetes (harus disertakan oleh serviceCIDR).	10.3.0.10
kubernetesVersi	Versi citra hyperkube yang akan digunakan. Ini adalah tag untuk repositori citra hyperkube.	v1.2.4_coreos.1
hyperkubeImageRepo	Repositori gambar Hyperkube yang akan digunakan.	quay.io/coreos/hyperkube
gunakanCalico	Apakah akan menggunakan Calico untuk kebijakan jaringan. Bila diatur ke "true", kubernetesVersion juga harus diperbarui untuk menyertakan versi yang diberi tag CN,I misalnya v1.2.4_coreos.cni.1.	Salah
stackTags: Nama	Tag AWS untuk sumber daya tumpukan CloudFormation.	“Kubernetes”
stackTags: Lingkungan:	Tag AWS untuk sumber daya tumpukan CloudFormation.	"Produksi"

Secara default, satu pengontrol Kubernetes dan satu pekerja Kubernetes diluncurkan. Sebagai contoh, kita akan mengubah jumlah pekerja Kubernetes menjadi 3. Buka cluster.yaml di editor vi:

```
sudo vi cluster.yaml
```

Tetapkan workerCount ke 3 seperti yang ditunjukkan pada Gambar 2.9 dan simpan file cluster.yaml.

```
# The ID of hosted zone to add the externalDNSName to.
# Either specify hostedZoneId or hostedZone, but not both
#hostedZoneId: ""

# Name of the SSH keypair already loaded into the AWS
# account being used to deploy this cluster.
keyName: kubernetes-coreos

# Region to provision Kubernetes cluster
region: us-east-1

# Availability Zone to provision Kubernetes cluster when placing nodes in a sing
le availability zone (not highly-available) Comment out for multi availability z
one setting and use the below `subnets` section instead.
availabilityZone: us-east-1c

# ARN of the KMS key used to encrypt TLS assets.
kmsKeyArn: "arn:aws:kms:us-east-1:672593526685:key/b7209ba2-cb87-4ccf-8401-5c6fd
4fb9f9b"

# Instance type for controller node
#controllerInstanceType: m3.medium

# Disk size (GiB) for controller node
#controllerRootVolumeSize: 30

# Number of worker nodes to create
workerCount: 3

# Instance type for worker nodes
:wq
```

Gambar 2.9 Memodifikasi cluster.yaml untuk menyetel Worker Nodes ke 3

Menyesuaikan cluster.yaml tidak mengharuskan 31list dirender ulang, tetapi jika file data pengguna atau templat tumpukan dimodifikasi, 31list 31lister harus dirender ulang (kita tidak perlu merender ulang):

```
kube-aws render
```

Memvalidasi Tumpukan CloudFormation

Setelah memodifikasi berkas apa pun (stack-template.json atau berkas data pengguna), tumpukan CloudFormation harus divalidasi:

```
kube-aws validate
```

Seperti yang ditunjukkan oleh keluaran pada Gambar 2.10, data pengguna dan templat tumpukan valid.

```
[ec2-user@ip-172-30-1-188 coreos-cluster]$ kube-aws validate
Validating UserData...
UserData is valid.

Validating stack template...
Validation Report: {
  Capabilities: ["CAPABILITY_IAM"],
  CapabilitiesReason: "The following resource(s) require capabilities: [AWS::IAM
::InstanceProfile, AWS::IAM::Role]",
  Description: "kube-aws Kubernetes cluster kube-coreos-cluster"
}
stack template is valid.

Validation OK!
[ec2-user@ip-172-30-1-188 coreos-cluster]$
```

Gambar 2.10 Memvalidasi Tumpukan Cloudformation

Meluncurkan CloudFormation Kluster

Luncurkan tumpukan CloudFormation dengan perintah berikut:

```
kube-aws up
```

Butuh waktu beberapa menit agar kluster diluncurkan dan agar pengontrol dan pekerja Kubernetes tersedia. Perintah sebelumnya tidak akan selesai hingga kluster diluncurkan. IP pengontrol dicantumkan saat kluster diluncurkan. Status kluster dapat ditemukan dengan perintah berikut:

```
kube-aws status
```

Seperti yang ditunjukkan oleh keluaran dari perintah sebelumnya pada Gambar 2.11, kluster diluncurkan.

```
[ec2-user@ip-172-30-1-188 coreos-cluster]$  kube-aws up
Creating AWS resources. This should take around 5 minutes.
Success! Your AWS resources have been created:
Cluster Name:   kube-coreos-cluster
Controller IP:  23.22.192.55

The containers that power your cluster are now being dowloaded.

You should be able to access the Kubernetes API once the containers finish downl
oading.
[ec2-user@ip-172-30-1-188 coreos-cluster]$ kube-aws status
Cluster Name:   kube-coreos-cluster
Controller IP:  23.22.192.55
```

Gambar 2.11 Meluncurkan Klaster Dan Memvalidasi Status

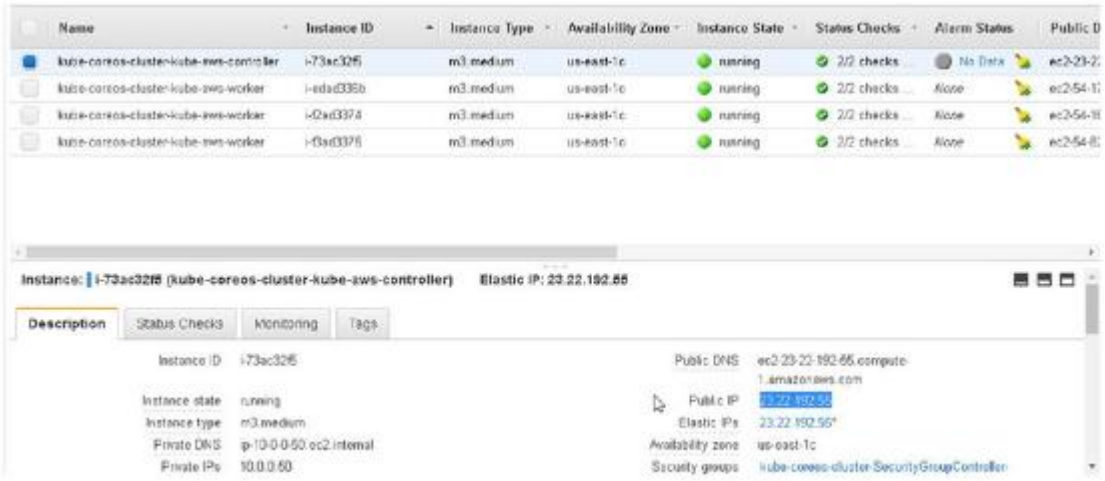
Konsol EC2 harus mencantumkan instans pengontrol dan pekerja sebagai sedang berjalan atau sedang diinisialisasi, seperti yang ditunjukkan pada Gambar 2.12.

Gambar 2.12 Mencantumkan Node Pengontrol Dan Pekerja

Grup keamanan EC2, grup penskalaan, dan konfigurasi peluncuran juga dibuat.

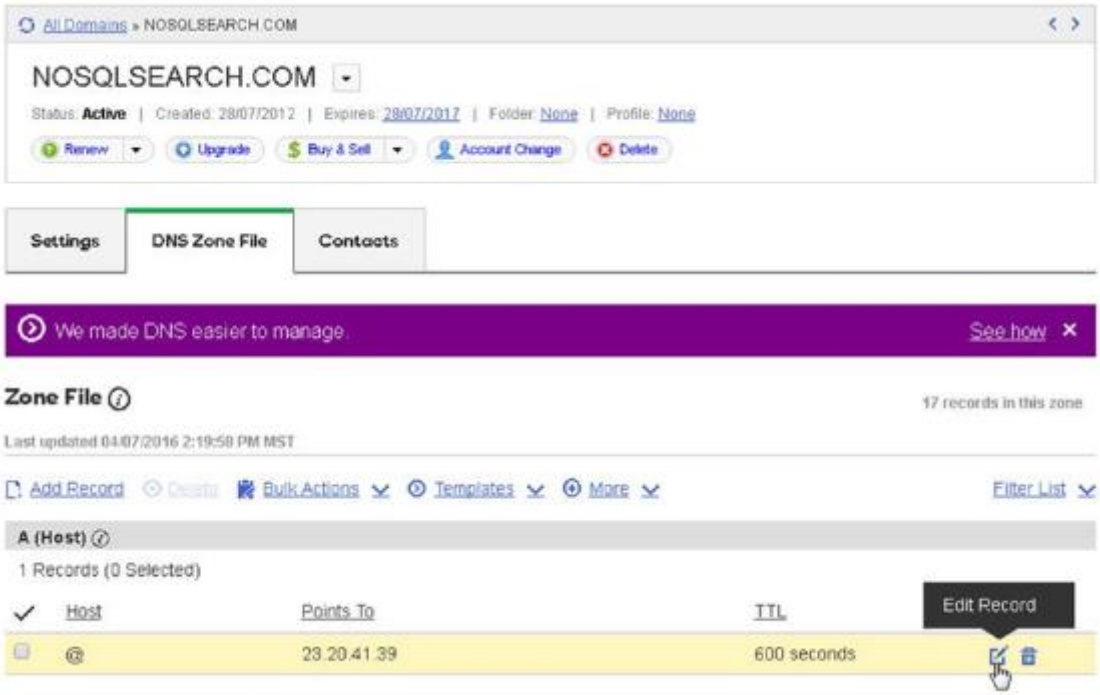
2.4 MENGONFIGURASI DNS

Selanjutnya, kita perlu mengonfigurasi DNS eksternal, NOSQLSEARCH.COM dalam contoh, untuk menambahkan rekaman A untuk alamat IP publik pengontrol. Dapatkan alamat IP publik pengontrol dari konsol EC2 seperti yang ditunjukkan pada Gambar 2.13.



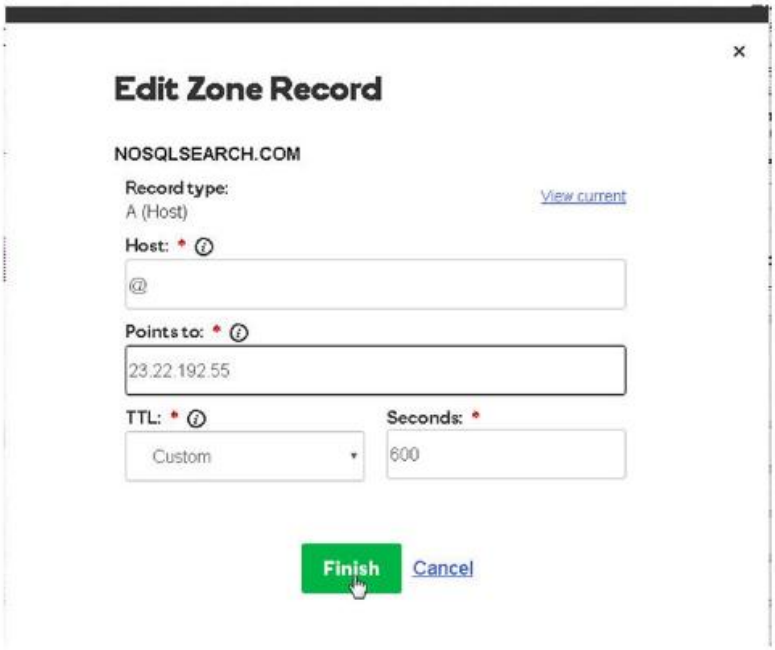
Gambar 2.13 Memperoleh Alamat IP Publik

Prosedur untuk menambahkan catatan A mungkin sedikit berbeda untuk registri domain yang berbeda. Di File Zona DNS untuk DNS eksternal NOSQLSEARCH.COM, pilih Edit Catatan seperti yang ditunjukkan pada Gambar 2.14 untuk mengubah catatan A.



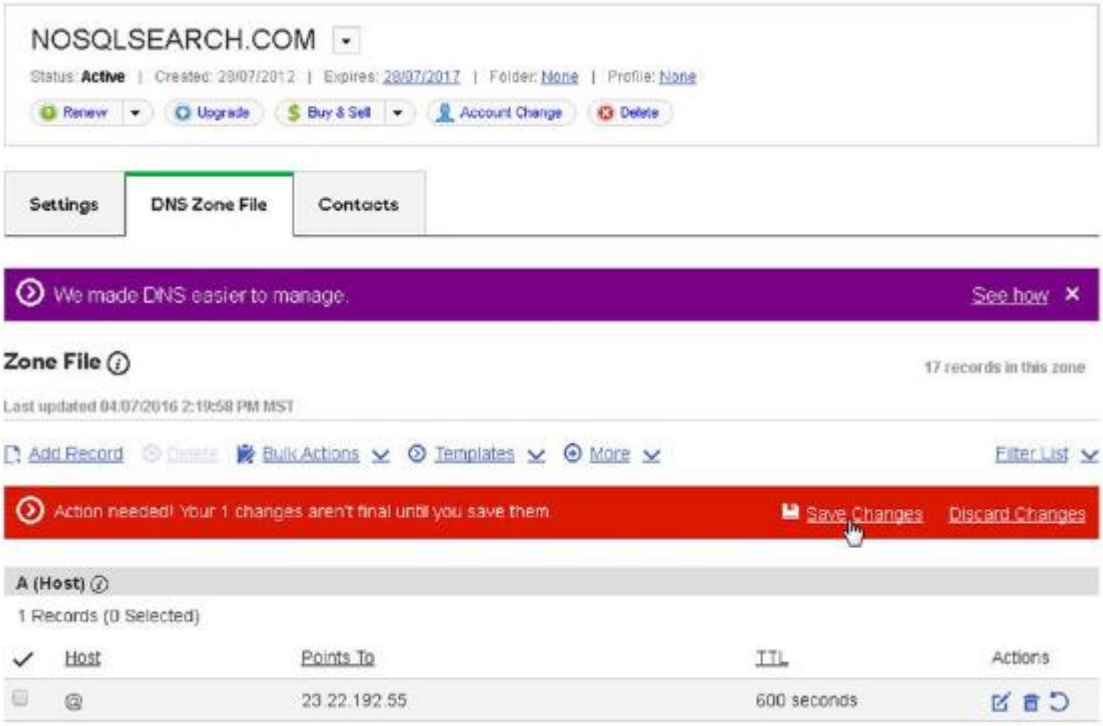
Gambar 2.14 Mengedit catatan A

Pada layar Edit Catatan Zona, tentukan alamat IP publik di kolom Points To dan klik Finish seperti yang ditunjukkan pada Gambar 2.15.



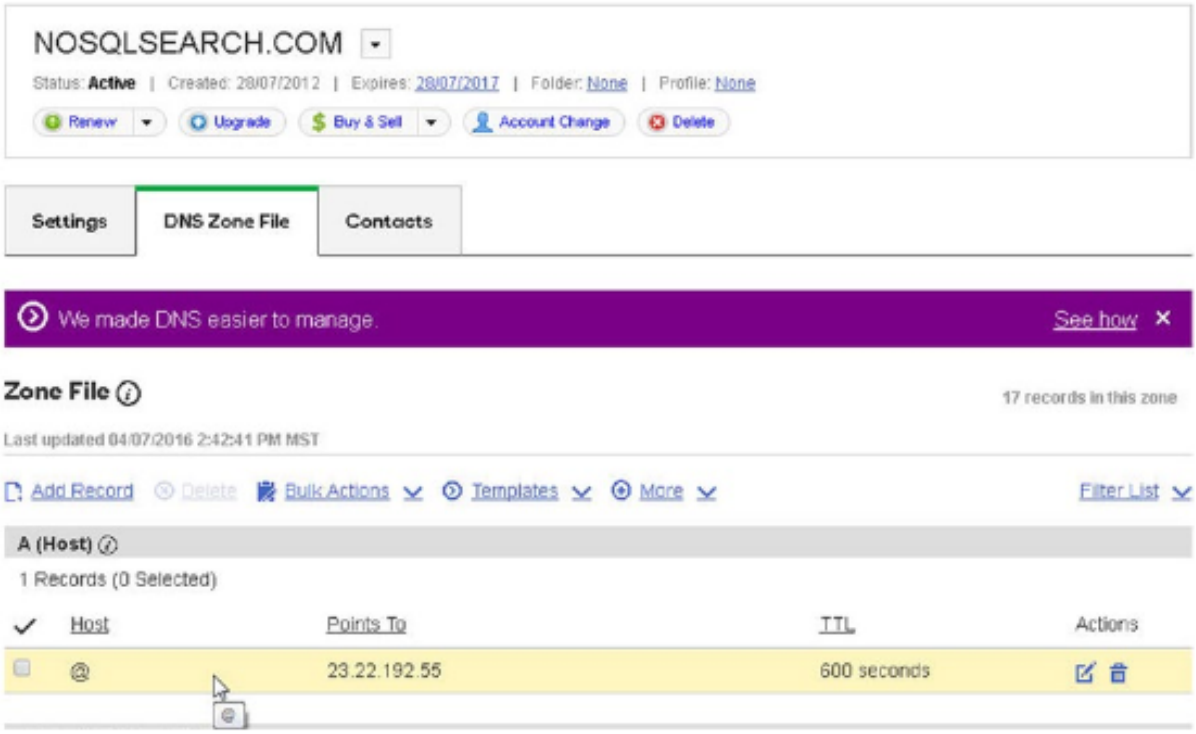
Gambar 2.15 Menambahkan record A

Klik Simpan Perubahan untuk menyimpan modifikasi pada rekaman A seperti yang ditunjukkan pada Gambar 2.16.



Gambar 2.16 Menyimpan Perubahan Pada Rekaman A

Rekaman A harus mencantumkan Points To sebagai alamat IP publik dari instansi pengontrol seperti yang ditunjukkan pada Gambar 2.17.



Gambar 2.17 Rekaman A yang diperbarui

2.5 MENGAKSES KLASTER

Unduh biner kubectl, yang digunakan untuk mengelola klaster Kubernetes. Tetapkan izin akses ke biner kubectl agar dapat dieksekusi, dan pindahkan biner kubectl ke /usr/local/bin, yang ada di jalur:

```
sudo wget https://storage.googleapis.com/kubernetes-release/release/v1.3.0/bin/linux/amd64/ kubectl
sudo chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

Gunakan berkas konfigurasi kubectl untuk mengakses klaster guna mencantumkan node. Satu node pengontrol dan tiga node pekerja harus tercantum seperti yang ditunjukkan pada Gambar 2.18. Node pengontrol tidak dapat dijadwalkan secara default, yang berarti pod tidak dapat dijalankan pada node tersebut.

```
[ec2-user@ip-172-30-1-188 ~]$ kubectl --kubeconfig=kubeconfig get nodes
NAME                          STATUS                     AGE
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   35m
ip-10-0-0-56.ec2.internal     Ready                      35m
ip-10-0-0-57.ec2.internal     Ready                      35m
ip-10-0-0-58.ec2.internal     Ready                      35m
[ec2-user@ip-172-30-1-188 ~]$
```

Gambar 2.18 Mencantumkan node kluster Kubernetes

Menggunakan IP publik dari instans pengontrol, akses instans pengontrol. Nama pengguna harus ditetapkan sebagai “core” karena instans menjalankan CoreOS.

```
ssh -i "kubernetes-coreos.pem" core@23.22.192.55
```

Perintah sebelumnya akan masuk ke instans pengontrol CoreOS seperti yang ditunjukkan pada Gambar 2.19.

```
[ec2-user@ip-172-30-1-188 ~]$ ssh -i "kubernetes-coreos.pem"  core@23.22.192.55
The authenticity of host '23.22.192.55 (23.22.192.55)' can't be established.
ECDSA key fingerprint is 95:f2:5e:04:60:a7:e7:fe:26:7d:c6:76:b2:6c:95:12.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '23.22.192.55' (ECDSA) to the list of known hosts.
CoreOS stable (1010.6.0)
Update Strategy: No Reboots
core@ip-10-0-0-50 ~ $
```

Gambar 2.19 Login SSH ke instance CoreOS

Unduh biner kubectl, atur izin, dan pindahkan biner ke direktori /usr/local/bin untuk memastikannya berada di jalur Anda. Perintah harus dijalankan ulang setelah login ke pengontrol.

```
sudo wget https://storage.googleapis.com/kubernetes-
release/release/v1.3.0/bin/linux/amd64/ kubectl
sudo chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

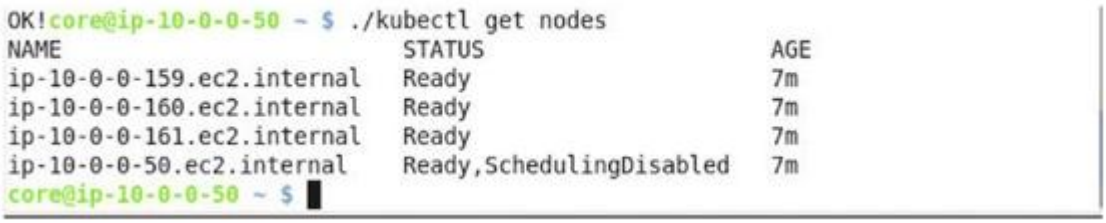
Biner kubectl diinstal seperti yang ditunjukkan pada Gambar 2.20.

Gambar 2.20. Menginstal Biner Kubectl

Daftarkan node-node tersebut:

```
kubectl get nodes
```

Node pengontrol tunggal dan tiga node pekerja dicantumkan seperti yang ditunjukkan pada Gambar 2.21.



Gambar 2.21 Mencantumkan Node Kluster Kubernetes

Menguji Kluster

Untuk menguji kluster, jalankan beberapa contoh aplikasi, seperti server nginx. Jalankan tiga replika pod dari aplikasi nginx:

```
./kubectl -s http://localhost:8080 run nginx --image=nginx -
```

```
replicas=3 --port=80
```

Daftar pengontrol replikasi:

```
./kubectl get rc
```

Daftar layanan:

```
./kubectl get services
```

Daftar deployment:

```
./kubectl get deployments
```

Daftar pod:

```
./kubectl get pods
```

Buat layanan untuk deployment nginx:

```
./kubectl exposed deployment nginx --port=80 --
type=LoadBalancer
```

Daftar layanan lagi, dan layanan nginx akan tercantum:

```
./kubectl get services
```

```
core@ip-10-0-0-50 ~ $ ./kubectl -s http://localhost:8080 run nginx --image=nginx
  --replicas=3 --port=80
deployment "nginx" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc
core@ip-10-0-0-50 ~ $ ./kubectl get services
NAME         CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.3.0.1     <none>        443/TCP   5m
core@ip-10-0-0-50 ~ $ ./kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     3         3         3            3           39s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-198147104-2u7b4    1/1       Running   0          1m
nginx-198147104-c8o3n    1/1       Running   0          1m
nginx-198147104-x0ah0    1/1       Running   0          1m
core@ip-10-0-0-50 ~ $ ./kubectl expose deployment nginx --port=80 --type=LoadBal
ancer
service "nginx" exposed
core@ip-10-0-0-50 ~ $ ./kubectl get services
NAME         CLUSTER-IP   EXTERNAL-IP        PORT(S)   AGE
kubernetes   10.3.0.1     <none>             443/TCP   7m
nginx        10.3.0.127   a1a2ad2f44231...   80/TCP    9s
core@ip-10-0-0-50 ~ $
```

Gambar 2.22 Menunjukkan Output Dari Perintah Sebelumnya.

Berikutnya, jelaskan layanan nginx:

```
./kubectl describe svc nginx
```

Deskripsi layanan mencantumkan titik akhirnya, seperti yang ditunjukkan pada Gambar 2.23.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc nginx
Name:                   nginx
Namespace:              default
Labels:                 run=nginx
Selector:               run=nginx
Type:                   LoadBalancer
IP:                     10.3.0.127
LoadBalancer Ingress:   a1a2ad2f4423111e6848d0a928873259-235290971.us-east-1.elb
.amazonaws.com
Port:                   <unset> 80/TCP
NodePort:               <unset> 31531/TCP
Endpoints:              10.2.29.3:80,10.2.32.2:80,10.2.32.3:80
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubobjectPath  T
ype             Reason                  Message
  ---------     --------        -----   ----                    -------------  -
------- ------                  -------
  38s           38s             1       {service-controller }                  N
ormal           CreatingLoadBalancer    Creating load balancer
  36s           36s             1       {service-controller }                  N
ormal           CreatedLoadBalancer     Created load balancer


core@ip-10-0-0-50 ~ $
```

Gambar 2.23 Deskripsi Layanan Mencantumkan Titik Akhir Layanan

Panggil titik akhir layanan:

```
curl 10.2.29.3
```

Markup HTML untuk aplikasi server nginx ditampilkan seperti yang ditunjukkan pada Gambar 2.24.

```
core@ip-10-0-0-50 ~ $ curl 10.2.29.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
core@ip-10-0-0-50 ~ $
```

Gambar 2.24 Memanggil Titik Akhir Layanan Dengan Curl

Panggil titik akhir lain dengan cara yang sama:

```
curl 10.2.32.2
```

Markup HTML aplikasi nginx tercantum, seperti yang ditunjukkan pada Gambar 2.25.

```
core@ip-10-0-0-50 ~ $ curl 10.2.32.2
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
core@ip-10-0-0-50 ~ $
```

Gambar 2.25 Memanggil Titik Akhir Layanan Lain

Agar dapat memanggil titik akhir layanan nginx di browser, kita perlu mengatur penerusan porta dari mesin lokal. Salin pasangan kunci kubernetes-coreos.pem ke mesin lokal:

```
scp -i docker.pem ec2-user@ec2-54-85-83-181.compute-
1.amazonaws.com:~/kubernetes-coreos.pem
~/kubernetes-coreos.pem
```

Dengan menggunakan pasangan kunci, atur penerusan porta dari mesin lokal ke titik akhir layanan pada instans pengontrol:

```
ssh -i kubernetes-coreos.pem -f -nNT -L 80:10.2.29.3:80
core@ec2-23-22-192-55.compute-1. amazonaws.com
```

Penerusan port dari mesin lokal ke titik akhir layanan diatur seperti yang ditunjukkan pada Gambar 2.26.

```
[root@localhost ~]# scp -i docker.pem ec2-user@ec2-54-85-83-181.compute-1.amazon
aws.com:~/kubernetes-coreos.pem ~/kubernetes-coreos.pem
The authenticity of host 'ec2-54-85-83-181.compute-1.amazonaws.com (54.85.83.181
)' can't be established.
RSA key fingerprint is be:cf:d6:dd:44:d4:39:b0:d9:1d:d0:8e:30:4e:1b:3a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-85-83-181.compute-1.amazonaws.com' (RSA) to t
he list of known hosts.
kubernetes-coreos.pem                         100% 1675     1.6KB/s   00:00
[root@localhost ~]# ssh -i kubernetes-coreos.pem -f -nNT -L 80:10.2.29.3:80 core
@ec2-23-22-192-55.compute-1.amazonaws.com
The authenticity of host 'ec2-23-22-192-55.compute-1.amazonaws.com (23.22.192.55
)' can't be established.
RSA key fingerprint is ad:bd:41:b9:ae:f9:12:47:52:0e:2f:fe:8f:ed:80:8e.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-23-22-192-55.compute-1.amazonaws.com,23.22.192.5
5' (RSA) to the list of known hosts.
[root@localhost ~]#
```

Gambar 2.26 Menetapkan Penerusan Port

Panggil titik akhir layanan pada peramban mesin lokal untuk menampilkan keluaran aplikasi nginx seperti yang ditunjukkan pada Gambar 2.27.

Gambar 2.27 Memanggil Layanan Di Peramban

Keluar dari instans pengontrol seperti yang ditunjukkan pada Gambar 2.28.

```
core@ip-10-0-0-50 ~ $ exit
logout

Connection to 23.22.192.55 closed.
[ec2-user@ip-172-30-1-188 ~]$
```

Gambar 2.28 Keluar Dari Instans Coreos

Ringkasan

Dalam bab ini, kami meluncurkan tumpukan AWS CloudFormation untuk kluster Kubernetes pada instans CoreOS. Prosedur yang kami ikuti adalah sebagai berikut: Pertama, instal kube-aws. Selanjutnya, atur parameter kluster, seperti membuat kunci KMS dan mengatur nama DNS eksternal. Untuk membuat kluster CloudFormation, buat direktori aset, inisialisasi kluster CloudFormation, render konten direktori aset, sesuaikan kluster, validasi kluster, dan luncurkan kluster. Setelah kluster diluncurkan, akses kluster dan buat kluster pod aplikasi nginx. Dalam bab berikutnya, kami akan menginstal Kubernetes di platform Google Cloud.

BAB 3
KUBERNETES DI GOOGLE CLOUD PLATFORM

3.1 MENGGUNAKAN KUBERNETES DI GOOGLE CLOUD PLATFORM

Google Cloud Platform adalah platform komputasi awan publik yang mencakup layanan dan infrastruktur basis data tempat aplikasi dan situs web dapat dihosting di mesin virtual terkelola. PaaS/IaaS terintegrasi ini adalah kumpulan layanan yang dapat dikategorikan menjadi Compute, Storage dan Basis Data, Jaringan, Big Data, dan Machine Learning, untuk menyebutkan beberapa di antaranya.

Masalah

Meskipun Docker sudah terinstal di CoreOS, Kubernetes tidak. Seperti yang dibahas di Bab 2, Kubernetes harus diinstal di CoreOS.

Solusi

Kategori layanan yang paling diminati untuk menggunakan Kubernetes adalah Compute, yang mencakup Compute Engine untuk menjalankan beban kerja skala besar di mesin virtual yang dihosting di infrastruktur Google, App Engine untuk mengembangkan aplikasi web dan seluler yang dapat diskalakan, dan Container Engine untuk menjalankan kontainer Docker di Kubernetes di infrastruktur Google. Google Container Engine adalah pengelola kluster berbasis Kubernetes untuk kontainer Docker dan dengan demikian tidak memerlukan instalasi Kubernetes.

Kita akan menggunakan Google Container Engine, layanan terkelola Google untuk Kubernetes. Google Container Engine telah memasang Docker dan menyediakan dukungan bawaan untuk Google Cloud Platform, yang sebagaimana disebutkan merupakan Infrastruktur sebagai Layanan (IaaS) dan Platform sebagai Layanan (PaaS). Google Cloud Platform merupakan alternatif untuk Amazon Web Services (AWS), yang merupakan penyedia cloud yang kita gunakan di sebagian besar bab lainnya.

Ikhtisar

Pola desain yang dibahas dalam bab-bab berikutnya dapat digunakan pada Google Cloud Platform juga, meskipun konfigurasinya mungkin berbeda. Dalam bab ini kita akan menggunakan Google Compute Engine untuk membuat instans mesin virtual, memasang Kubernetes di dalamnya menggunakan biner, dan selanjutnya membuat contoh aplikasi dan layanan Kubernetes. Kita juga akan membahas penggunaan Google Container Engine, yang merupakan pengelola kluster berbasis Kubernetes untuk kontainer Docker. Langkah-langkah yang akan kita ambil adalah sebagai berikut:

- Menetapkan Lingkungan
- Membuat Proyek di Google Cloud Platform
- Mengaktifkan Izin
- Mengaktifkan API Compute Engine
- Membuat Instans VM
- Menghubungkan ke Instans VM

- Memesan Alamat Statis
- Membuat Kluster Kubernetes
- Membuat Aplikasi dan Layanan Kubernetes
- Menghentikan Kluster
- Menggunakan Kubernetes dengan Google Container Engine

Mengatur Lingkungan

Untuk membuat kluster Kubernetes dan menerapkan aplikasi di dalamnya, kita akan menggunakan prosedur berikut di Google Cloud Platform:

1. Buat proyek.
2. Aktifkan Compute Engine API.
3. Aktifkan izin.
4. Buat dan hubungkan ke instans Mesin Virtual.
5. Pesan alamat statis.
6. Buat kluster Kubernetes.
7. Buat contoh aplikasi dan layanan Kubernetes.

1. Satu-satunya prasyarat adalah menginstal SSH untuk Google Cloud Platform seperti yang ditunjukkan pada Gambar 3.1.

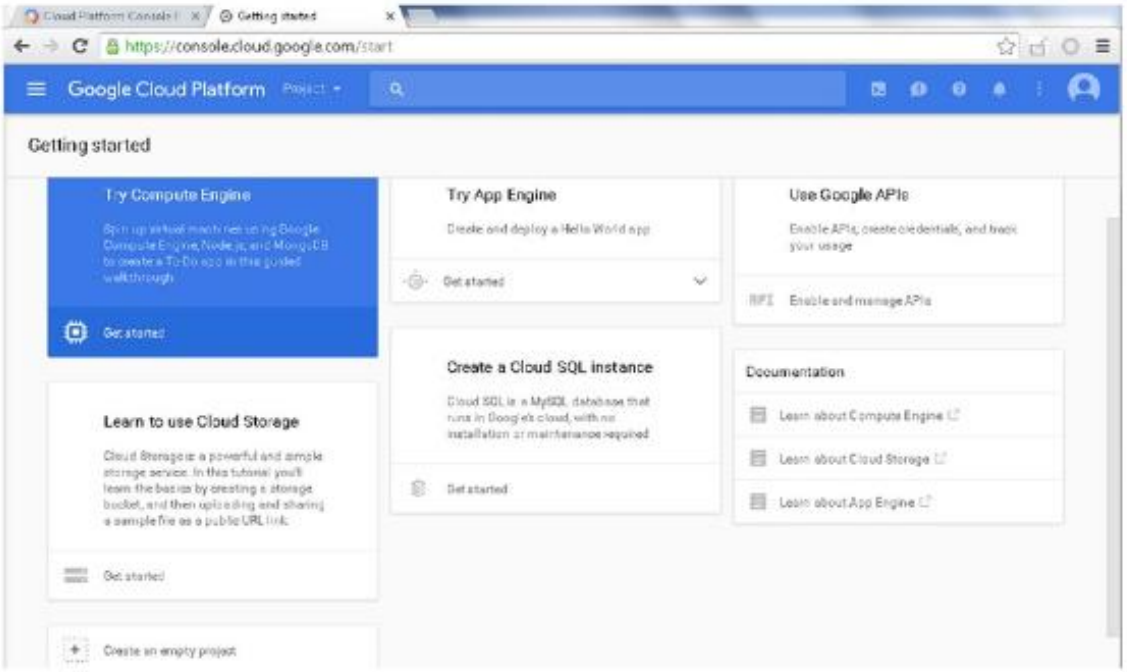
Gambar 3.1 Menginstal SSH untuk Google Cloud Platform

Kita juga perlu membuat Akun Penagihan baru di https://console.cloud.google.com/billing. Sebelum kita dapat menggunakan Compute Engine API, penagihan harus diaktifkan untuk proyek. Sebagian besar artefak Google Cloud Platform juga dapat dibuat dan/atau dikelola dengan alat baris perintah gcloud.

Kita telah menggunakan konsol Google Cloud Platform untuk sebagian besar bab ini kecuali untuk menetapkan beberapa konfigurasi. Ini termasuk mengonfigurasi kubectl untuk menggunakan proyek tertentu, untuk mengirim citra Docker ke Google Container Registry, dan untuk menghapus kluster Google Container Engine.

3.2 MEMBUAT PROYEK DI GOOGLE CLOUD PLATFORM

Untuk membuat proyek, navigasikan ke konsol Google Cloud Platform di https://console.cloud.google.com/start. Konsol Google Cloud Platform ditampilkan seperti yang ditunjukkan pada Gambar 3.2.

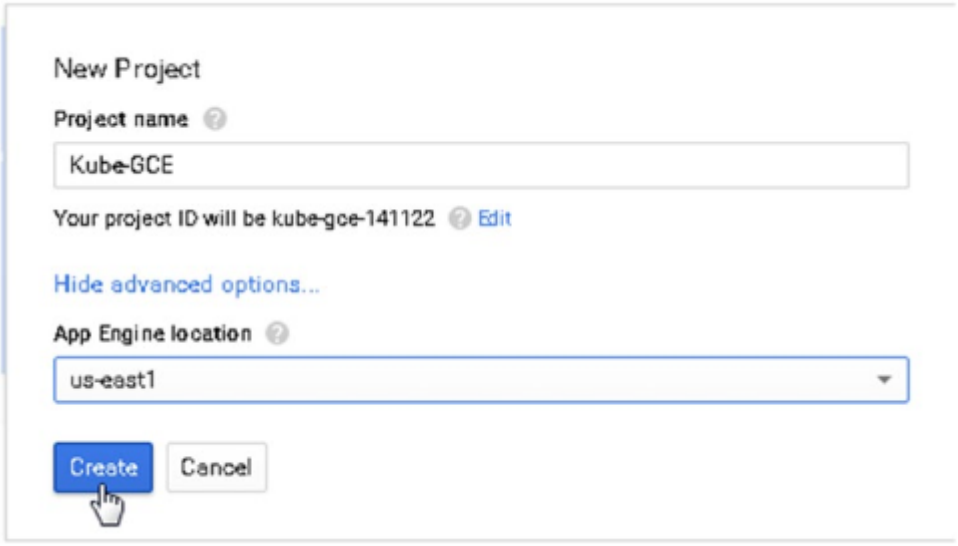


Gambar 3.2 Menampilkan konsol Google Cloud Platform

Pilih menu tarik-turun Proyek dan klik Buat Proyek seperti yang ditunjukkan pada Gambar 3.3.

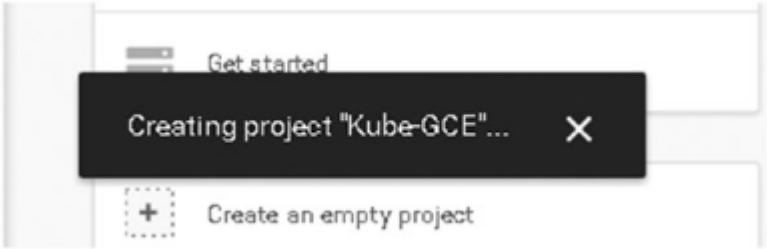
Gambar 3.3 Memilih Buat Proyek Untuk Memulai Pembuatan Proyek

Dalam dialog Proyek Baru, tentukan nama Proyek (misalnya Kube-GCE) dan secara opsional pilih lokasi App Engine di opsi lanjutan. Klik Buat seperti yang ditunjukkan pada Gambar 3.4.



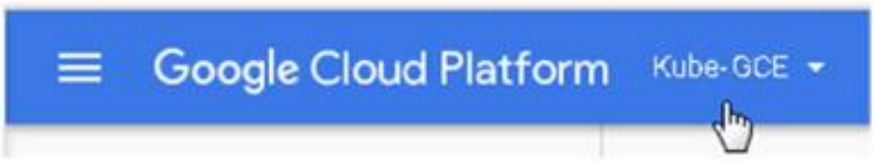
Gambar 3.4 Membuat proyek

Pesan Membuat proyek "Kube-GCE" ditampilkan, seperti yang ditunjukkan pada Gambar 3.5.



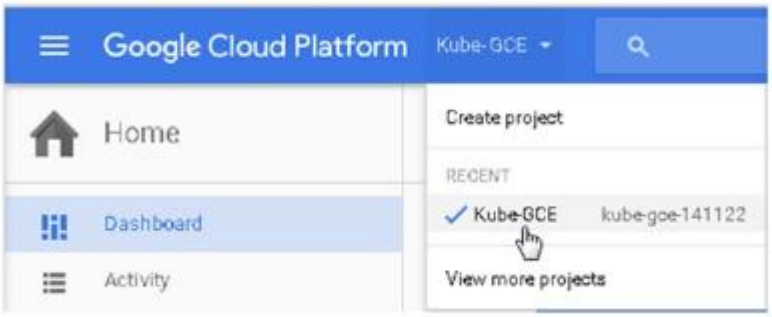
Gambar 3.5 Membuat pesan proyek “Kube-GCE”

Proyek baru ditambahkan di Proyek pada Dasbor seperti yang ditunjukkan pada Gambar 3.6.



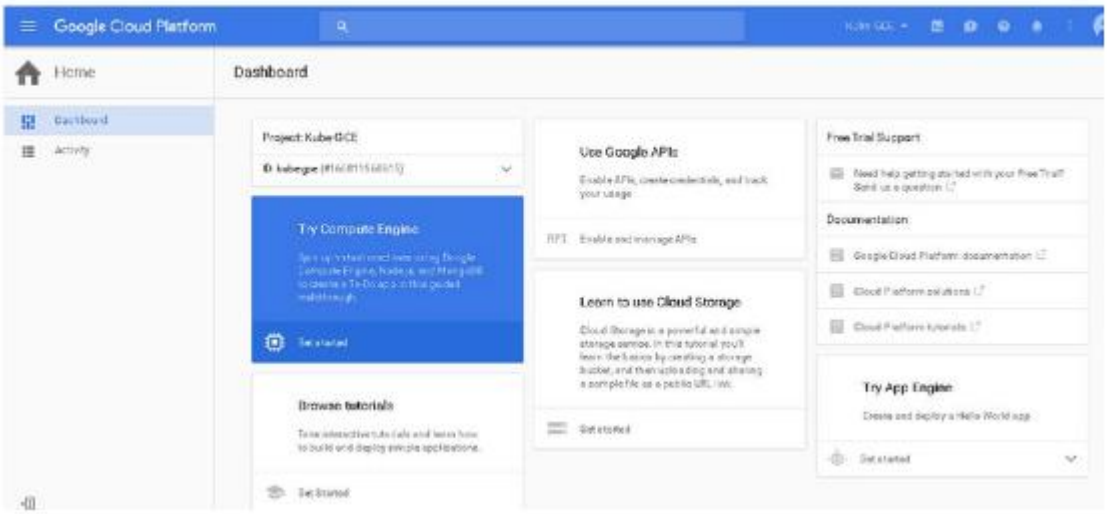
Gambar 3.6 Proyek Baru Ditambahkan

Proyek Kube-GCE dapat dipilih dari daftar proyek untuk menampilkan detailnya seperti yang ditunjukkan pada Gambar 3.7, atau Anda dapat membuat proyek lain dengan Buat Proyek.

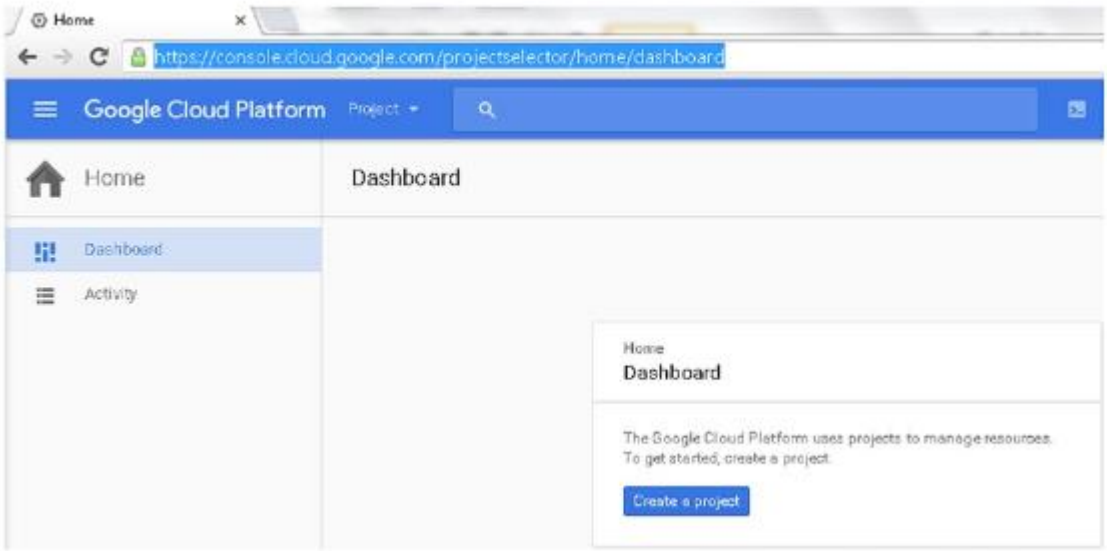


Gambar 3.7 Memilih Proyek

Dasbor dapat diakses di https://console.cloud.google.com/projectselector/home/ dashboard. Proyek Kube-GCE harus tercantum di Dasbor seperti yang ditunjukkan pada Gambar 3.8.

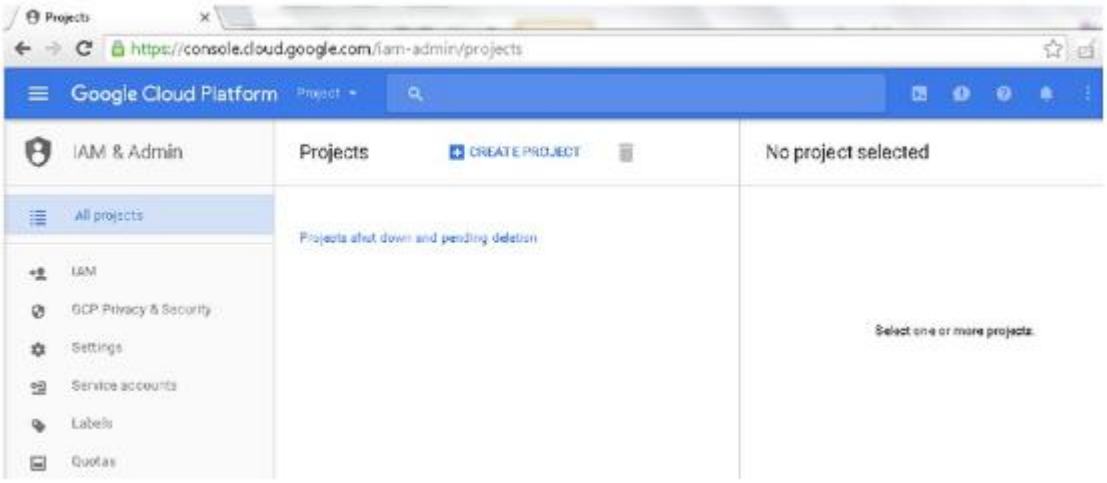


Gambar 3.8 Deskripsi Proyek Di Dasbor



Gambar 3.9 Tautan Buat Proyek Di Dialog Dasbor

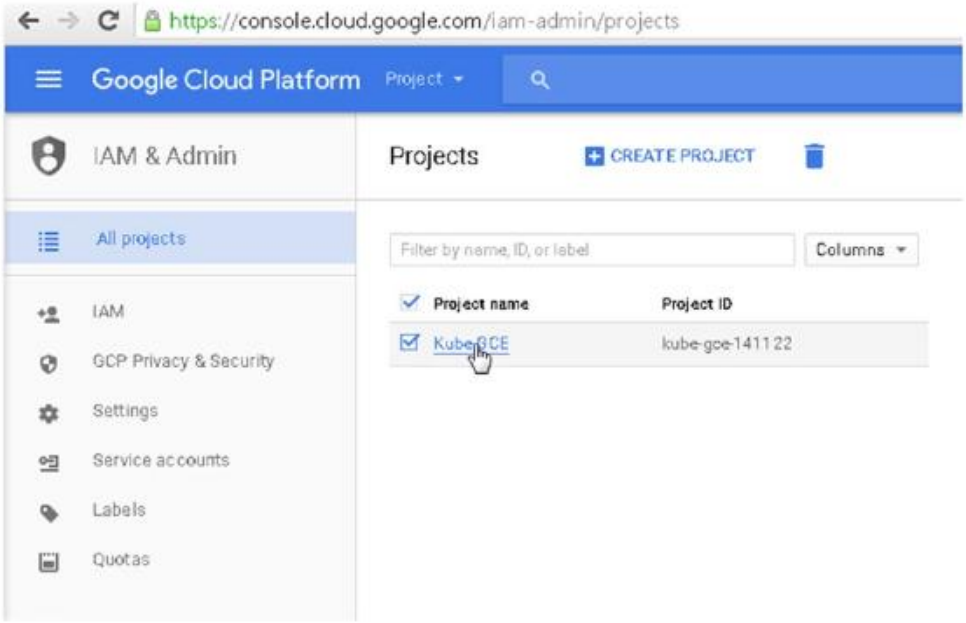
Jika belum ada proyek, URL Dasbor https://console.cloud.google.com/projectselector/home/dashboard menampilkan perintah dialog untuk membuat proyek seperti yang ditunjukkan pada Gambar 3.9. Proyek dapat dikelola di https://console.cloud.google.com/iam-admin/projects, seperti yang ditunjukkan pada Gambar 3.10.



Gambar 3.10 Mengelola Proyek Di Semua Proyek

Mengaktifkan Izin

Untuk mengaktifkan izin untuk suatu proyek, navigasikan ke halaman Proyek di https://console.cloud.google.com/iam-admin/projects. Pilih proyek Kube-GCE seperti yang ditunjukkan pada Gambar 3.11.



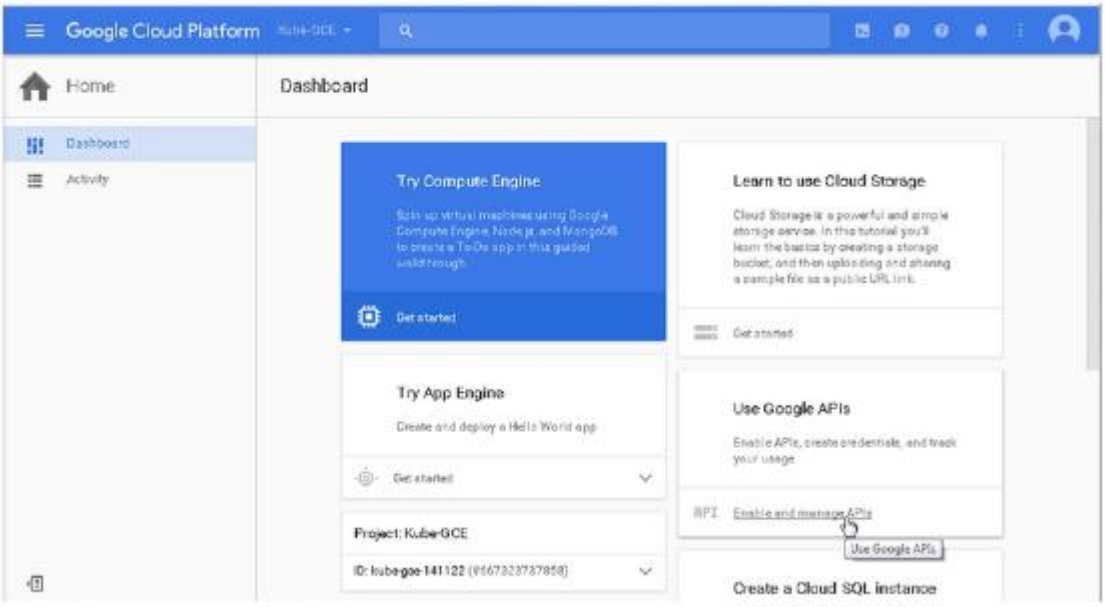
Gambar 3.11 Memilih Proyek

Izin untuk sumber daya proyek ditampilkan. Ubah semua izin menjadi “Pemilik” seperti yang ditunjukkan pada Gambar 3.12.

Gambar 3.12 Menetapkan Izin Untuk Proyek

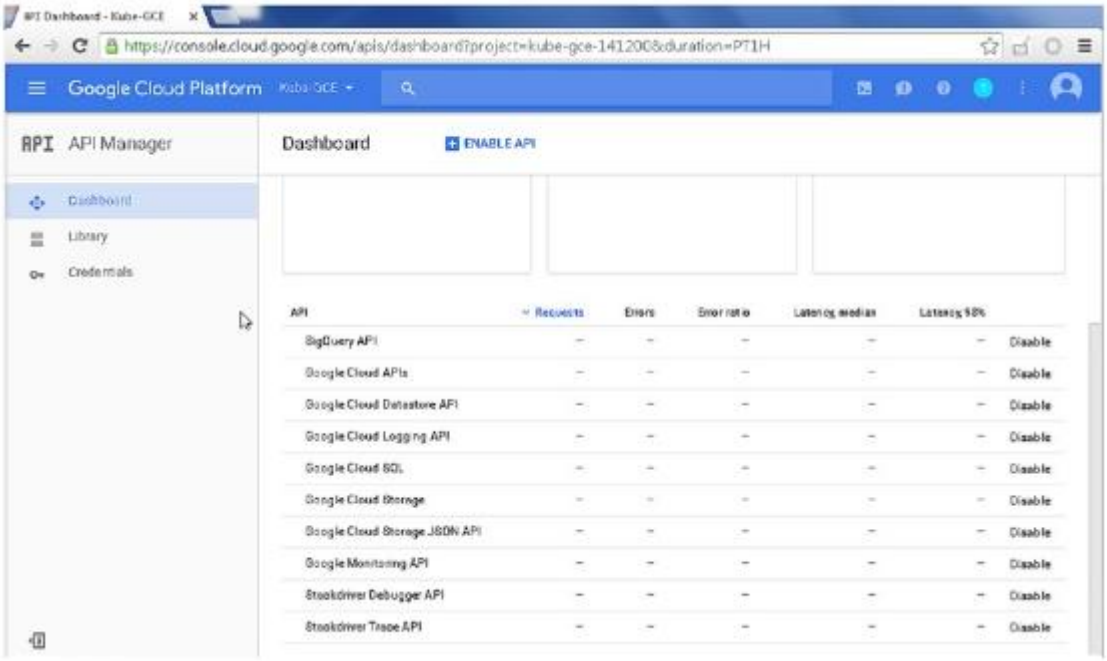
3.3 MENGAKTIFKAN API COMPUTE ENGINE

Agar dapat membuat mesin virtual dan membuat kluster Kubernetes, kita perlu mengaktifkan API Compute Engine. Akses Dasbor di https://console.cloud.google.com/apis/dashboard. Di kolom Use Google APIs, klik tautan Enable and Manage APIs seperti yang ditunjukkan pada Gambar 3.13.



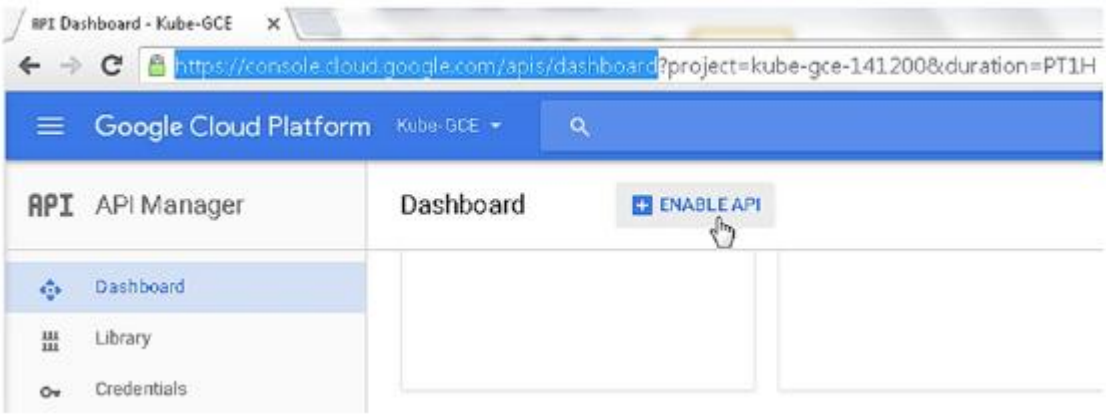
Gambar 3.13 Memilih Tautan Aktifkan Dan Kelola API

API Compute Engine tidak tercantum secara default untuk proyek baru, seperti yang ditunjukkan pada Gambar 3.14.



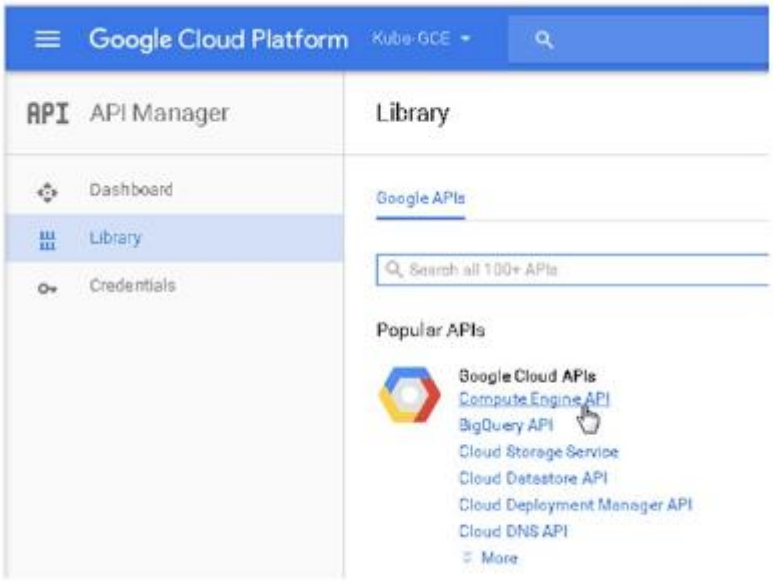
Gambar 3.14 Mencantumkan API Yang Diaktifkan Dan Dinonaktifkan

Untuk memperbaikinya, klik AKTIFKAN API seperti yang ditunjukkan pada Gambar 3.15.



Gambar 3.15 Klik AKTIFKAN API

Kemudian pilih API Compute Engine, seperti yang ditunjukkan pada Gambar 3.16.

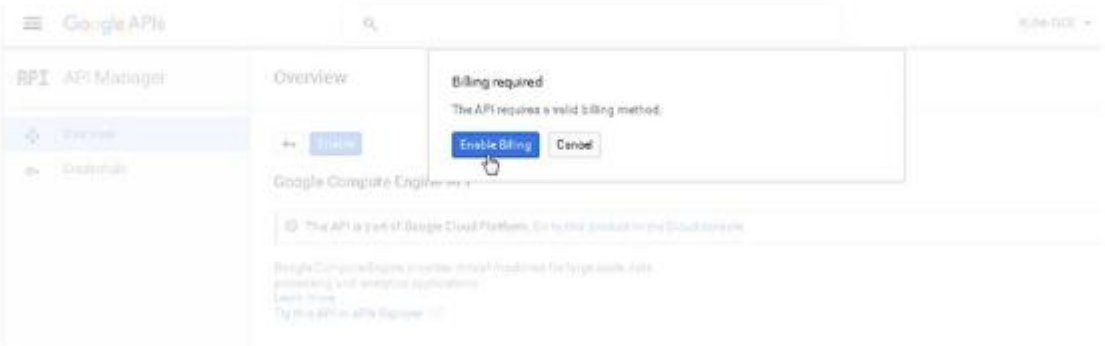


Gambar 3.16 Memilih Google Compute Engine API

Compute Engine API dipilih. Klik AKTIFKAN seperti yang ditunjukkan pada Gambar 3.17.

Gambar 3.17 Mengaktifkan Compute Engine API

Agar dapat mengaktifkan API, Penagihan harus diaktifkan untuk proyek jika belum diaktifkan. Klik Aktifkan Penagihan dalam dialog Penagihan Diperlukan seperti yang ditunjukkan pada Gambar 3.18.



Gambar 3.18 Mengaktifkan penagihan

Pesan ENABLING ditampilkan, seperti yang ditunjukkan pada Gambar 3.19.

Gambar 3.19 Google Compute Engine API sedang diaktifkan

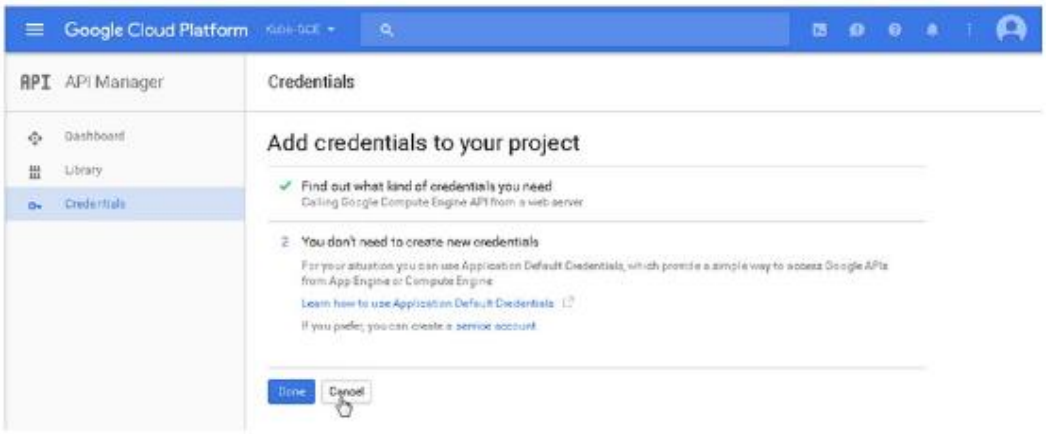
Jika Akun Penagihan tidak ada, klik Buat Akun Penagihan dalam dialog Aktifkan Penagihan untuk Proyek seperti yang ditunjukkan pada Gambar 3.20.

Gambar 3.20 Membuat Akun Penagihan

Setelah akun Penagihan dibuat, API Compute Engine harus diaktifkan. Untuk mengetahui apakah kredensial perlu dibuat untuk suatu proyek, klik Buka Kredensial seperti yang ditunjukkan pada Gambar 3.21.

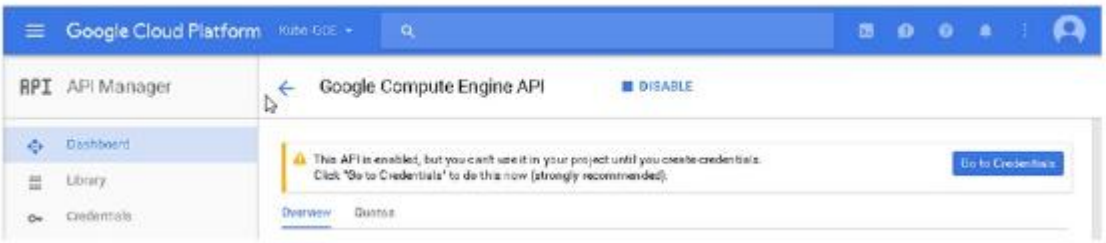
Gambar 3.21 Menavigasi ke halaman Kredensial

Seperti yang ditunjukkan di halaman Kredensial, kredensial baru tidak perlu dibuat dan Kredensial Default Aplikasi dapat digunakan. Klik Batal seperti yang ditunjukkan pada Gambar 3.22.



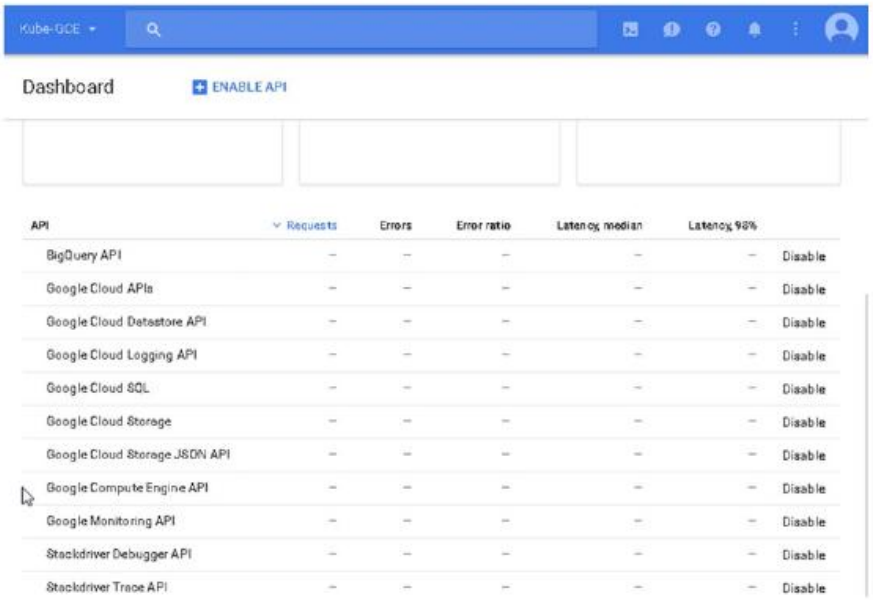
Gambar 3.22 Menentukan Apakah Kredensial Perlu Ditambahkan

API Google Compute Engine diaktifkan untuk proyek Kube-GCE seperti yang ditunjukkan pada Gambar 3.23.



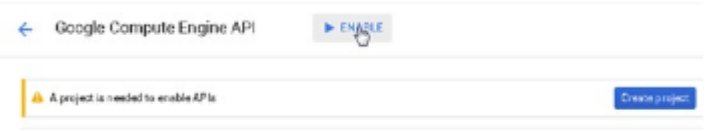
Gambar 3.23 API Google Compute Engine diaktifkan

API Google Compute Engine harus tercantum di Dasbor seperti yang ditunjukkan pada Gambar 3.24.



Gambar 3.24 Google Compute Engine API Tercantum Sebagai Diaktifkan

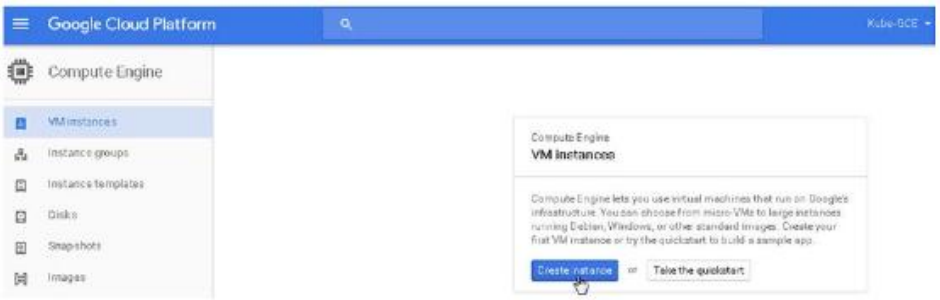
Sebagaimana ditunjukkan oleh pesan Diperlukan proyek untuk mengaktifkan API pada Gambar 3.25, untuk mengaktifkan Google Compute Engine API, diperlukan proyek.



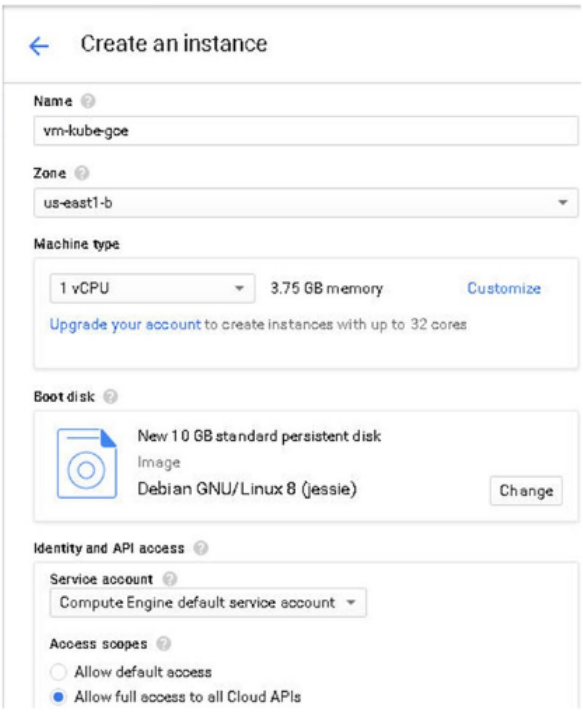
Gambar 3.25 Diperlukan Proyek Untuk Mengaktifkan API

3.4 MEMBUAT INSTANS MESIN VIRTUAL

API Compute Engine menyediakan instans mesin virtual. Untuk membuat instans mesin virtual, navigasikan ke halaman Instans Mesin Virtual di https://console.cloud.google.com/compute/instances. Dalam dialog Compute Engine, klik Buat Instans seperti yang ditunjukkan pada Gambar 3.26.

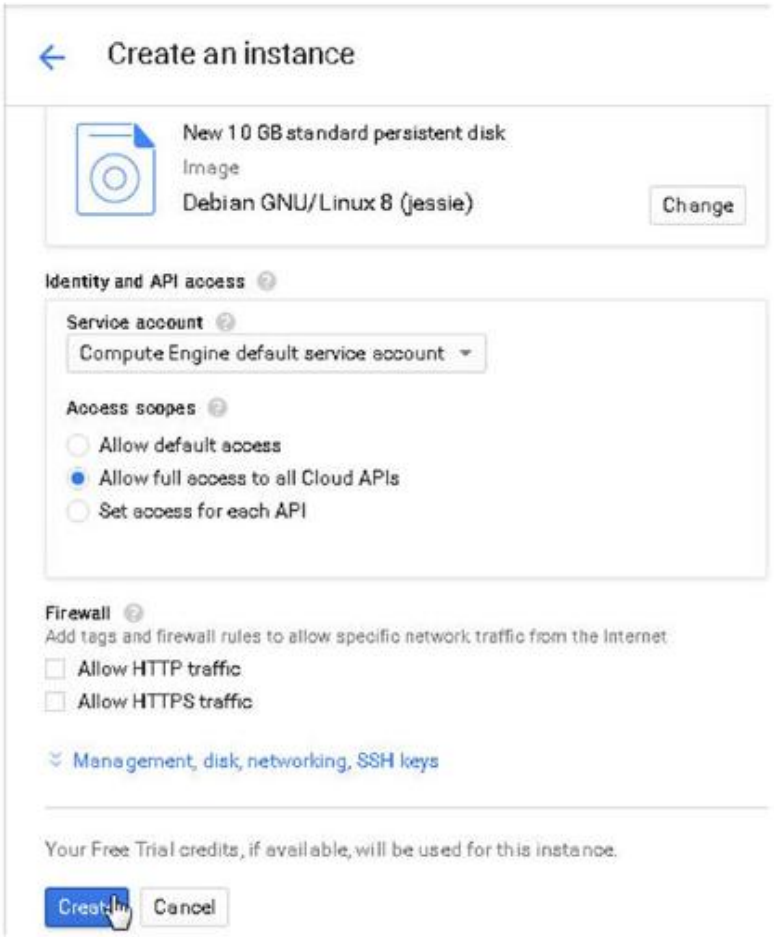


Gambar 3.26 Mengklik Buat Instans

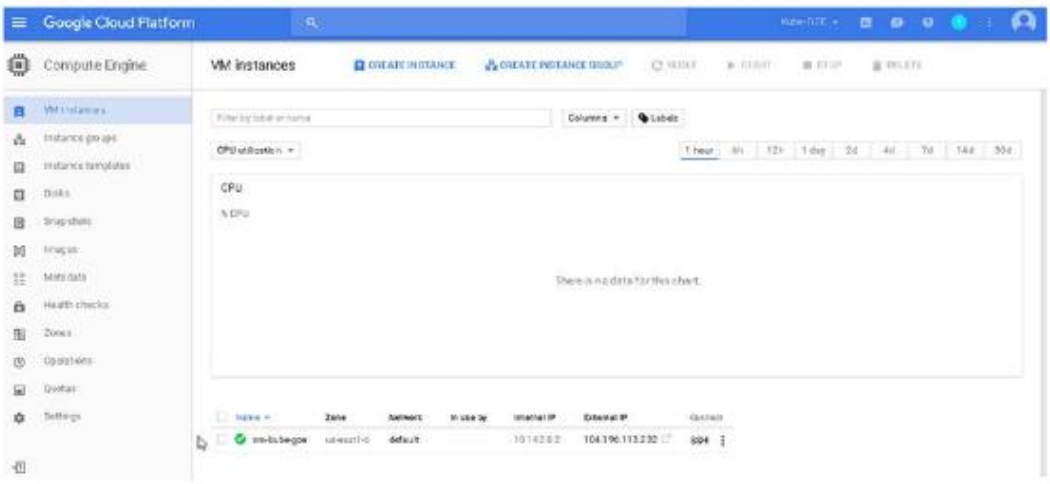


Gambar 3.27 Halaman Buat Instansi

Di halaman Buat instans, tentukan nama instans (misalnya vm-kube-gce). Pilih Zona, Jenis Mesin, Identitas, dan Akses API seperti yang ditunjukkan pada Gambar 3.27. Klik Buat seperti yang ditunjukkan pada Gambar 3.28.

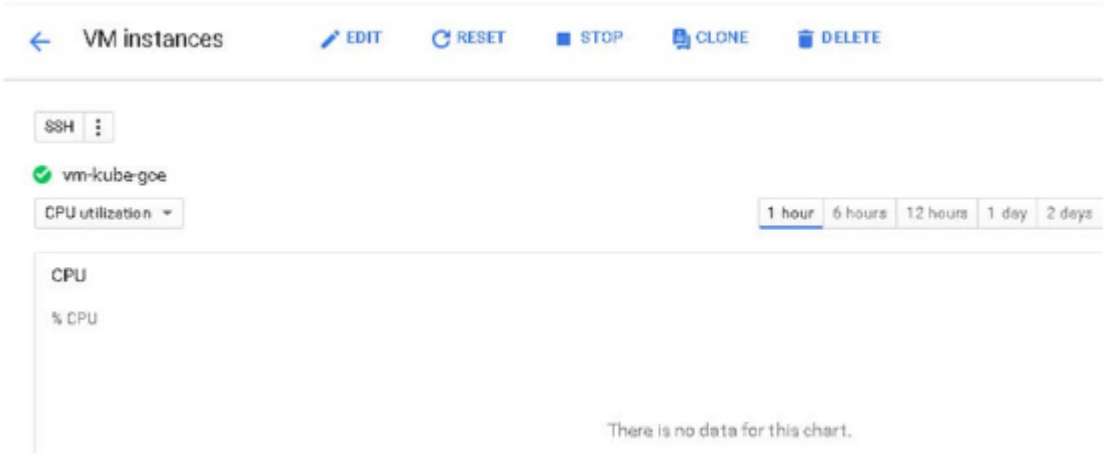


Gambar 3.28 Mengklik Buat



Gambar 3.29 Instans VM baru

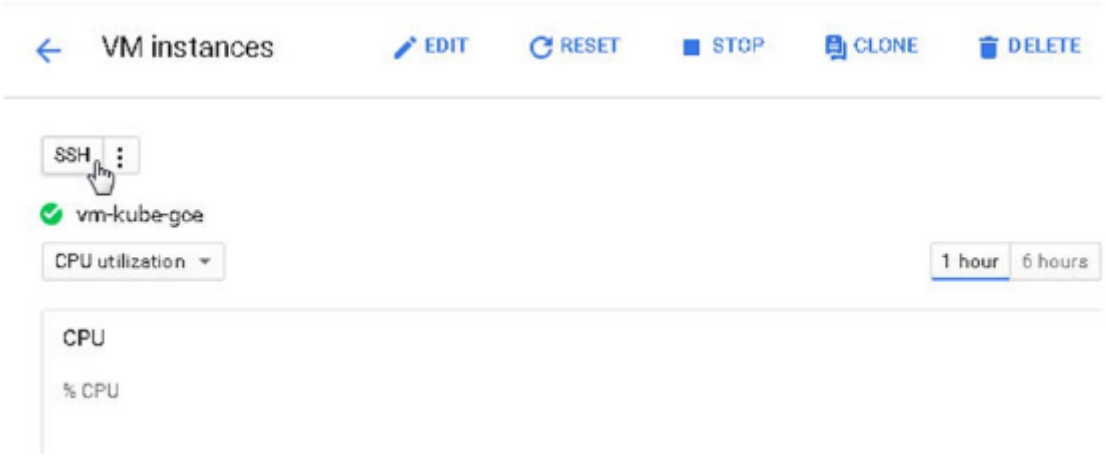
Sebuah instans VM baru dibuat, seperti yang ditunjukkan pada Gambar 3.29. Pilih instans VM untuk mencantumkan statistiknya, seperti Pemanfaatan CPU seperti yang ditunjukkan pada Gambar 3-30. Awalnya, bagan mungkin tidak memiliki data.



Gambar 3.30 Mencantumkan Penggunaan CPU VM

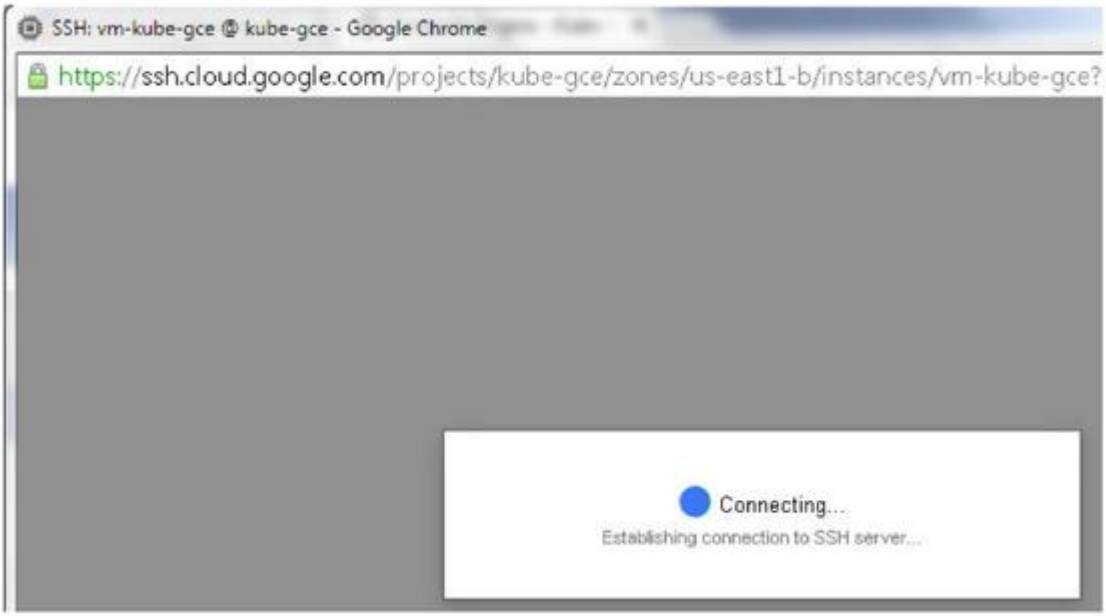
Menghubungkan ke Instans VM

Untuk menghubungkan ke instans VM, klik SSH seperti yang ditunjukkan pada Gambar 3.31.



Gambar 3.31. Mengklik SSH Untuk Mulai Menghubungkan Ke Instans VM

Pesan Connecting akan ditampilkan seperti yang ditunjukkan pada Gambar 3.32.

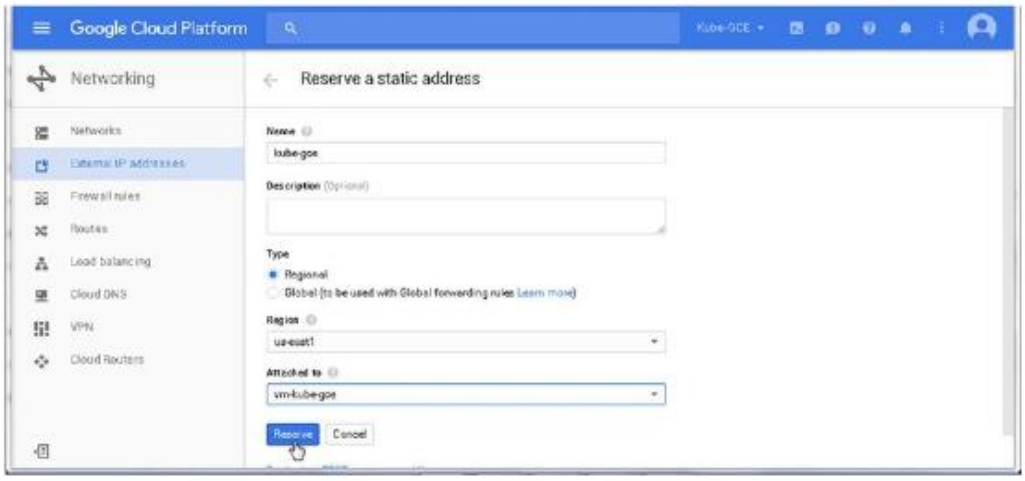


Gambar 3.32 Menghubungkan ke instans VM

Instans VM terhubung dan prompt perintah ditampilkan.

Memesan Alamat IP Eksternal Statis

Setiap instans VM diberi alamat IP internal, yang digunakan untuk berkomunikasi dengan instans VM lain di jaringan yang sama. Agar dapat berkomunikasi di luar jaringan, dengan Internet untuk mengunduh biner Kubernetes misalnya, kita perlu menetapkan alamat IP eksternal statis ke instans VM. Arahkan ke URL https://console.cloud.google.com/networking/addresses untuk membuat alamat IP eksternal statis.



Gambar 3.33 Mengklik Pesan Untuk Memesan Alamat Statis

Di halaman Pesan Alamat Statis, klik Regional dan tentukan Wilayah. Di kolom Attached To, pilih instans VM yang dibuat sebelumnya. Klik Pesan seperti yang ditunjukkan pada Gambar

3.33. Alamat IP eksternal statis dicadangkan untuk instans VM.

Membuat Kluster Kubernetes

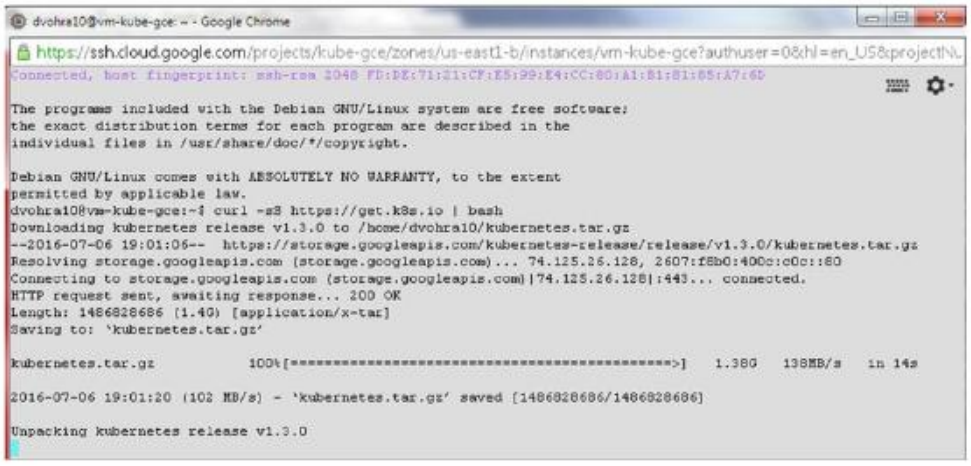
Untuk membuat kluster Kubernetes, jalankan salah satu perintah berikut di shell untuk instans VM.

```
curl -sS https://get.k8s.io | bash
```

atau

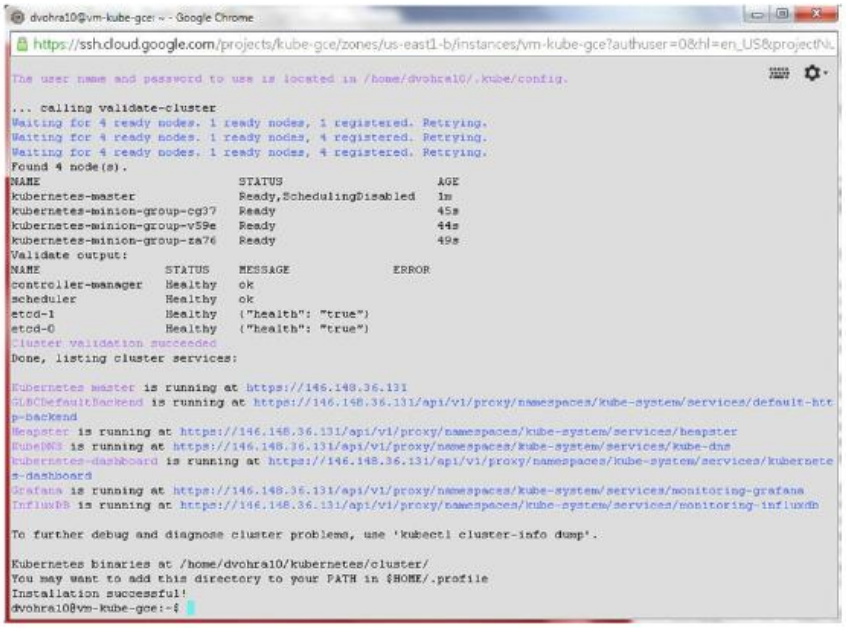
```
wget -q -O - https://get.k8s.io | bash
```

Biner Kubernetes diunduh seperti yang ditunjukkan pada Gambar 3.34.



Gambar 3.34 Mengunduh Biner Kubernetes

Kemudian klaster Kubernetes dimulai seperti yang ditunjukkan pada Gambar 3.35.

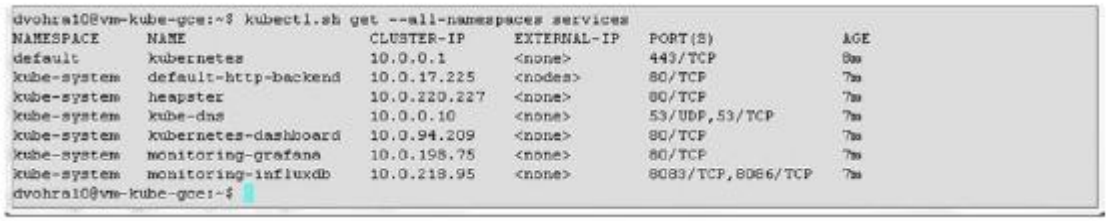


Gambar 3.35 Memulai Klaster Kubernetes Dengan Empat Node

Untuk mencantumkan layanan, jalankan perintah berikut:

```
kubectl.sh get --all-namespaces services
```

Layanan tercantum seperti yang ditunjukkan pada Gambar 3.36.



Gambar 3.36 Mencantumkan Layanan Di Semua Namespace

Untuk mencantumkan semua pod, jalankan perintah berikut:

```
kubectl.sh get --all-namespaces pods
```

Semua pod di semua namespace tercantum, seperti yang ditunjukkan pada Gambar 3.37.

```
dvohra10@vm-kube-gce:~$ kubectl get --all-namespaces pods
-bash: kubectl: command not found
dvohra10@vm-kube-gce:~$ kubectl.sh get --all-namespaces pods
NAMESPACE     NAME                                                READY     STATUS    RESTARTS   AGE
kube-system   etcd-server-events-kubernetes-master                1/1       Running   0          9m
kube-system   etcd-server-kubernetes-master                       1/1       Running   0          9m
kube-system   fluentd-cloud-logging-kubernetes-master             1/1       Running   0          8m
kube-system   fluentd-cloud-logging-kubernetes-minion-group-cg37   1/1       Running   0          8m
kube-system   fluentd-cloud-logging-kubernetes-minion-group-v59e   1/1       Running   0          8m
kube-system   fluentd-cloud-logging-kubernetes-minion-group-za76   1/1       Running   0          8m
kube-system   heapster-v1.1.0-527143062-t57xt                      4/4       Running   0          7m
kube-system   kube-addon-manager-kubernetes-master                1/1       Running   0          9m
kube-system   kube-apiserver-kubernetes-master                    1/1       Running   1          9m
kube-system   kube-controller-manager-kubernetes-master           1/1       Running   0          8m
kube-system   kube-dns-v17-yz76j                                  3/3       Running   0          8m
kube-system   kube-proxy-kubernetes-minion-group-cg37             1/1       Running   0          8m
kube-system   kube-proxy-kubernetes-minion-group-v59e             1/1       Running   0          8m
kube-system   kube-proxy-kubernetes-minion-group-za76             1/1       Running   0          8m
kube-system   kube-scheduler-kubernetes-master                    1/1       Running   0          8m
kube-system   kubernetes-dashboard-v1.1.0-4u21q                   1/1       Running   0          8m
kube-system   l7-default-backend-v1.0-1norr                       1/1       Running   0          8m
kube-system   l7-lb-controller-v0.7.0-kubernetes-master           1/1       Running   0          9m
kube-system   monitoring-influxdb-grafana-v3-spg18                2/2       Running   0          8m
kube-system   node-problem-detector-v0.1-01zvh                    1/1       Running   0          8m
kube-system   node-problem-detector-v0.1-5fw5i                    1/1       Running   0          8m
kube-system   node-problem-detector-v0.1-8sl7g                    1/1       Running   0          8m
kube-system   node-problem-detector-v0.1-jn5r2                    1/1       Running   0          8m
dvohra10@vm-kube-gce:~$
```

Gambar 3.37 Mencantumkan Semua Pod

Untuk mencantumkan semua node, jalankan perintah berikut:

```
kubectl.sh get nodes
```

Satu node kontroler dan tiga node minion tercantum, seperti yang ditunjukkan pada Gambar 3.38.

```
dvohra10@vm-kube-gce:~$ kubectl.sh get nodes
NAME                       STATUS                     AGE
kubernetes-master          Ready,SchedulingDisabled   10m
kubernetes-minion-group-cg37   Ready                  9m
kubernetes-minion-group-v59e   Ready                  9m
kubernetes-minion-group-za76   Ready                  9m
dvohra10@vm-kube-gce:~$
```

Gambar 3.38 Mencantumkan Node Kluster Kubernetes

Cantumkan semua namespace dengan perintah berikut:

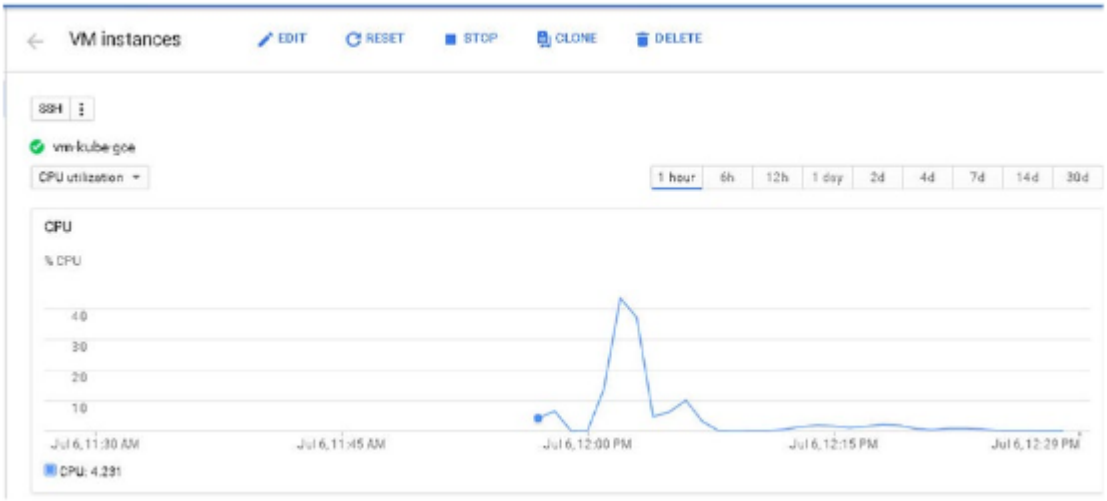
```
kubectl.sh get namespaces
```

Dua namespace default dan kube-system tercantum, seperti yang ditunjukkan pada Gambar 3.39.

```
dvohra10@vm-kube-gce:~$ kubectl.sh get namespaces
NAME          STATUS    AGE
default       Active    15m
kube-system   Active    15m
dvohra10@vm-kube-gce:~$
```

Gambar 3.39 Mencantumkan Namespace

Pemanfaatan CPU dari instans VM dapat ditampilkan di konsol seperti yang ditunjukkan pada Gambar 3.40.

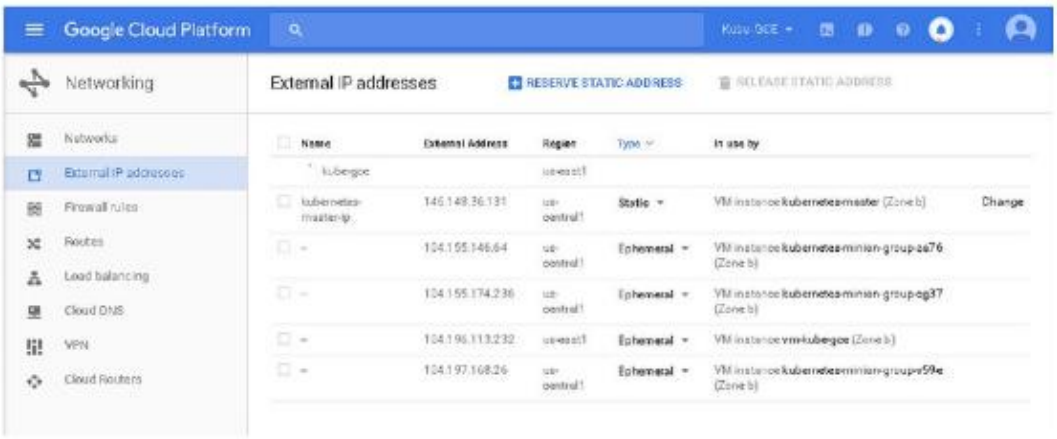


Gambar 3.40 Menampilkan Grafik Untuk Pemanfaatan CPU

VM Instances juga mencantumkan instance kontroler dan minion yang dimulai untuk kluster Kubernetes, seperti yang ditunjukkan pada Gambar 3.41.

Gambar 3.41 Mencantumkan Instance Kontroler Dan Minion Kubernetes

Pilih tab External IP Addresses untuk mencantumkan semua alamat IP eksternal, termasuk instance kontroler dan minion, seperti yang ditunjukkan pada Gambar 3.42.



Gambar 3.42 Mencantumkan Alamat IP Eksternal

3.5 MEMBUAT APLIKASI DAN LAYANAN KUBERNETES

Di bagian ini, kita akan membuat contoh aplikasi Kubernetes menggunakan image Docker nginx. Perintah berikut membuat deployment untuk image Docker nginx.

```
kubectl.sh --namespace=default run nginx --image=nginx –
replicas=3 –port=80
```

Deployment "nginx" dibuat seperti yang ditunjukkan pada Gambar 3.43.

```
dvohra10@vm-kube-gce:~$ kubectl.sh --namespace=default  run nginx --image=nginx  --replicas=3 --port=80
deployment "nginx" created
dvohra10@vm-kube-gce:~$
```

Gambar 3.43 Membuat Deployment Nginx

Daftar pod, termasuk node tempat pod berjalan:

```
kubectl.sh get pods -o wide
```

Tiga replika pod termasuk node dicantumkan seperti yang ditunjukkan pada Gambar 3.44.

```
dvohra10@vm-kube-gce:~$ kubectl.sh --namespace=default  run nginx --image=nginx  --replicas=3 --port=80
deployment "nginx" created
dvohra10@vm-kube-gce:~$ kubectl.sh get pods -o wide
NAME                     READY     STATUS    RESTARTS   AGE       IP           NODE
nginx-2032906785-a9b1g   1/1       Running   0          27s       10.244.1.5   kubernetes-minion-group-za76
nginx-2032906785-lyig2   1/1       Running   0          27s       10.244.3.4   kubernetes-minion-group-v59e
nginx-2032906785-qmc10   1/1       Running   0          27s       10.244.2.5   kubernetes-minion-group-cg37
dvohra10@vm-kube-gce:~$
```

Gambar 3.44 Mencantumkan Replika Node

Daftar deployment:

```
kubectl.sh get deployments
```

Buat layanan untuk tipe LoadBalancer:

```
kubectl.sh exposed deployment nginx --port=80 --
type=LoadBalancer
```

Daftar layanan:

```
kubectl.sh get services
```

Output dari perintah sebelumnya ditunjukkan pada Gambar 3.45.

```
dvohra10@vm-kube-gce:~$ kubectl.sh get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     3         3         3            3           59s
dvohra10@vm-kube-gce:~$ kubectl.sh expose deployment nginx --port=80 --type=LoadBalancer
service "nginx" exposed
dvohra10@vm-kube-gce:~$ kubectl.sh get services
NAME         CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   10.0.0.1        <none>        443/TCP   31m
nginx        10.0.213.175    <pending>     80/TCP    10s
dvohra10@vm-kube-gce:~$
```

Gambar 3.45 Mencantumkan Deployment Dan Layanan

Jelaskan layanan nginx:

```
kubectl.sh describe svc nginx
```

Deskripsi layanan, termasuk titik akhir layanan dan pesan kesalahan apa pun, dicantumkan seperti yang ditunjukkan pada Gambar 3.46.

```
dvohra10@vm-kube-gce:~$ kubectl.sh describe svc nginx
Name:                   nginx
Namespace:              default
Labels:                 run=nginx
Selector:               run=nginx
Type:                   LoadBalancer
IP:                     10.0.213.175
Port:                   <unset> 80/TCP
NodePort:               <unset> 31882/TCP
Endpoints:              10.244.1.5:80,10.244.2.5:80,10.244.3.4:80
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubobjectPath   Type            Reason          M
essage
  ---------     --------        -----   ----                    -------------   --------        ------          -
------
  1m            36s             5       {service-controller }                   Normal          CreatingLoadBala
ncer            Creating load balancer
  1m            36s             5       {service-controller }                   Warning         CreatingLoadBala
ncerFailed      Error creating load balancer (will retry): Failed to create load balancer for service default/ng
inx: failed to ensure static IP : error creating gce static IP address: googleapi: Error 403: Quota 'STATIC_ADDR
ESSES' exceeded. Limit: 1.0, quotaExceeded


dvohra10@vm-kube-gce:~$
```

Gambar 3.46 Mencantumkan Deskripsi Layanan

Selanjutnya, kita akan memanggil titik akhir layanan. Salin titik akhir layanan seperti yang

ditunjukkan pada Gambar 3.47.

```
dvohra10@vm-kube-gce:~$ kubectl.sh describe svc nginx
Name:                   nginx
Namespace:              default
Labels:                 run=nginx
Selector:               run=nginx
Type:                   LoadBalancer
IP:                     10.0.213.175
Port:                   <unset> 80/TCP
NodePort:               <unset> 31882/TCP
Endpoints:              10.244.1.5:80,10.244.2.5:80,10.244.3.4:80
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubobjectPath   Type            Reason          M
essage
```

Gambar 3.47 Memperoleh Titik Akhir Layanan

Panggil titik akhir layanan dengan curl:

```
curl 10.244.1.5
```

Markup HTML untuk layanan tersebut dicantumkan seperti yang ditunjukkan pada Gambar 3.48.

```
dvohra10@vm-kube-gce:~$ curl 10.244.1.5
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
dvohra10@vm-kube-gce:~$
```

Gambar 3.48 Memanggil Titik Akhir Layanan

Demikian pula, panggil titik akhir layanan lain:

```
curl 10.244.2.5
```

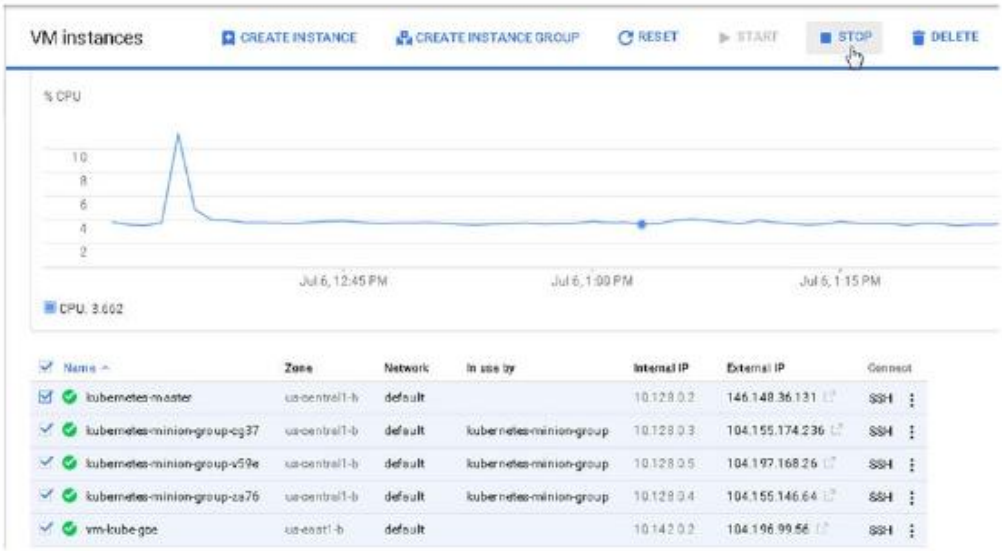
Titik akhir layanan kedua juga dipanggil, seperti yang ditunjukkan pada Gambar 3.49.

```
dvohra10@vm-kube-gce:~$ curl 10.244.2.5
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
dvohra10@vm-kube-gce:~$
```

Gambar 3.49 Memanggil Titik Akhir Layanan Lain

Menghentikan Kluster

Untuk menghentikan instans VM, pilih instans di konsol dan klik Hentikan seperti yang ditunjukkan pada Gambar 3.50.



Gambar 3.50 Memilih Semua Node Dan Mengeklik Stop

Pada dialog verifikasi, pilih Stop seperti yang ditunjukkan pada Gambar 3.51.

Gambar 3.51 Menghentikan Instans VM

Instans VM dihentikan seperti yang ditunjukkan pada Gambar 3.52.

Gambar 3.52 Instans VM Yang Dihentikan

3.6 MENGGUNAKAN KUBERNETES DENGAN GOOGLE CONTAINER ENGINE

Google Container Engine adalah layanan yang dikelola Google untuk kluster Kubernetes yang menjalankan kontainer Docker. Google Container Engine adalah komponen dari Google Cloud Platform. Layanan ini sepenuhnya mengelola dan mengatur kluster, termasuk menjadwalkan kontainer dan menjalankannya berdasarkan persyaratan CPU dan memori yang ditentukan.

Google Container Engine memberikan fleksibilitas penggunaan cloud privat, publik, atau hibrida, dan menyediakan penskalaan otomatis kluster berdasarkan pemanfaatan sumber daya. Layanan Google seperti Google Cloud Logging, Google Cloud VPN, Google Container Registry, dan akun Google serta izin peran terintegrasi dengan Google Container Engine. Untuk menjalankan aplikasi Kubernetes di Google Container Engine, prosedur berikut digunakan.

1. Buat Akun Penagihan jika belum ada.
2. Buat Proyek di Google Cloud Platform.
3. Aktifkan Izin untuk proyek.
4. Aktifkan Penagihan untuk proyek.
5. Aktifkan Google Compute Engine dan Google Container Engine API.
6. Buat Google Container Cluster.
7. Hubungkan ke Google Cloud Shell.

8. Konfigurasikan kubectl untuk cluster container.
9. Uji cluster Kubernetes.

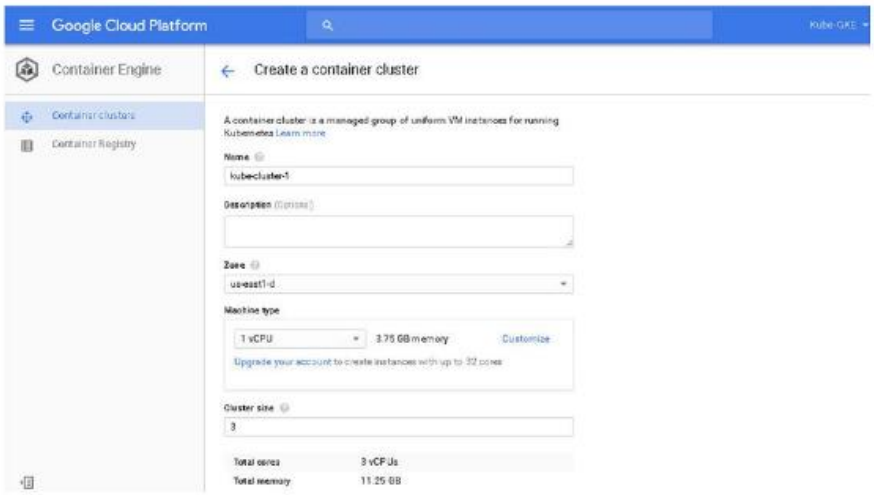
Kami telah membahas langkah 1 hingga 5 sebelumnya dalam bab ini, kecuali bahwa Google Container Engine API juga perlu diaktifkan. Di bagian ini, kami akan membahas langkah 6 dan seterusnya. Kami telah menggunakan proyek yang disebut Kube-GKE.

Membuat Google Container Cluster

Pilih proyek tempat Google Container Cluster akan dibuat di Google Container Engine di URL https://console.cloud.google.com/kubernetes. URL yang mirip dengan https://console.cloud.
google.com/kubernetes/list?project=kubernete-gke dipanggil. Di Container Clusters, klik Create a Container Cluster seperti yang ditunjukkan pada Gambar 3.53.

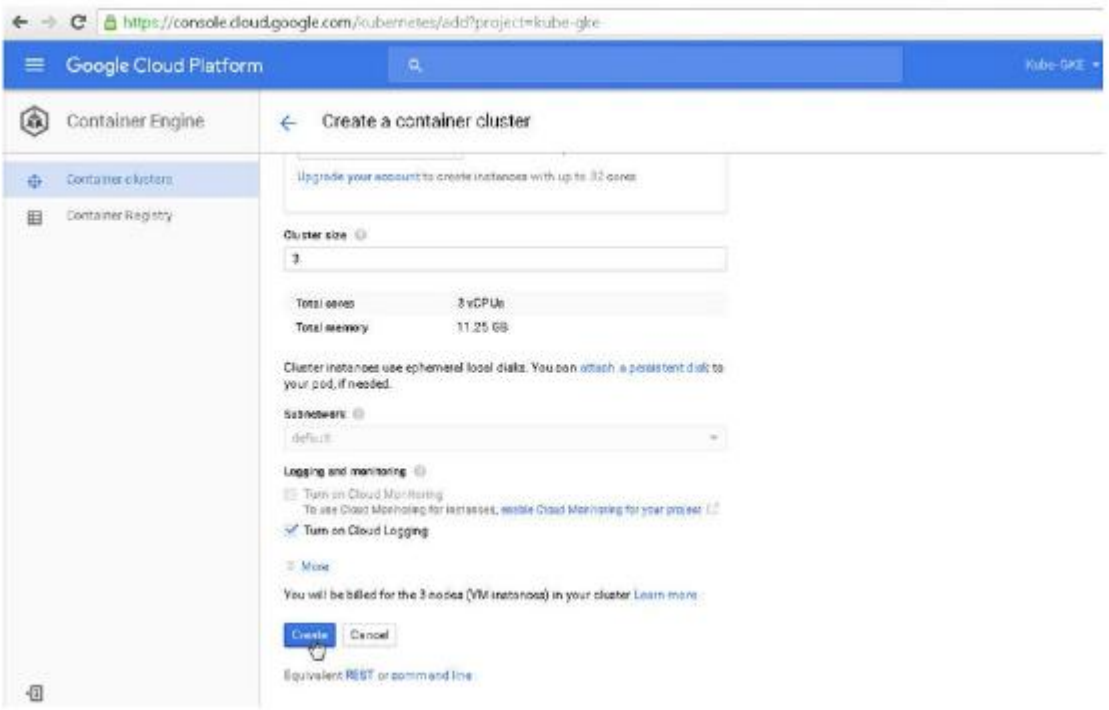
Gambar 3.53 Mengklik Buat Klaster Kontainer

URL https://console.cloud.google.com/kubernetes/add?project=kubernetes-gke (URL mungkin sedikit berbeda) dipanggil, dan formulir input ditampilkan untuk menentukan detail klaster kontainer seperti yang ditunjukkan pada Gambar 3.54.



Gambar 3.54. Menentukan Nama Klaster, Zona, Dan Jenis Mesin

Tentukan nama klaster atau pertahankan nama klaster default (misalnya kube-cluster-1). Pilih Zona, misalnya us-east1-d. Pilih Jenis Mesin sebagai jumlah inti CPU. Misalnya, pilih 1 vCPU, yang memiliki memori 3,75 GB. Tentukan Ukuran Klaster, misalnya 3. Pertahankan pengaturan "default" untuk Subnetwork. Secara opsional pilih Opsi pencatatan dan pemantauan dan klik Buat seperti yang ditunjukkan pada Gambar 3.55.



Gambar 3.55 Membuat Klaster Kontainer

Klaster kontainer dibuat seperti yang ditunjukkan pada Gambar 3.56.

Gambar 3.56 Klaster kontainer kube-cluster-1

Menghubungkan ke Google Cloud Shell

Untuk menghubungkan ke Google Cloud Shell, klik ikon ➤ seperti yang ditunjukkan pada Gambar 3.57. Pesan Selamat Datang di Cloud Shell dan prompt perintah untuk Cloud Shell akan ditampilkan.

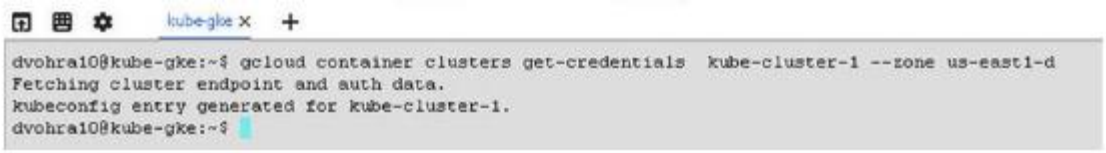
Gambar 3.57 Menghubungkan Ke Google Cloud Shell

3.7 MENGONFIGURASI KUBECTL

Antarmuka baris perintah kubectl digunakan untuk mengelola sumber daya dalam kluster. Jika terdapat lebih dari satu kluster kontainer, kubectl perlu dikonfigurasi agar kluster dapat dikelola. Dengan menggunakan gcloud, yang merupakan alat baris perintah untuk Google Cloud Platform, jalankan perintah berikut untuk mengonfigurasi kubectl ke kluster tertentu. Zona harus disertakan dalam perintah, dengan opsi –zone:

```
gcloud container clusters get-credentials kube-cluster-1 --zone
us-east1-d
```

Titik akhir kluster dan data autentikasi diambil, dan entri kubeconfig dibuat untuk kube-cluster-1 seperti yang ditunjukkan pada Gambar 3.58.



Gambar 3.58 Mengonfigurasi Kubectl Untuk Cluster

Menguji Kluster Kubernetes

Info kluster dapat dicantumkan dengan perintah berikut:

```
kubectl cluster-info
```

Seperti yang ditunjukkan pada Gambar 3.59, master Kubernetes dan komponen kluster lainnya sedang berjalan.

Gambar 3.59 Mencantumkan Info Kluster

Selanjutnya, kita akan membuat aplikasi Node untuk menguji kluster. Buat folder bernama hellonode (atau nama folder lainnya). Di folder hellonode buat file Node server.js dengan editor vi seperti yang ditunjukkan pada Gambar 3.60.

```
dvohra10@kube-gke:~$ cd hellonode
dvohra10@kube-gke:~/hellonode$ ls -l
total 0
dvohra10@kube-gke:~/hellonode$ sudo vi server.js
```

Gambar 3.60 Membuat skrip Node server.js

Skrip Node server.js menanggapi setiap permintaan dengan respons Hello World!.

```
var http = require('http');
var handleRequest = function (request, response) {
  response.writeHead(200);
  response.end('Hello World!');
};
var www = http.createServer(handleRequest);
www.listen(8080);
```

server.js ditampilkan dalam editor vi pada Gambar 3.61.

Gambar 3.61 Skrip Node Server.Js

Selanjutnya, buat file Docker, juga di folder hellonode, untuk menjelaskan citra Docker yang

akan dibangun termasuk port yang didengarkan aplikasi.

```
FROM node: 4.4
EXPOSE 8080
COPY server.js .
CMD node server.js
```

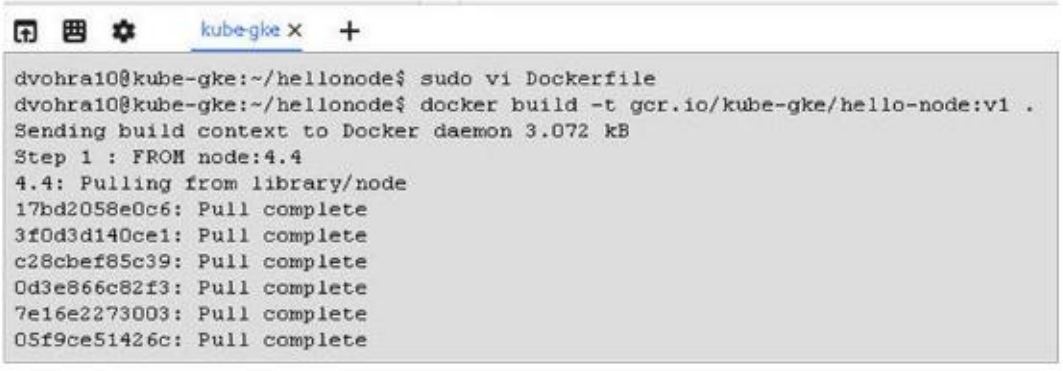
File Docker ditunjukkan pada Gambar 3.62.

Gambar 3.62 Dockerfile

Selanjutnya, buat image Docker menggunakan perintah docker build.

```
docker build -t gcr.io/kube-gke/hello-node:v1.
```

Image Docker node:4.4, tempat image gcr.io/kube-gke/hello-node:v1 dibuat, ditarik seperti yang ditunjukkan pada Gambar 3.63.



Gambar 3.63 Menjalankan Perintah Docker Build

Image Docker dibuat seperti yang ditunjukkan pada Gambar 3.64.

Gambar 3.64 Image Docker Yang Dibuat

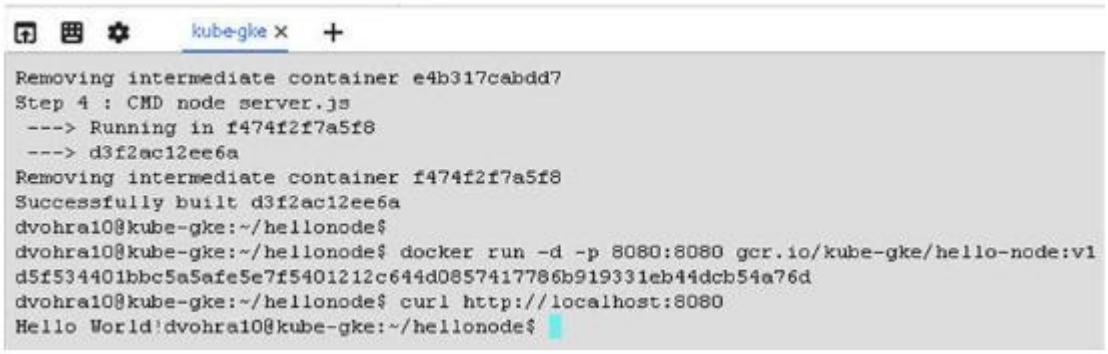
Jalankan citra Docker dengan perintah docker run:

```
docker run -d -p 8080:8080 gcr.io/kube-gke/hello-node:v1
```

Panggil aplikasi dengan perintah curl:

```
curl http://localhost:8080
```

Pesan Hello World! ditampilkan seperti yang ditunjukkan pada Gambar 3.65.



Gambar 3-65. Menjalankan dan Memanggil Aplikasi hello-node

Citra Docker dapat diunggah ke Google Container Registry, dengan perintah berikut:

```
gcloud docker push gcr.io/kube-gke/hello-node:v1
```

Keluaran perintah ditampilkan pada Gambar 3.66.

Gambar 3.66 Mengunggah Citra Docker ke Google Container Registry

Citra Docker diunggah ke repositori seperti yang ditunjukkan pada Gambar 3.67.

Gambar 3.67 Citra Docker yang Diunggah ke Repositori

Citra repositori dapat digunakan untuk membuat penyebaran dan Layanan Kubernetes. Jalankan perintah kubectl run untuk membuat penyebaran:

```
kubectl run hello-node --image=gcr.io/kube-gke/hello-node:v1 --
port=8080
```

Halo-node penyebaran dibuat seperti yang ditunjukkan pada Gambar 3.68.

Gambar 3.68 Membuat Penyebaran

Daftar penyebaran dan pod seperti yang ditunjukkan pada Gambar 3.69, dan Anda akan melihat penyebaran hello-node dan pod dengan awalan hello-node yang tercantum.

Gambar 3.69 Mencantumkan Penyebaran Dan Pod

Buat layanan jenis LoadBalancer untuk penyebaran:

```
kubectl exposed deployment hello-node --type="LoadBalancer"
```

Kemudian jelaskan layanannya. Seperti yang ditunjukkan pada Gambar 3.70, layanan dibuat dan deskripsi layanan menyertakan titik akhir.

Gambar 3.70 Membuat Dan Menjelaskan Layanan

Daftar layanan hello-node, dan cluster-IP, external-IP, dan port untuk layanan tersebut tercantum seperti yang ditunjukkan pada Gambar 3.71.

```
dvohra10@kube-gke:~/hellonode$ kubectl get services hello-node
NAME          CLUSTER-IP       EXTERNAL-IP      PORT(S)    AGE
hello-node    10.127.245.128   104.196.11.21    8080/TCP   3m
dvohra10@kube-gke:~/hellonode$
```

Gambar 3.71 Memperoleh IP Dan Port Eksternal Layanan

Dengan menggunakan perintah external-ip:port, panggil layanan di browser seperti yang ditunjukkan pada Gambar 3.72.

Gambar 3.72 Memanggil Layanan Di Browser

Layanan dan penerapan kini dapat dihapus:

```
kubectl delete service,deployment hello-node
```

Layanan hello-node dan penerapan hello-node dihapus, seperti yang ditunjukkan pada Gambar 3.73.

```
dvohra10@kube-gke:~/hellonode$ kubectl get services hello-node
NAME         CLUSTER-IP       EXTERNAL-IP     PORT(S)    AGE
hello-node   10.127.245.128   104.196.11.21   8080/TCP   14m
dvohra10@kube-gke:~/hellonode$ kubectl delete service,deployment hello-node
service "hello-node" deleted
deployment "hello-node" deleted
```

Gambar 3.73 Menghapus Penerapan Dan Layanan

Kluster kontainer kube-cluster-1 juga dapat dihapus:

```
gcloud container clusters delete kube-cluster-1 --zone us-east1-d
```

Tentukan Y untuk menghapus klaster saat diminta, seperti yang ditunjukkan pada Gambar 3.74.

```
dvohra10@kube-gke:~/hellonode$
dvohra10@kube-gke:~/hellonode$ gcloud container clusters delete kube-cluster-1 --zone us-east1-d
The following clusters will be deleted.
 - [kube-cluster-1] in [us-east1-d]

Do you want to continue (Y/n)?  Y
```

Gambar 3.74 Menghapus Klaster

Ringkasan

Dalam bab ini, kami membahas pembuatan klaster Kubernetes di Google Cloud Platform. Prosedurnya adalah sebagai berikut: Pertama, buat proyek di konsol Google Cloud Platform. Selanjutnya, aktifkan API Compute Engine dan izin. Buat dan hubungkan ke instans mesin virtual dan pesan alamat statis. Buat klaster Kubernetes dan uji klaster dengan membuat aplikasi. Kami juga membahas penggunaan Kubernetes di Google Container Engine. Pada bab berikutnya kita akan membahas penggunaan beberapa zona untuk kluster Kubernetes.

BAB 4
MENGGUNAKAN BEBERAPA ZONA

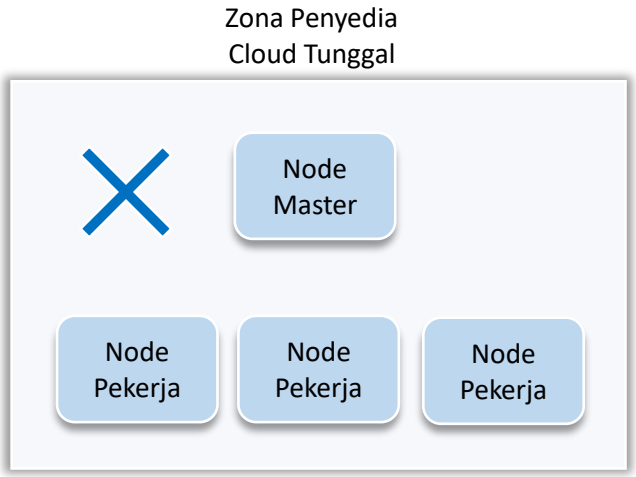
4.1 KETERSEDIAAN TINGGI KUBERNETES DENGAN ZONA KETERSEDIAAN

Ketersediaan tinggi dalam kluster Kubernetes diimplementasikan menggunakan berbagai parameter. Ketersediaan tinggi pengontrol utama akan menyediakan beberapa pengontrol utama. Ketersediaan tinggi etcd akan menyediakan beberapa node etcd. Ketersediaan tinggi DNS publik akan menyediakan beberapa DNS publik. Dalam aplikasi berbasis cloud, ketersediaan kluster akan bergantung pada ketersediaan wilayah atau zona tempat node dijalankan.

AWS menyediakan berbagai pola desain ketersediaan tinggi, seperti Arsitektur Multi Wilayah, Beberapa Penyedia Cloud, Tingkat Penyeimbangan Beban DNS, dan Beberapa Zona Ketersediaan. Dalam bab ini, kita akan membahas pola desain Beberapa Zona Ketersediaan sebagaimana diimplementasikan oleh Kubernetes. Zona ketersediaan Amazon AWS adalah lokasi fisik yang berbeda dengan daya, jaringan, dan keamanan yang independen serta terisolasi dari kegagalan di zona ketersediaan lainnya. Zona ketersediaan dalam wilayah yang sama memiliki konektivitas jaringan latensi rendah di antara keduanya.

Masalah

Jika semua node dalam kluster Kubernetes dijalankan di zona penyedia cloud yang sama (sebagaimana didefinisikan oleh Amazon AWS dan Google Cloud Platform), kegagalan satu zona akan melumpuhkan seluruh kluster Kubernetes seperti yang ditunjukkan pada Gambar 4.1.

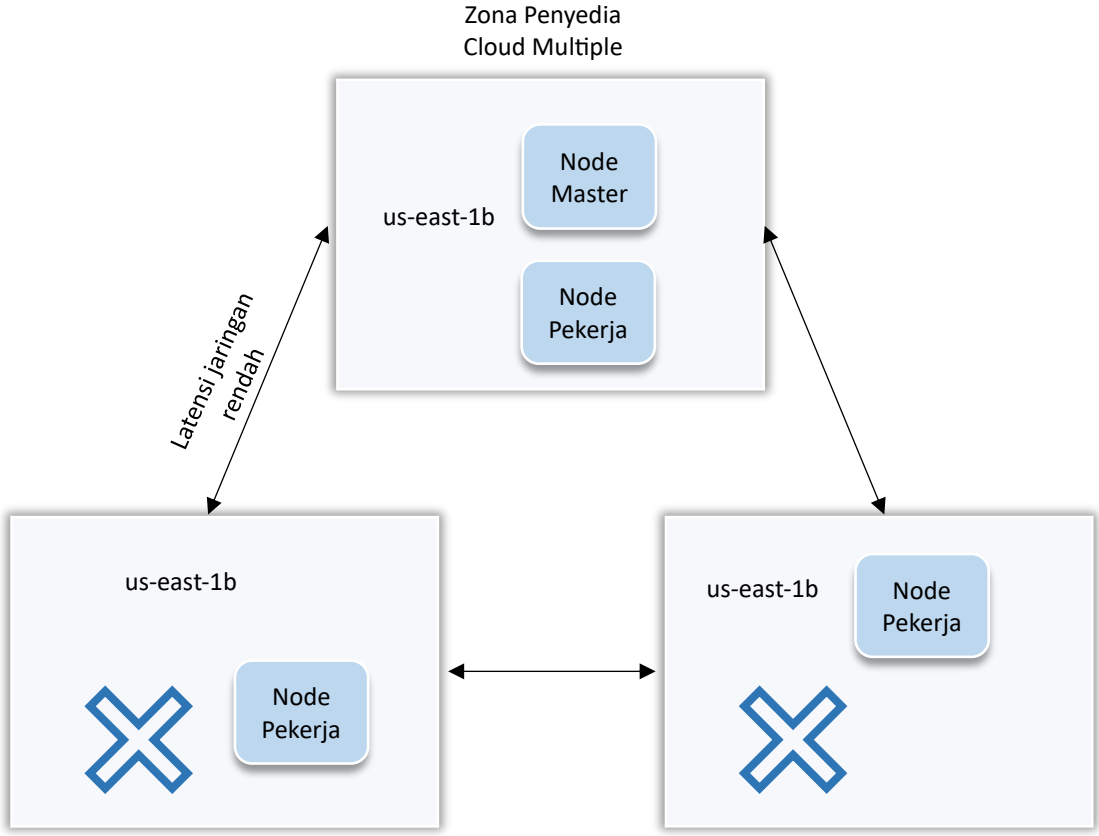


Gambar 4.1 Dalam Cluster Zona Tunggal, Tidak Ada Toleransi Kesalahan Yang Diberikan

Solusi

Dimulai dengan Kubernetes 1.2, sebuah kluster dapat disediakan di beberapa zona penyedia cloud. Pod yang dikelola oleh pengontrol atau layanan replikasi disebarkan di

beberapa zona sehingga kegagalan satu zona tidak memengaruhi ketersediaan pengontrol atau layanan replikasi di zona lain, seperti yang ditunjukkan pada Gambar 4.2.



Gambar 4.2 Kegagalan Dua Zona Dalam Kluster Tiga Zona Tidak Menyebabkan Seluruh Kluster Gagal

Zona hanya didukung dengan penyedia cloud GCE (Google Compute Engine) dan AWS (Amazon Web Services). AWS menyebut zona sebagai "zona ketersediaan." Pod yang menentukan volume persisten ditempatkan di zona yang sama dengan volume. Namun, dukungan untuk zona memiliki beberapa batasan:

- Beberapa zona harus berada di wilayah yang sama. Sebuah kluster tidak boleh mencakup beberapa formasi cloud.
- Zona diasumsikan berada dalam jarak dekat untuk menghindari latensi jaringan karena tidak ada perutean yang memperhatikan zona.
- Kolokasi pod-volume di zona yang sama hanya berlaku untuk volume persisten dan tidak untuk jenis volume lain seperti volume EBS.
- Node berada di beberapa zona, tetapi pengontrol master tunggal dibangun secara default dan pengontrol master berada di satu zona.

Tinjauan Umum

Dalam bab ini, kita akan membuat AWS CloudFormation multizona di CoreOS. Kita juga akan menunjukkan afinitas volume-zona untuk volume persisten pada kluster multizona dengan penyedia cloud AWS. Langkah-langkah yang akan kita ambil adalah sebagai berikut:

- Menetapkan lingkungan
- Menginisialisasi CloudFormation
- Mengonfigurasi cluster.yaml untuk beberapa zona
- Meluncurkan CloudFormation
- Mengonfigurasi DNS Eksternal
- Menjalankan Aplikasi Kubernetes
- Menggunakan Beberapa Zona di AWS

4.2 MENETAPKAN LINGKUNGAN

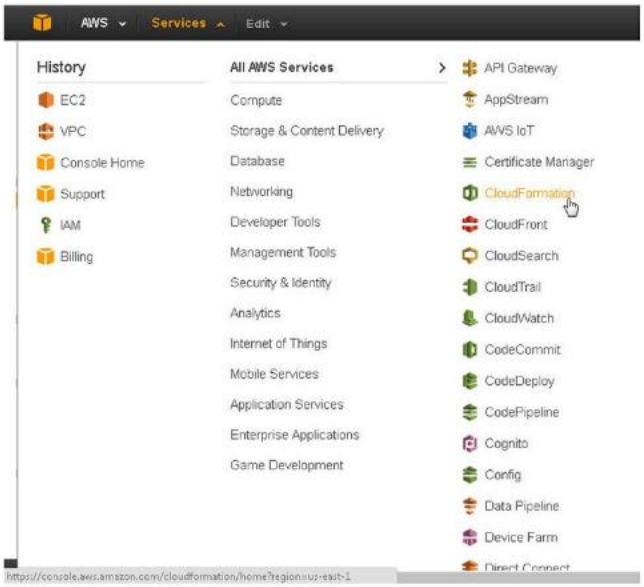
Anda akan menemukan detail tentang pembuatan kluster Kubernetes pada AWS CloudFormation CoreOS di Bab 2. Kita hanya perlu memulai satu instans EC2 untuk meluncurkan CloudFormation. Buat instans EC2 menggunakan Amazon Linux AMI, yang telah menginstal AWS CLI secara default; AWS CLI digunakan untuk menginisialisasi dan meluncurkan CloudFormation. Dapatkan alamat IP Publik instans EC2 dari konsol EC2. Log masuk SSH ke instans EC2:

```
ssh -i "docker.pem" ec2-user@184.73.19.214
```

Prompt perintah Amazon Linux AMI ditampilkan. Karena kita akan meluncurkan AWS CloudFormation untuk kluster Kubernetes, nama tumpukan CloudFormation harus belum pernah digunakan. Jika nama tumpukan CloudFormation sudah digunakan, akan muncul galat seperti berikut (Gambar 4.3).

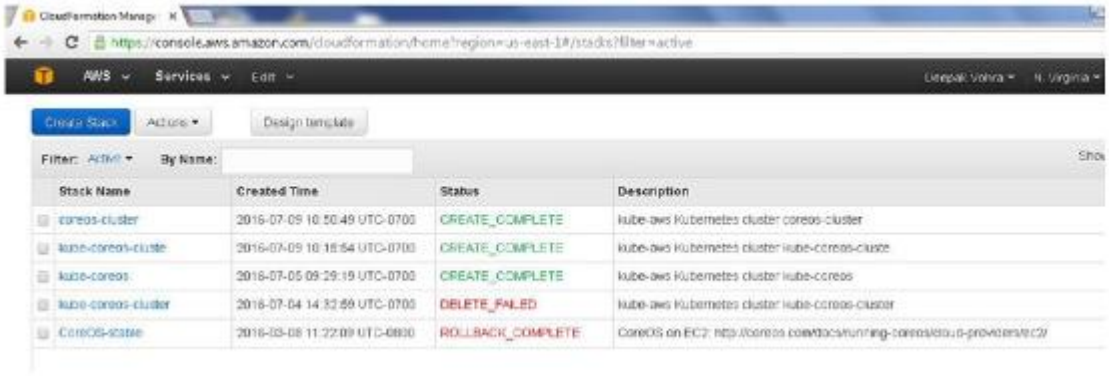
```
[ec2-user@ip-10-0-0-126 coreos-cluster]$ kube-aws up
Creating AWS resources. This should take around 5 minutes.
Error: Error creating cluster: AlreadyExistsException: Stack [kubernetes-coreos-
cluster] already exists
        status code: 400, request id: e1729893-4861-11e6-9731-6334ac51fa70
```

Gambar 4.3 Galat Tumpukan Sudah Ada



Gambar 4.4 Memilih Layanan ➤ Cloudformation

Untuk mengetahui apakah nama tumpukan CloudFormation dapat digunakan, klik Layanan ➤ CloudFormation seperti yang ditunjukkan pada Gambar 4.4. Tumpukan tercantum seperti yang ditunjukkan pada Gambar 4.5. Nama tumpukan yang sama dengan yang tercantum tidak dapat digunakan untuk membuat tumpukan baru.



Gambar 4.5 Mencantumkan Tumpukan Cloudformation

Menginisialisasi CloudFormation

Menginisialisasi tumpukan CloudFormation dibahas secara rinci di Bab 2. Prosedur untuk membuat AWS CloudFormation adalah sebagai berikut:

1. Instal Kube-aws (harus diinstal hanya sekali untuk instans Amazon Linux).
2. Siapkan Parameter Klaster, seperti membuat pasangan kunci EC2 (kubernetes-coreos), kunci KMS, dan nama DNS Eksternal (oramagsearch.com).
3. Buat Direktori Aset untuk klaster CloudFormation.
4. Inisialisasi klaster CloudFormation.
5. Render Konten direktori aset.

Perintah umum untuk membuat pasangan kunci EC2 adalah sebagai berikut:

```
aws ec2 create-key-pair --key-name kubernetes-coreos --query
'KeyMaterial' --output text > kubernetes-coreos.pem

chmod 400 kubernetes-coreos.pem
```

Perintah untuk membuat kunci KMS adalah sebagai berikut:

```
aws kms --region=us-east-1 create-key --description="kube-aws
assets"
```

Salin string KeyMetadata.Arn dan gunakan untuk menginisialisasi tumpukan CloudFormation; misalnya, kluster yang disebut kubernetes-coreos-cluster dengan direktori aset kube-coreos-cluster diinisialisasi sebagai berikut:

```
mkdir kube-coreos-cluster cd kube-coreos-clusterkube-aws init -
-cluster-name=kubernetes-coreos-cluster --external-dns-
name=ORAMAGSEARCH.COM--region=us-east-1 --availability-zone=us-
east-1c --key-name=kubernetes-coreos --kms-key-
arn="arn:aws:kms:us-east-1:xxxxxxxxxx:key/xxxxxxxxxxxxxxxxxxx"
```

Perintah untuk merender konten direktori aset adalah sebagai berikut:

```
kube-aws render
```

4.3 MENGONFIGURASI CLUSTER.YAML UNTUK BEBERAPA ZONA

Secara default, satu zona digunakan untuk meluncurkan CloudFormation. Selanjutnya, kita akan menyesuaikan CloudFormation untuk mengonfigurasi beberapa zona. Buka file cluster.yaml di editor vi:

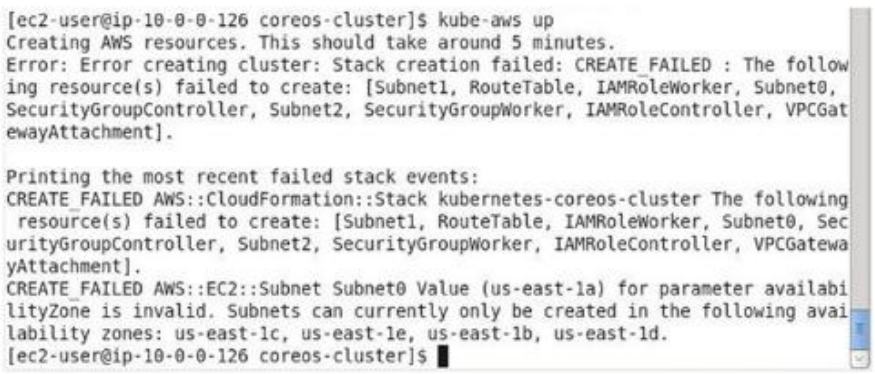
```
sudo vi cluster.yaml
```

Wilayah untuk menyediakan CloudFormation ditetapkan ke us-east-1 sebagaimana ditetapkan dalam perintah kube-aws init. availabilityZone ditetapkan ke us-east-1c juga sebagaimana ditetapkan dalam perintah kube-aws init. Untuk zona ketersediaan ganda atau beberapa zona, beri komentar pada availabilityZone. Secara default workerCount, yang menetapkan jumlah node pekerja yang akan dibuat, ditetapkan ke 1. Untuk mendemonstrasikan kluster zona ganda, node pekerja harus ditetapkan ke setidaknya jumlah zona yang akan dikonfigurasi. Tetapkan workerCount ke 6 seperti yang ditunjukkan pada Gambar 4.6.

Gambar 4.6 Menetapkan Workercount Ke 6

Cluster.yaml dikonfigurasi untuk satu zona ketersediaan secara default, dan pengaturan instanceCIDR menentukan CIDR untuk subnet Kubernetes. Untuk beberapa zona ketersediaan, instanceCIDR harus dikomentari, karena kita perlu mengonfigurasi beberapa subnet di cluster.yaml. Dalam menetapkan subnet, tentukan subnet Kubernetes dan CIDR serta zona ketersediaannya. Tujuan dari ketersediaan tinggi adalah bahwa kegagalan satu zona tidak mengakibatkan gangguan dalam layanan.

Setidaknya dua subnet harus ditentukan untuk ketersediaan zona yang tinggi. Setiap subnet ditentukan sebagai pengaturan availabilityZone dan pengaturan instanceCIDR. Zona ketersediaan yang dapat ditentukan harus tersedia untuk membuat subnet. Jika zona ketersediaan tidak tersedia, kesalahan seperti yang ditunjukkan pada Gambar 4-7 akan muncul saat CloudFormation diluncurkan.



Gambar 4.7 Pesan Kesalahan Saat Subnet Tidak Dapat Dibuat Karena Zona Ketersediaan Tidak Valid

Jalankan perintah berikut untuk menemukan zona ketersediaan.

```
ec2-availability-zones –aws-access-key <access key id> --aws-secret-key <access key>
```

Zona ketersediaan tercantum seperti yang ditunjukkan pada Gambar 4.8. Seperti yang ditunjukkan, zona ketersediaan untuk wilayah us-east-1 adalah us-east-1a, us-east-1b, us-east-1c, us-east-1d, dan us-east-1e.

Gambar 4.8 Mencantumkan Zona Ketersediaan

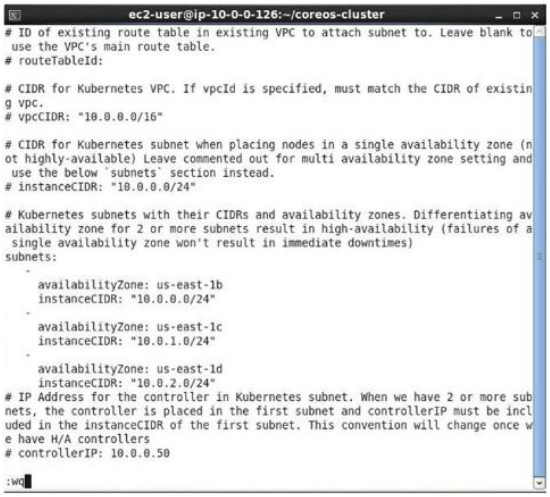
Blok instanceCIDR menentukan rentang IP yang akan digunakan. Ukuran blok harus antara netmask /16 dan netmask /28. Tentukan tiga subnet untuk tiga zona ketersediaan yang berbeda:

```
Subnets:
   –
   availabilityZone: us-east-1b
   instanceCIDR: "10.0.0.0/24"
   –
   availabilityZone: us-east-1c
   instanceCIDR: "10.0.0.0/24"
   –
   availabilityZone: us-east-1d
   instanceCIDR: "10.0.0.0/24"
```

Pengaturan lain yang perlu dikomentari adalah controllerIP. Pengaturan controllerIP menentukan pengontrol dalam subnet Kubernetes. Dengan dua atau lebih subnet, pengontrol ditempatkan di subnet pertama, dan controllerIP harus disertakan dalam instanceCIDR dari subnet pertama. Jika tidak ada instanceCIDR dalam Subnet yang dikonfigurasi yang berisi controllerIP dan controllerIP tidak dikomentari, kesalahan yang ditunjukkan pada Gambar 4-9 akan muncul.

Gambar 4.9 Pesan Kesalahan Saat Tidak Ada Instancecidr Dalam Subnet Yang Dikonfigurasi Yang Berisi Controllerip

Subnet harus diformat seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Mencantumkan Subnet Yang Diformat

Meluncurkan CloudFormation

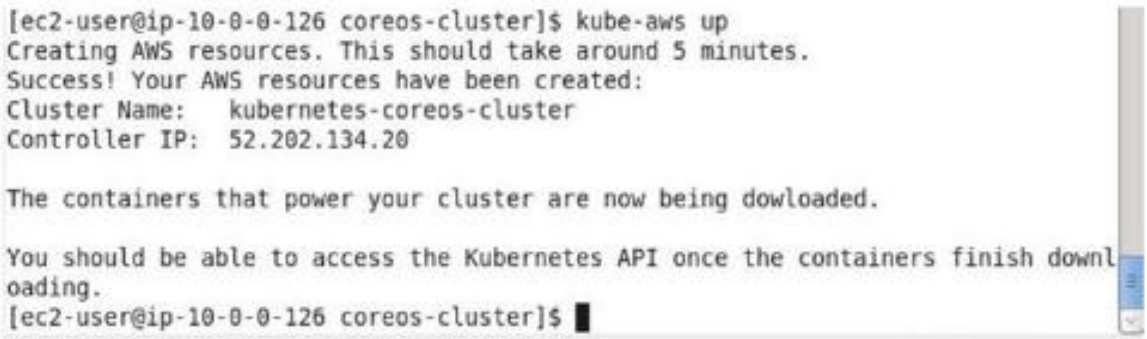
Setelah kita memodifikasi cluster.yaml, tumpukan CloudFormation harus divalidasi. Validasi tumpukan CloudFormation
dengan perintah berikut:

```
kube-aws validate
```

Luncurkan tumpukan CloudFormation.

```
kube-aws up
```

Sumber daya AWS, seperti instans EC2, grup penskalaan, dan konfigurasi peluncuran dibuat, dan CloudFormation diluncurkan seperti yang ditunjukkan pada Gambar 4.11.



Gambar 4.11 Meluncurkan Cloudformation

Status CloudFormation dapat ditemukan dengan perintah berikut:

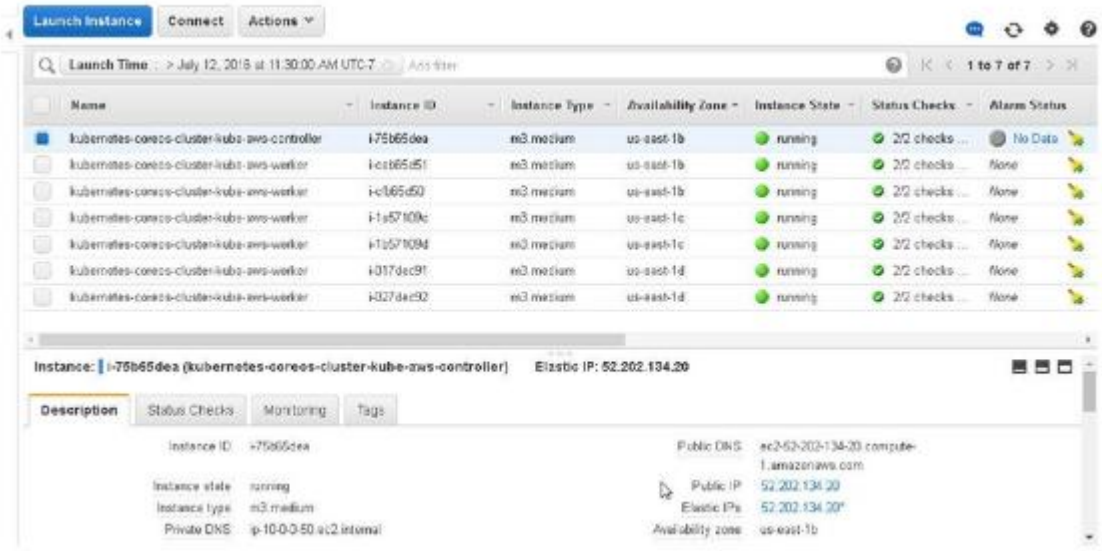
```
kube-aws status
```

IP pengontrol tercantum, seperti yang ditunjukkan pada Gambar 4.12.

```
[ec2-user@ip-10-0-0-126 coreos-cluster]$ kube-aws status
Cluster Name:   kubernetes-coreos-cluster
Controller IP:  52.202.134.20
[ec2-user@ip-10-0-0-126 coreos-cluster]$
```

Gambar 4.12 Menemukan status CloudFormation

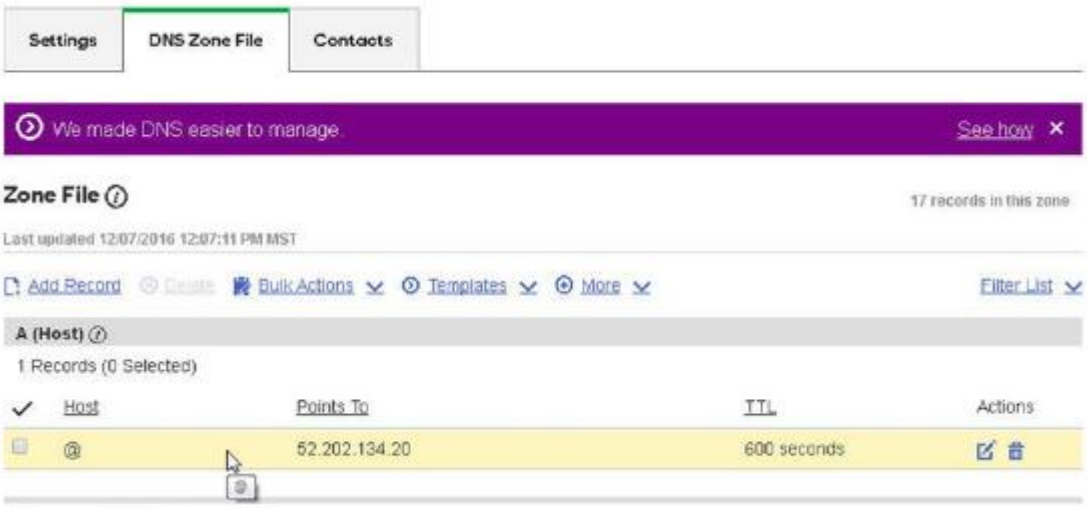
Instans EC2 yang diluncurkan oleh tumpukan CloudFormation ditunjukkan pada Gambar 4.13. Seperti yang ditunjukkan di kolom Availability Zone, dua instans masing-masing diluncurkan di zona us-east-1b, us-east-1c, dan us-east-1d. Pengontrol tunggal berjalan di zona us-east-1b.



Gambar 4.13 Mencantumkan Subnet Yang Diformat

4.4 MENGONFIGURASI DNS EKSTERNAL

Konfigurasikan alamat IP Publik dari instans pengontrol dalam nama DNS Publik untuk domain nosqlsearch.com pada pendaftar domain. Tambahkan rekaman A untuk IP Publik dari instans pengontrol seperti yang ditunjukkan pada Gambar 4.14.



Gambar 4.14 Mencantumkan Subnet Yang Diformat

Menjalankan Aplikasi Kubernetes

Selanjutnya, kita akan menguji kluster Kubernetes untuk memastikan bahwa pod dalam aplikasi benar-benar dialokasikan di seluruh node di zona yang berbeda. Hubungkan ke instance pengontrol:

```
ssh -i "kubernetes-coreos.pem" core@52.202.134.20
```

Instance pengontrol login seperti yang ditunjukkan pada Gambar 4.15.

Gambar 4.15 Login SSH Ke Instance Coreos Pengontrol

Instal biner kubectl dan atur izin.

```
sudo wget https://storage.googleapis.com/kubernetes-
release/release/v1.3.0/bin/linux/ amd64/./kubectl
```

```
sudo chmod +x ./kubectl
```

Biner Kubectl diinstal. Pindahkan biner kubectl ke /usr/local/bin/, yang ada di jalur:

```
sudo mv ./kubectl /usr/local/bin/
```

Daftar node dalam kluster:

```
./kubectl get nodes
```

Node master tunggal dan enam node pekerja tercantum seperti yang ditunjukkan pada Gambar 4.16.

Gambar 4.16 Mencantumkan Node Dalam Kluster Kubernetes

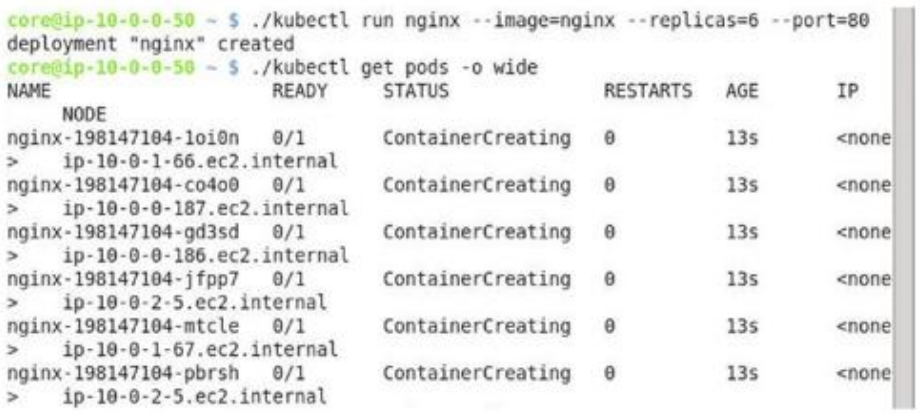
Jalankan image Docker nginx untuk membuat enam 86eplica pod:

```
kubectl run nginx –image=nginx –replicas=6 –port=80
```

Selanjutnya, daftarkan pod:

```
kubectl get pods –o wide
```

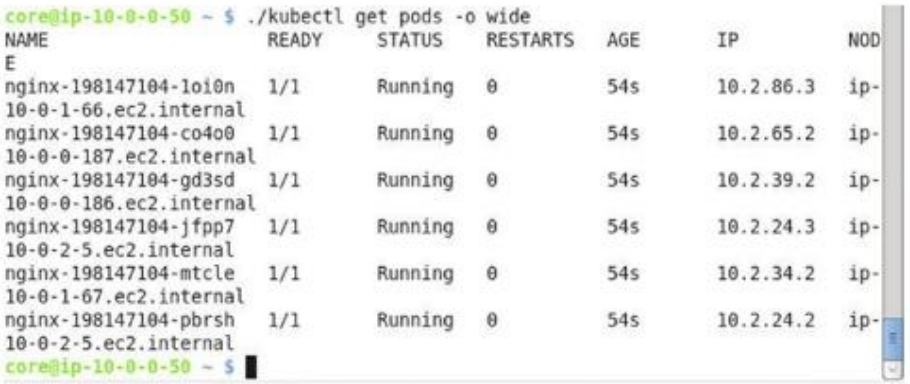
Penerapan nginx dibuat dan pod dicantumkan. Awalnya pod mungkin dicantumkan sebagai tidak siap, seperti yang ditunjukkan oleh nilai kolom READY sebesar 0/1 dan nilai kolom STATUS ContainerCreating pada Gambar 4.17.



Gambar 4.17 Menjalankan Aplikasi Nginx Kubernetes

Jalankan perintah kubectl get pods –o wide lagi setelah beberapa detik (hingga satu menit) dan semua pod akan berjalan dan siap seperti yang ditunjukkan pada Gambar 4.18. Seperti

yang ditunjukkan pada kolom NODE, masing-masing dari enam pod berjalan pada node yang berbeda, yang berarti bahwa pod tersebar di seluruh zona dalam kluster. Kegagalan satu zona tidak akan memengaruhi ketersediaan penerapan.



Gambar 4.18 Semua Pod Berjalan Dan Siap

Menggunakan Beberapa Zona di AWS

Jika klaster Kubernetes akan dimulai dengan kapabilitas multizona, parameter MULTIZONE harus ditetapkan ke true. Menetapkan MULTIZONE ke true tidak secara otomatis memulai node yang berjalan di beberapa zona; ini hanya menambahkan kapabilitas untuk mengelola klaster multizona. Jika node klaster akan dijalankan di beberapa zona, beberapa set node harus dimulai di zona terpisah menggunakan pengontrol utama yang sama dengan set node zona pertama.

Saat set node dimulai di klaster yang sadar zona, node diberi label yang menunjukkan zona tempat node berjalan. Pertama, mulai klaster yang sadar multizona menggunakan penyedia AWS Kubernetes dengan menetapkan MULTIZONE=true. Menetapkan KUBE_AWS_ZONE ke true akan membuat node pengontrol utama dan semua node minion di zona yang ditentukan. Nilai NUM_NODES menetapkan jumlah node yang akan dibuat. Jalankan perintah berikut untuk memulai klaster di zona us-east-1c dengan tiga node:

```
curl -sS https://get.k8s.io | MULTIZONE=true
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-east- 1c NUM_NODES=3
bash
```

Biner Kubernetes diunduh dengan perintah MULTIZONE, seperti yang ditunjukkan pada Gambar 4.19.

```
[ec2-user@ip-10-0-0-126 ~]$ curl -sS https://get.k8s.io | MULTIZONE=true KUBERNE
TES_PROVIDER=aws KUBE_AWS_ZONE=us-east-1c NUM_NODES=3 bash
Downloading kubernetes release v1.3.0 to /home/ec2-user/kubernetes.tar.gz
--2016-07-12 16:24:03--  https://storage.googleapis.com/kubernetes-release/relea
se/v1.3.0/kubernetes.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)... 173.194.204.128, 26
07:f8b0:400d:c00::80
Connecting to storage.googleapis.com (storage.googleapis.com)|173.194.204.128|:4
43... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1486828686 (1.4G) [application/x-tar]
Saving to: 'kubernetes.tar.gz'

kubernetes.tar.gz    13%[=>                  ] 195.03M  18.0MB/s    eta 66s
```

Gambar 4.19 Memulai Klaster Yang Mendukung Multizona

Kluster Kubernetes multizona dimulai seperti yang ditunjukkan pada Gambar 4.20. Yang membedakan klaster ini adalah klaster ini mendukung multizona.

```
Done, listing cluster services:

Kubernetes master is running at https://52.206.28.220
Elasticsearch is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-s
ystem/services/elasticsearch-logging
Heapster is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/heapster
Kibana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/s
ervices/kibana-logging
KubeDNS is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/kube-dns
kubernetes-dashboard is running at https://52.206.28.220/api/v1/proxy/namespaces
/kube-system/services/kubernetes-dashboard
Grafana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/monitoring-grafana
InfluxDB is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/monitoring-influxdb

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

Kubernetes binaries at /home/ec2-user/kubernetes/cluster/
You may want to add this directory to your PATH in $HOME/.profile
Installation successful!
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.20 Memulai Klaster Yang Mendukung Multizona

Daftar node dengan kubectl get nodes seperti yang ditunjukkan pada Gambar 4.21.

```
[ec2-user@ip-10-0-0-126 ~]$ kubectl get nodes
NAME                          STATUS    AGE
ip-172-20-0-239.ec2.internal   Ready     8m
ip-172-20-0-240.ec2.internal   Ready     8m
ip-172-20-0-241.ec2.internal   Ready     8m
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.21 Mencantumkan Node

Berikutnya, cantumkan node dan sertakan label yang akan dicantumkan seperti yang ditunjukkan pada Gambar 4.22.

```
kubectl get nodes --show-labels
```

Label tersebut mencakup `failure-domain.beta.kubernetes.io/region` untuk region dan `failure-domain.beta.kubernetes.io/zone` untuk zone.

```
[ec2-user@ip-10-0-0-126 ~]$ kubectl get nodes --show-labels
NAME                          STATUS    AGE       LABELS
ip-172-20-0-239.ec2.internal   Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-239.ec2.internal
ip-172-20-0-240.ec2.internal   Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-240.ec2.internal
ip-172-20-0-241.ec2.internal   Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-241.ec2.internal
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.22 Mencantumkan Node Termasuk Labelnya

Seperti yang ditunjukkan di konsol EC2 pada Gambar 4.23, semua node berjalan di zona yang sama, us-east-1c. Mengapa zona yang sama meskipun MULTIZONE diatur ke true? Karena pengaturan tersebut membuat klaster menjadi multi-zona dan bukan multi-zona untuk memulai. Kita akan membahas selanjutnya tentang penambahan set node di zona lain menggunakan pengendali utama yang sama.

Gambar 4.23 Semua Node Di Zona Yang Sama, Us-East-1c

Selanjutnya, mulai set node lain di zona yang berbeda tetapi menggunakan master yang sama dengan set node pertama. Dapatkan IP Pribadi dari instans master dari konsol `EC2`. Jalankan perintah berikut di mana `MASTER_INTERNAL_IP` menentukan IP pribadi pengendali utama dan `KUBE_SUBNET_CIDR` menentukan `CIDR subnet`. `KUBE_USE_EXISTING_MASTER` diatur ke true, yang menyiratkan bahwa master yang ada akan digunakan. `KUBE_AWS_ZONE` diatur ke zona yang berbeda, `us-east-1b`.

```
KUBE_USE_EXISTING_MASTER=true                    MULTIZONE=true
KUBERNETES_PROVIDER=aws  KUBE_AWS_ZONE=us-east-  1b  NUM_NODES=3
KUBE_SUBNET_CIDR=172.20.1.0/24    MASTER_INTERNAL_IP=172.20.0.9
kubernetes/
cluster/kube-up.sh
```

Simpul lain yang diatur di zona yang berbeda, us-east-1b, dimulai seperti yang ditunjukkan oleh keluaran perintah pada Gambar 4.24.

```
[ec2-user@ip-10-0-0-126 ~]$ KUBE_USE_EXISTING_MASTER=true MULTIZONE=true KUBERNE
TES_PROVIDER=aws KUBE_AWS_ZONE=us-east-1b NUM_NODES=3 KUBE_SUBNET_CIDR=172.20.1.
0/24 MASTER_INTERNAL_IP=172.20.0.9 kubernetes/cluster/kube-up.sh
Using subnet CIDR override: 172.20.1.0/24
... Starting cluster in us-east-1b using provider aws
... calling verify-prereqs
... calling kube-up
Starting cluster using os distro: jessie
Uploading to Amazon S3
+++ Staging server tars to S3 Storage: kubernetes-staging-3b2de58189ba7d2340027c
ecbbbe5060/devel
upload: ../../tmp/kubernetes.LOSThs/s3/bootstrap-script to s3://kubernetes-stagi
ng-3b2de58189ba7d2340027cecbbbe5060/devel/bootstrap-script
Uploaded server tars:
  SERVER_BINARY_TAR_URL: https://s3.amazonaws.com/kubernetes-staging-3b2de58189b
a7d2340027cecbbbe5060/devel/kubernetes-server-linux-amd64.tar.gz
  SALT_TAR_URL: https://s3.amazonaws.com/kubernetes-staging-3b2de58189ba7d234002
7cecbbbe5060/devel/kubernetes-salt.tar.gz
  BOOTSTRAP_SCRIPT_URL: https://s3.amazonaws.com/kubernetes-staging-3b2de58189ba
7d2340027cecbbbe5060/devel/bootstrap-script
INSTANCEPROFILE arn:aws:iam::672593526685:instance-profile/kubernetes-master   2
016-01-29T00:18:58Z     AIPAJR6YCBYPX27F553HI   kubernetes-master       /
ROLES   arn:aws:iam::672593526685:role/kubernetes-master        2016-01-29T00:18
:57Z    /       AROAIDG4HG76MJPGRWEEW   kubernetes-master
ASSUMEROLEPOLICYDOCUMENT        2012-10-17
STATEMENT       sts:AssumeRole  Allow
PRINCIPAL       ec2.amazonaws.com
INSTANCEPROFILE arn:aws:iam::672593526685:instance-profile/kubernetes-minion   2
016-01-29T00:19:00Z     AIPAJHMVQBPLMRBJE5MNO   kubernetes-minion       /
ROLES   arn:aws:iam::672593526685:role/kubernetes-minion        2016-01-29T00:18
:59Z    /       AROAJU44B2VYHK5GKUB3S   kubernetes-minion
```

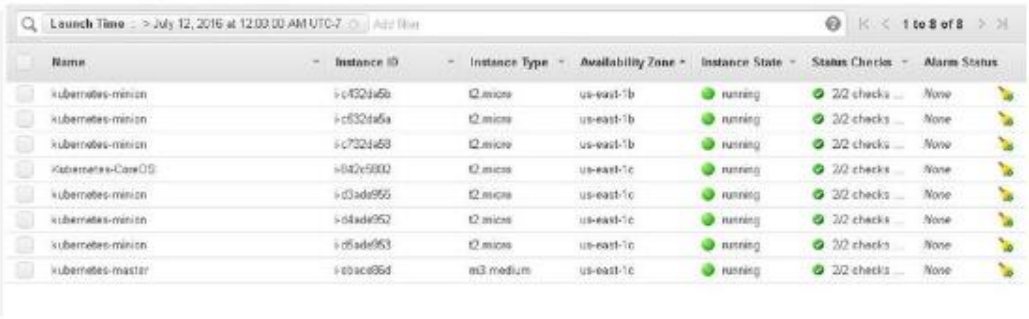
Gambar 4.24 Memulai Kluster Node Kubernetes Di Zona Lain, Us-East-1b

Seperti yang ditunjukkan oleh output pada Gambar 4.25, IP master sama tetapi CIDR subnet berbeda.

```
ip-172-20-0-241.ec2.internal   Ready     40m
Validate output:
Using subnet CIDR override: 172.20.1.0/24
NAME                 STATUS    MESSAGE             ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
Cluster validation succeeded
Done, listing cluster services:

Using subnet CIDR override: 172.20.1.0/24
Kubernetes master is running at https://52.206.28.220
Elasticsearch is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-s
ystem/services/elasticsearch-logging
Heapster is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/heapster
Kibana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/s
ervices/kibana-logging
KubeDNS is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/kube-dns
kubernetes-dashboard is running at https://52.206.28.220/api/v1/proxy/namespaces
/kube-system/services/kubernetes-dashboard
Grafana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/monitoring-grafana
InfluxDB is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/monitoring-influxdb

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.25 IP Induk Yang Sama Tetapi CIDR Subnet Yang Berbeda

Konsol EC2 mencantumkan kumpulan node lain di zona yang berbeda, us-east-1b seperti yang ditunjukkan pada Gambar 4.26. Kluster hanya memiliki satu induk di zona us-east-1c tetapi minion di zona yang berbeda, us-east-1b dan us-east-1c.



Gambar 4.26 IP Induk Yang Sama Tetapi CIDR Subnet Yang Berbeda

Mencantumkan node akan menampilkan enam node, seperti yang ditunjukkan pada Gambar 4.27.

```
[ec2-user@ip-10-0-0-126 ~]$ kubectl get nodes
NAME                          STATUS    AGE
ip-172-20-0-239.ec2.internal   Ready     56m
ip-172-20-0-240.ec2.internal   Ready     56m
ip-172-20-0-241.ec2.internal   Ready     56m
ip-172-20-1-96.ec2.internal    Ready     10m
ip-172-20-1-97.ec2.internal    Ready     10m
ip-172-20-1-98.ec2.internal    Ready     10m
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.27 Mencantumkan Node Dalam Dua Zona Berbeda

Mencantumkan node termasuk label akan menampilkan enam node, tiga di zona us-east-1c dan tiga di us-east-1b, seperti yang ditunjukkan pada Gambar 4.28.

```
[ec2-user@ip-10-0-0-126 ~]$  kubectl get nodes --show-labels
NAME                          STATUS    AGE       LABELS
ip-172-20-0-239.ec2.internal   Ready     57m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-239.ec2.internal
ip-172-20-0-240.ec2.internal   Ready     57m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-240.ec2.internal
ip-172-20-0-241.ec2.internal   Ready     57m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-241.ec2.internal
ip-172-20-1-96.ec2.internal    Ready     11m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1b,kubernetes.io/hostname=ip-172-20-1-96.ec2.internal
ip-172-20-1-97.ec2.internal    Ready     11m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1b,kubernetes.io/hostname=ip-172-20-1-97.ec2.internal
ip-172-20-1-98.ec2.internal    Ready     11m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1b,kubernetes.io/hostname=ip-172-20-1-98.ec2.internal
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.28 Mencantumkan Node Dalam Dua Zona Termasuk Label

Luncurkan kumpulan node lain di zona `us-east-1d` menggunakan node induk yang sama. Tentukan CIDR subnet yang berbeda untuk `zona us-east-1d`.

```
KUBE_USE_EXISTING_MASTER=benar MULTIZONE=benar
KUBERNETES_PROVIDER=aws KUBE_AWS_ZONE=us-east- 1d NUM_NODE=3
KUBE_SUBNET_CIDR=172.20.2.0/24 MASTER_INTERNAL_IP=172.20.0.9
kubernetes/
cluster/kube-up.sh
```

Kumpulan node dimulai di zona us-east-1d seperti yang ditunjukkan pada Gambar 4.29.

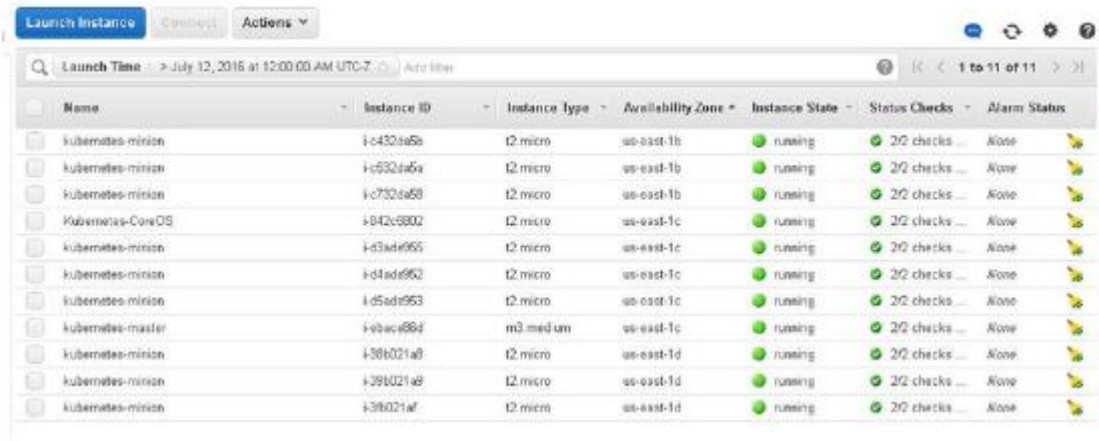
Gambar 4.29 Meluncurkan Kluster Di Zona Us-East-1d

Seperti yang ditunjukkan oleh keluaran kluster pada Gambar 4.30, IP induknya sama tetapi CIDR subnetnya berbeda.

```
ip-172-20-1-98.ec2.internal    Ready     28m
Validate output:
Using subnet CIDR override: 172.20.2.0/24
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-1               Healthy   {"health": "true"}
etcd-0               Healthy   {"health": "true"}
Cluster validation succeeded
Done, listing cluster services:

Using subnet CIDR override: 172.20.2.0/24
Kubernetes master is running at https://52.206.28.220
Elasticsearch is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-s
ystem/services/elasticsearch-logging
Heapster is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/heapster
Kibana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/s
ervices/kibana-logging
KubeDNS is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/kube-dns
kubernetes-dashboard is running at https://52.206.28.220/api/v1/proxy/namespaces
/kube-system/services/kubernetes-dashboard
Grafana is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system/
services/monitoring-grafana
InfluxDB is running at https://52.206.28.220/api/v1/proxy/namespaces/kube-system
/services/monitoring-influxdb

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.30 IP Induk Yang Sama Tetapi CIDR Subnet Yang Berbeda

Konsol EC2 mencantumkan tiga set minion, masing-masing satu di zona us-east-1b, us-east-1c, dan us-east-1d seperti yang ditunjukkan pada Gambar 4.31. Satu induk berada di zona us-east-1c.



Gambar 4.31 Mencantumkan Node Dalam Tiga Zona

Mencantumkan node akan menampilkan 9 node. Beberapa node mungkin awalnya dalam status NotReady saat rangkaian node dimulai, seperti yang ditunjukkan pada Gambar 4.32.

```
[ec2-user@ip-10-0-0-126 ~]$ kubectl get nodes
NAME                           STATUS     AGE
ip-172-20-0-239.ec2.internal   Ready      1h
ip-172-20-0-240.ec2.internal   Ready      1h
ip-172-20-0-241.ec2.internal   Ready      1h
ip-172-20-1-96.ec2.internal    Ready      34m
ip-172-20-1-97.ec2.internal    Ready      34m
ip-172-20-1-98.ec2.internal    Ready      34m
ip-172-20-2-23.ec2.internal    NotReady   30s
ip-172-20-2-24.ec2.internal    NotReady   27s
ip-172-20-2-25.ec2.internal    NotReady   26s
[ec2-user@ip-10-0-0-126 ~]$ kubectl get nodes
NAME                           STATUS    AGE
ip-172-20-0-239.ec2.internal   Ready     1h
ip-172-20-0-240.ec2.internal   Ready     1h
ip-172-20-0-241.ec2.internal   Ready     1h
ip-172-20-1-96.ec2.internal    Ready     35m
ip-172-20-1-97.ec2.internal    Ready     35m
ip-172-20-1-98.ec2.internal    Ready     35m
ip-172-20-2-23.ec2.internal    Ready     44s
ip-172-20-2-24.ec2.internal    Ready     41s
ip-172-20-2-25.ec2.internal    Ready     40s
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.32 Mencantumkan Node Kubernetes

Menyertakan label akan mencantumkan node yang berada di tiga zona berbeda, seperti yang ditunjukkan pada Gambar 4.33.

```
[ec2-user@ip-10-0-0-126 ~]$ kubectl get node ip-172-20-1-96.ec2.internal ip-172-
20-0-241.ec2.internal ip-172-20-2-23.ec2.internal ip-172-20-2-25.ec2.internal ip
-172-20-1-98.ec2.internal ip-172-20-2-25.ec2.internal ip-172-20-1-96.ec2.interna
l ip-172-20-1-97.ec2.internal ip-172-20-0-240.ec2.internal ip-172-20-2-24.ec2.in
ternal --show-labels
NAME                           STATUS    AGE       LABELS
ip-172-20-1-96.ec2.internal    Ready     43m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1b,kubernetes.io/hostname=ip-172-20-1-96.ec2.internal
ip-172-20-0-241.ec2.internal   Ready     1h        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1c,kubernetes.io/hostname=ip-172-20-0-241.ec2.internal
ip-172-20-2-23.ec2.internal    Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1d,kubernetes.io/hostname=ip-172-20-2-23.ec2.internal
ip-172-20-2-25.ec2.internal    Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1d,kubernetes.io/hostname=ip-172-20-2-25.ec2.internal
ip-172-20-1-98.ec2.internal    Ready     43m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1b,kubernetes.io/hostname=ip-172-20-1-98.ec2.internal
ip-172-20-2-25.ec2.internal    Ready     8m        beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
=us-east-1d,kubernetes.io/hostname=ip-172-20-2-25.ec2.internal
ip-172-20-1-96.ec2.internal    Ready     43m       beta.kubernetes.io/arch=amd64
,beta.kubernetes.io/instance-type=t2.micro,beta.kubernetes.io/os=linux,failure-d
omain.beta.kubernetes.io/region=us-east-1,failure-domain.beta.kubernetes.io/zone
```

Gambar 4.33 Mencantumkan Node Yang Menyertakan Label

PersistentVolume (PV) adalah penyimpanan jaringan yang disediakan dalam kluster, dan PersistentVolumeClaim (PVC) adalah permintaan penyimpanan oleh pengguna. PVC menggunakan sumber daya PV seperti halnya pod menggunakan sumber daya node. Selanjutnya, kita akan membuat klaim volume persisten dan selanjutnya mengklaim volume tersebut dalam spesifikasi pod. Tujuan aplikasi ini adalah untuk menunjukkan bahwa volume persisten tidak dapat dipasang di seluruh zona. Volume persisten diberi label dengan zona tempat volume tersebut dibuat, dan pod yang menggunakan volume persisten tersebut dialokasikan di zona yang sama dengan volume persisten. Pertama, buat file spesifikasi JSON claim.yaml untuk klaim volume persisten:

```
sudo vi claim1.json
```

Salin kode sumber berikut ke claim.json:

```
{
kind: PersistentVolumeClaim
apiVersion: v1
metadata:{
  name: claim1
  annotations:{
    volume.alpha.kubernetes.io/storage-class: "foo"
  }
}
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: "3Gi"
```

Claim1.json yang dihasilkan ditunjukkan dalam editor vi pada Gambar 4.34.

```
{
  "kind": "PersistentVolumeClaim",
  "apiVersion": "v1",
  "metadata": {
    "name": "claim1",
    "annotations": {
        "volume.alpha.kubernetes.io/storage-class": "foo"
    }
  },
  "spec": {
    "accessModes": [
      "ReadWriteOnce"
    ],
    "resources": {
      "requests": {
        "storage": "3Gi"
      }
    }
  }
}
```

Gambar 4.34 Klaim PersistentVolumeClaim1.json

Buat PVC dengan perintah kubectl create:

```
kubectl create -f claim1.json
```

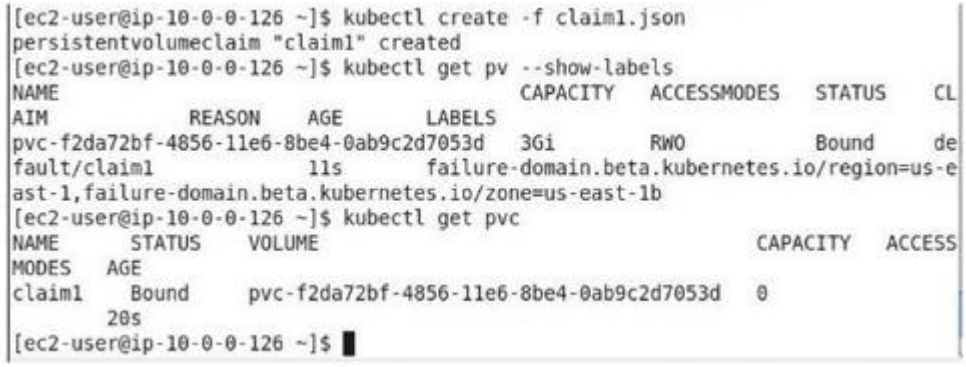
Daftar volume persisten, termasuk label:

```
kubectl get pv --show-labels
```

Volume persisten dicantumkan sebagai yang digunakan oleh klaim volume persisten:

```
kubectl get pvc
```

Seperti yang ditunjukkan oleh keluaran perintah, persistentvolumeclaim dibuat. Volume persisten dicantumkan berada di zona us-east-1b seperti yang ditunjukkan pada Gambar 4.35.



```
[ec2-user@ip-10-0-0-126 ~]$ kubectl create -f claim1.json
persistentvolumeclaim "claim1" created
[ec2-user@ip-10-0-0-126 ~]$ kubectl get pv --show-labels
NAME                                       CAPACITY   ACCESSMODES   STATUS    CL
AIM            REASON    AGE       LABELS
pvc-f2da72bf-4856-11e6-8be4-0ab9c2d7053d   3Gi        RWO           Bound     de
fault/claim1             11s       failure-domain.beta.kubernetes.io/region=us-e
ast-1,failure-domain.beta.kubernetes.io/zone=us-east-1b
[ec2-user@ip-10-0-0-126 ~]$ kubectl get pvc
NAME      STATUS    VOLUME                                     CAPACITY   ACCESS
MODES   AGE
claim1    Bound     pvc-f2da72bf-4856-11e6-8be4-0ab9c2d7053d   0
        20s
[ec2-user@ip-10-0-0-126 ~]$
```

Gambar 4.35 Persistentvolumeclaim Claim1.Json

Selanjutnya, tentukan spesifikasi pod yang menggunakan PVC.

```
sudo vi pod.yaml
```

Salin kode berikut ke pod.yaml:

```
---
apiVersion:v1
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim1
  annotations:
    volume.alpha.kubernetes.io/storage-class: "foo"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: "3Gi"
name: pv
persistentVolumeClaim
  claimName: claim1
```

Pod.yaml yang dihasilkan ditampilkan dalam editor vi pada Gambar 4.36.

Gambar 4.36 Menggunakan Persistentvolumeclaim Claim1.Json Dalam Sebuah Pod

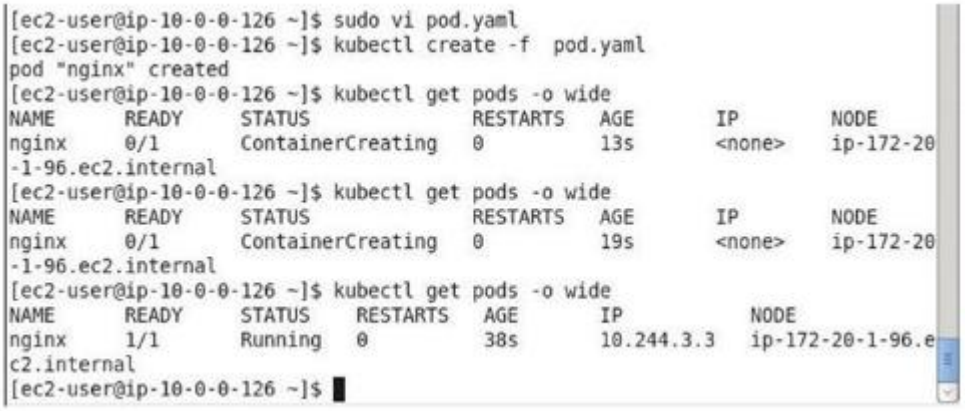
Buat sebuah pod dari pod.yaml:

```
./kubectl create -f pod.yaml
```

Sebuah pod dibuat. Berikutnya, daftarkan pod di seluruh kluster:

```
kubectl get pods -o wide
```

Node tempat pod berjalan akan dicantumkan, seperti yang ditunjukkan pada Gambar 4.37.

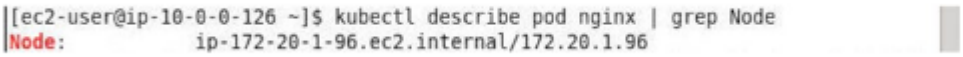


Gambar 4.37 Membuat Sebuah Pod Dan Mencantumkan Node-Nya

Alternatifnya, dapatkan IP Node sebagai berikut:

```
kubectl describe deployment nginx | grep Node
```

IP Node ditampilkan seperti yang ditunjukkan pada Gambar 4.38.

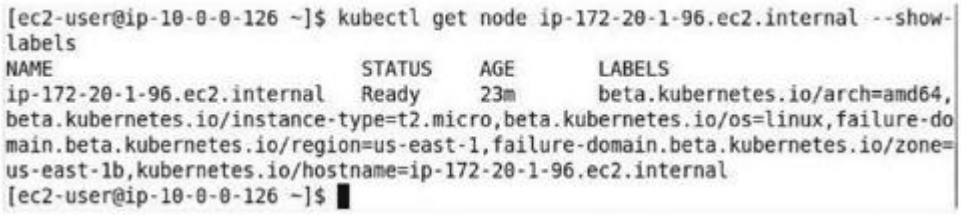


Gambar 4.38 Persistentvolumeclaim Claim1.Json

Berikutnya, cantumkan label node:

```
kubectl get node <node ip> --show-labels
```

Node berjalan di zona us-east-1b, yang sama dengan zona volume persisten, seperti yang ditunjukkan pada Gambar 4.39.



Gambar 4.39 Node Dijadwalkan Pada Zona Yang Sama Dengan Volume Persisten

Ringkasan

Dalam bab ini, kami membuat kluster Kubernetes menggunakan beberapa zona pada CoreOS. Kluster multizona adalah kluster dengan ketersediaan tinggi. Kluster multizona dikonfigurasi dengan menentukan beberapa subnet Kubernetes beserta `CIDR` dan zona ketersediaannya di cluster.yaml di bagian subnet. Kami juga membahas pembuatan kluster multizona pada penyedia cloud AWS dengan menyetel parameter `MULTIZONE` ke true dalam perintah `curl -sS https://get.k8s.io` untuk meluncurkan kluster Kubernetes. Pada bab berikutnya, kami akan membahas penggunaan konsol Tectonic.

BAB 5
MENGGUNAKAN TECTONIC CONSOLE

5.1 MENGELOLA KLUSTER KUBERNETES DENGAN TECTONIC CONSOLE

Tectonic adalah platform Kubernetes perusahaan komersial yang menyediakan keamanan, skalabilitas, dan keandalan tingkat perusahaan. Tectonic menyediakan platform terintegrasi berdasarkan Kubernetes dan CoreOS Linux. Arsitektur Tectonic terdiri dari manajer klaster Kubernetes yang mengatur kontainer rkt yang berjalan pada CoreOS. Tectonic menyediakan Distributed Trusted Computing menggunakan verifikasi kriptografi seluruh lingkungan, dari perangkat keras hingga klaster. Tectonic menyempurnakan Kubernetes sumber terbuka, dan aplikasi dapat disebarkan antara lingkungan cloud dan pusat data.

Masalah

CoreOS Linux menyediakan platform yang cocok untuk mengembangkan aplikasi yang dikontainerisasi, tetapi antarmuka baris perintah masih harus digunakan untuk menjalankan perintah Kubernetes guna membuat dan mengelola pengontrol replikasi, penyebaran, pod, atau layanan.

Solusi

Tectonic Console adalah antarmuka pengguna grafis (GUI) untuk mengelola klaster Kubernetes dari peramban web. Konsol dapat digunakan untuk menyebarkan aplikasi baru, membuat pemutakhiran bergulir untuk penyebaran, dan membuat pod, pengontrol replikasi, dan layanan. Beberapa manfaat Konsol Tektonik adalah sebagai berikut:

- Kluster Kubernetes siap pakai
- Kerangka kerja otorisasi
- Autentikasi perusahaan
- Skalabilitas yang ditingkatkan
- Dasbor yang mudah digunakan
- Pembaruan terjadwal untuk perangkat lunak kluster
- Arsitektur yang fleksibel
- Penyeimbangan Beban dan Layanan Otomatis
- Rollback
- Pemanfaatan mesin yang lebih baik
- Konsistensi lingkungan di seluruh tim
- Penyimpanan dan distribusi kredensial bawaan
- Batasan antara OS dan aplikasi
- Autentikasi aman berbasis LDAP

Ikhtisar

Konsol dapat digunakan untuk menyebarkan aplikasi baru, membuat pemutakhiran bergulir untuk penyebaran, dan membuat pod, pengontrol replikasi, dan layanan. Kami akan

membahas topik-topik berikut:

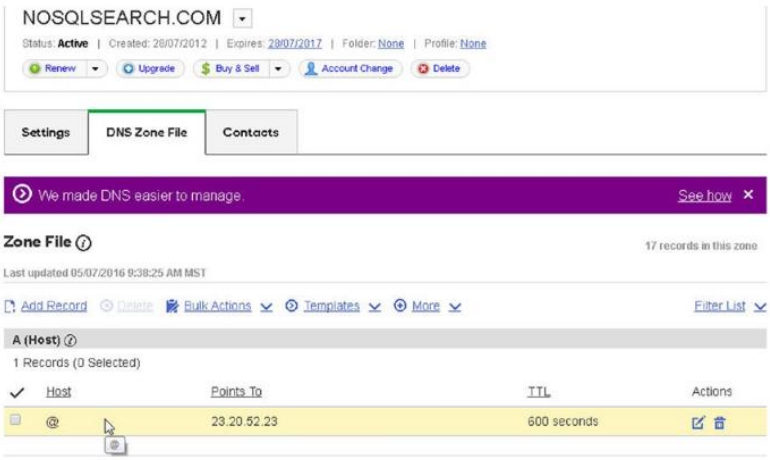
- Menetapkan lingkungan
- Mengunduh rahasia penarikan dan manifes Konsol Tektonik
- Memasang rahasia penarikan dan Konsol Tektonik
- Mengakses Konsol Tektonik
- Menggunakan Konsol Tektonik
- Menghapus Konsol Tektonik

Menetapkan Lingkungan

Sebagai prasyarat, instal kluster Kubernetes. Menginstal Kubernetes pada CoreOS pada penyedia cloud AWS dibahas dalam Bab 2. Untuk mengulanginya secara singkat, pertama-tama buat instans Amazon EC2 untuk meluncurkan AWS CloudFormation untuk kluster Kubernetes. AMI instans EC2 haruslah Amazon Linux, karena Amazon Client Interface (CLI) telah diinstal sebelumnya pada instans berbasis AMI Amazon Linux. Dapatkan alamat IP publik instans EC2 dan masuk ke instans dengan log SSH. Buat CloudFormation untuk kluster Kubernetes yang terdiri dari satu node master dan tiga node pekerja seperti yang ditunjukkan pada Gambar 5.1.

Gambar 5.1 Instans Cloudformation EC2 Untuk Kluster Kubernetes

Dapatkan alamat IP publik untuk pengontrol dari konsol EC2 dan tambahkan rekaman A untuk alamat IP tersebut ke DNS publik yang digunakan untuk menginisialisasi tumpukan CloudFormation seperti yang ditunjukkan pada Gambar 5.2.



Gambar 5.2 Menambahkan Catatan A Untuk Alamat IP Publik Pengontrol

Login SSH ke instans pengontrol dan instal kubectl seperti yang ditunjukkan pada Gambar 5.3.

```
[ec2-user@ip-172-30-1-188 ~]$ ssh -i "kubernetes-coreos.pem"  core@23.20.52.23
CoreOS stable (1010.6.0)
Update Strategy: No Reboots
core@ip-10-0-0-50 ~ $
```

Gambar 5.3 Log Masuk SSH Ke Instans Pengontrol Coreos

Jalankan perintah berikut untuk mencantumkan biner.

```
kubectl get nodes
```

Node master tunggal dan tiga node pekerja tercantum, seperti yang ditunjukkan pada Gambar 5.4.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-107.ec2.internal    Ready                      3m
ip-10-0-0-108.ec2.internal    Ready                      3m
ip-10-0-0-109.ec2.internal    Ready                      3m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   3m
core@ip-10-0-0-50 ~ $
```

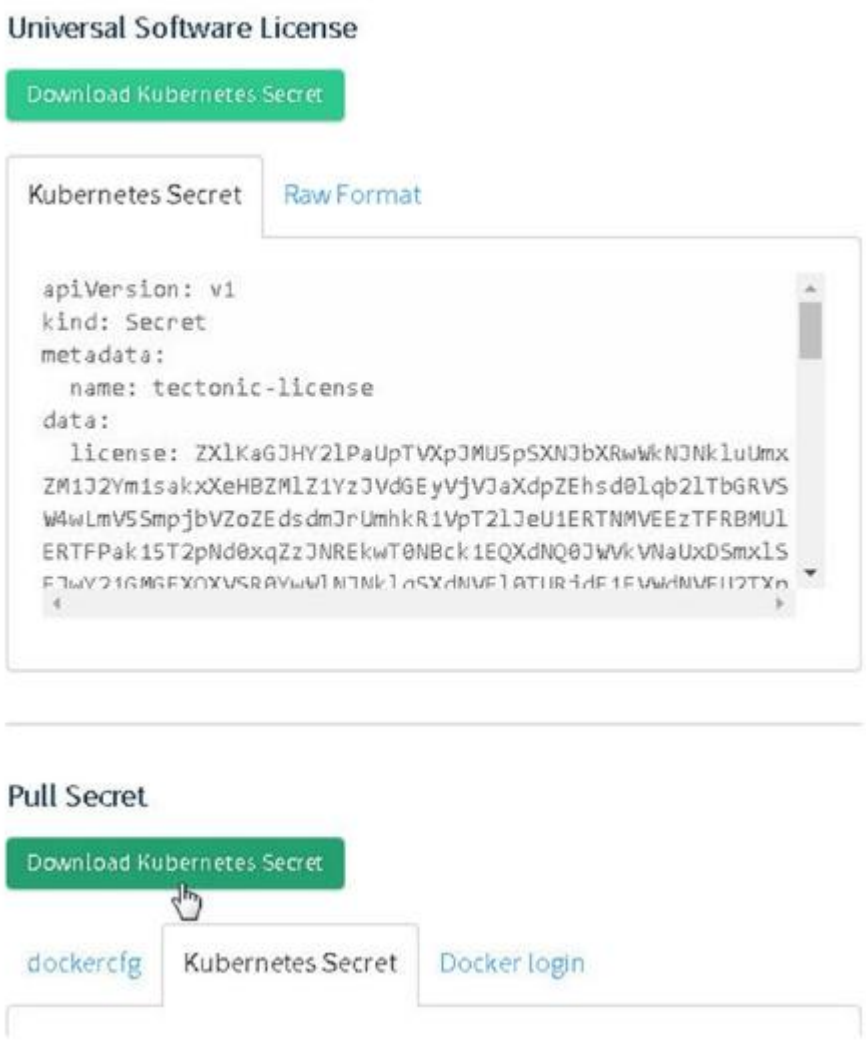
Gambar 5.4 Mencantumkan Node

Tectonic menyediakan layanan berikut untuk aplikasi yang diterapkan di kluster pengguna: Tectonic Console, Tectonic Identity, Tectonic Support, Tectonic Wizard, dan Tectonic Manager. Tectonic Console adalah konsol manajemen web untuk kluster Kubernetes. Tectonic Identity ditujukan untuk manajemen pengguna untuk layanan di kluster Kubernetes. Tectonic Support adalah dukungan dari tim. Tectonic Wizard adalah wizard instalasi dan konfigurasi Tectonic. Tectonic Manager ditujukan untuk manajemen layanan Tectonic itu sendiri. Tectonic tersedia dalam tiga tingkat langganan: Starter, Lab, dan Enterprise. Starter hanya menyertakan Tectonic Console dan tidak menyediakan fitur keamanan (SSO); layanan ini cocok sebagai tingkat pemula.

Tectonic Lab menyertakan Tectonic Console dengan Quay Enterprise Basic tetapi tidak menyertakan fitur keamanan. Tectonic Enterprise menyertakan Tectonic Identity for Trusted Computing sebagai tambahan dari Tectonic Console dengan Quay Enterprise Basic dan cocok untuk produksi. Sementara level Lab dan Enterprise berbayar, Starter gratis. Kami akan menggunakan level langganan Tectonic Starter dalam bab ini. Level Tectonic Starter tidak mengautentikasi pengguna. Selanjutnya, daftar untuk akun Tectonic Starter di https://tectonic.com/starter/.

5.2 MENGUNDUH PULL SECRET DAN TECTONIC CONSOLE MANIFEST

Tectonic terutama merupakan platform infrastruktur yang memungkinkan perusahaan untuk menjalankan kontainer dengan Kubernetes di mana saja, dengan aman dan andal. Klaster Kubernetes menggunakan Pull Secret untuk mengunduh citra Tectonic Console. Pull Secret adalah file berformat Kubernetes yang berisi kredensial yang diperlukan untuk mengunduh citra Tectonic Console. Klik Aset Akun setelah membuat proyek Tectonic Starter dan klik Unduh Kubernetes Secret untuk file Pull Secret coreos-pull-secret.yml seperti yang ditunjukkan pada Gambar 5.5.



Gambar 5.5 Mengunduh Rahasia Kubernetes

Jika file Pull Secret diunduh ke mesin lokal, scp salin file tersebut ke instans CoreOS untuk pengontrol Kubernetes. Pertama, pasangan kunci mungkin perlu disalin ke mesin lokal:

```
scp -i docker.pem ec2-user@ec2-52-201-216-175.compute-
1.amazonaws.com:~/kubernetes-coreos. pem ~/kubernetes-
coreos.pem
```

Pasangan kunci yang digunakan untuk masuk SSH ke instans pengontrol CoreOS disalin ke mesin lokal. Berikutnya salin file coreos-pull-secret.yml ke instance kontroler:

```
scp -i kubernetes-coreos.pem
/media/sf_VMShared/kubernetes/tectonic/coreos-pull-secret.yml
core@ec2-23-20-52-23.compute-1.amazonaws.com:~/coreos-pull-
secret.yml
```

File lain yang diperlukan untuk Tectonic Console adalah file Tectonic Console Manifest tectonic-console.yaml, yang mendefinisikan penyebaran Kubernetes yang diperlukan untuk menjalankan kontainer untuk Tectonic Console pada kluster Kubernetes. Unduh Tectonic Console Manifest dari https://tectonic.com/enterprise/docs/latest/deployer/files/tectonic-console.yaml. Salin Tectonic Console Manifest ke instance kontroler CoreOS.

```
scp -i kubernetes-coreos.pem
/media/sf_VMShared/kubernetes/tectonic/tectonic-console.yaml
core@ec2-23-20-52-23.compute-1.amazonaws.com:~/tectonic-
console.yaml
```

Tectonic Console Manifest disalin ke instance CoreOS kontroler.

Jika perintah `ls -l` dijalankan pada instance CoreOS untuk kontroler, file coreos-pull-secret.yml dan tectonic-console.yaml akan dicantumkan seperti yang ditunjukkan pada Gambar 5.6.

```
core@ip-10-0-0-50 ~ $ ls -l
total 55212
-rwxr-xr-x 1 core core      343 Jul  5 16:55 coreos-pull-secret.yml
-rwxr-xr-x 1 root root 56515944 Jul  1 20:06 kubectl
-rwxr-xr-x 1 core core     2121 Jul  5 16:55 tectonic-console.yaml
core@ip-10-0-0-50 ~ $
```

Gambar 5.6 Mencantumkan File Dalam Instance Coreos Kontroler

Memasang Pull Secret dan Tectonic Console Manifest

Selanjutnya, pasang Pull Secret pada kluster Kubernetes:

```
kubectl create -f coreos-pull-secret.yml
```

Kubernetes Secret yang disebut coreos-pull-secret dibuat; ini akan digunakan oleh Kubernetes untuk menarik dan memasang citra untuk Tectonic Console. Selanjutnya, pasang Tectonic Console menggunakan Tectonic Console Manifest, dengan menggunakan Pull Secret untuk menarik dan memasang citra untuk tectonic-console. Perintah berikut membuat pengontrol

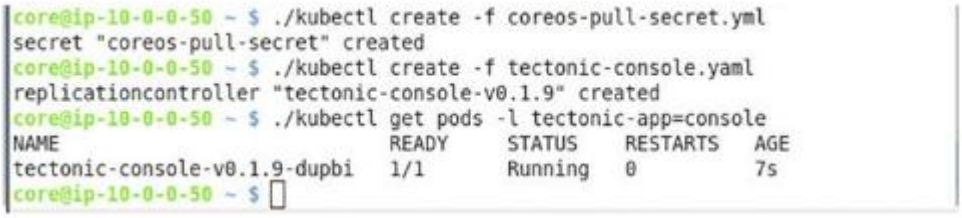
replikasi yang disebut tectonic-console.

```
kubectl create -f tectonic-console.yaml
```

Daftar pod, yang seharusnya menentukan hanya pod tectonic-console yang akan dicantumkan:

```
kubectl get pods -l tectonic-app=console
```

Jika Tectonic Console telah diinstal, output yang mirip dengan Gambar 5.7 akan dihasilkan dari perintah sebelumnya.



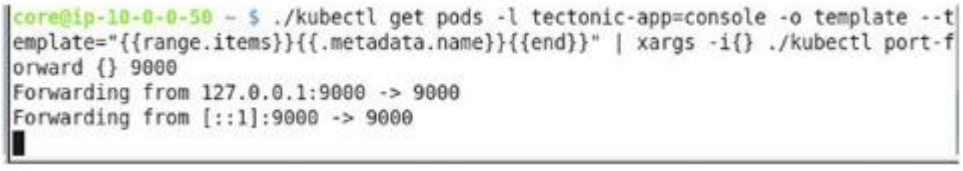
Gambar 5.7 Membuat Pengontrol Replikasi Dan Pod Untuk Tectonic Console

Mengakses Tectonic Console

Karena Tectonic Starter tidak mengautentikasi pengguna, antarmuka tidak terekspos di luar klaster, dan penerusan port harus ditetapkan dari mesin pengontrol ke port layanan Tectonic Console 9000. Perintah berikut menyiapkan penerusan port dari 127.0.0.1:9000 ke port 9000 pada pod berlabel app=tectonic-console:

```
kubectl get pods -l tectonic-app=console -o template --
template="{{range.items}}{{.metadata. name}}{{end}}" | xargs -
i{} kubectl port-forward {} 9000
```

Penerusan port dari mesin tempat perintah sebelumnya dijalankan, yang merupakan instans pengontrol, ke pod tempat kontainer untuk Konsol Tektonik dijalankan, akan disiapkan seperti yang ditunjukkan pada Gambar 5.8.



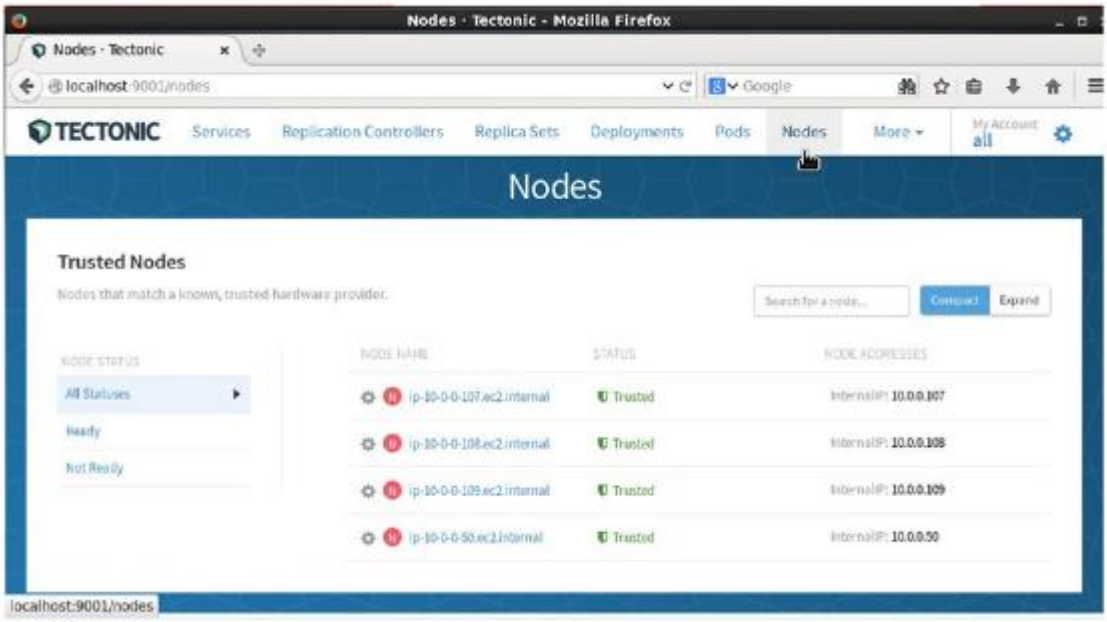
Gambar 5.8 Menetapkan Penerusan Port

Untuk memanggil Konsol Tektonik di peramban web, kita masih perlu menetapkan penerusan port lain dari mesin lokal ke mesin pengontrol, yang memiliki IP publik 23.20.52.23 dan DNS

publik ec2-23-20-52-23. compute-1.amazonaws.com. Port selain 9000 dapat digunakan pada mesin lokal untuk meneruskan ke port Konsol Tektonik. Perintahnya terlihat seperti ini:

```
ssh -i kubernetes-coreos.pem -f -nNT -L 9001:127.0.0.1:9000
core@ec2-23-20-52-23.compute-1. amazonaws.com
```

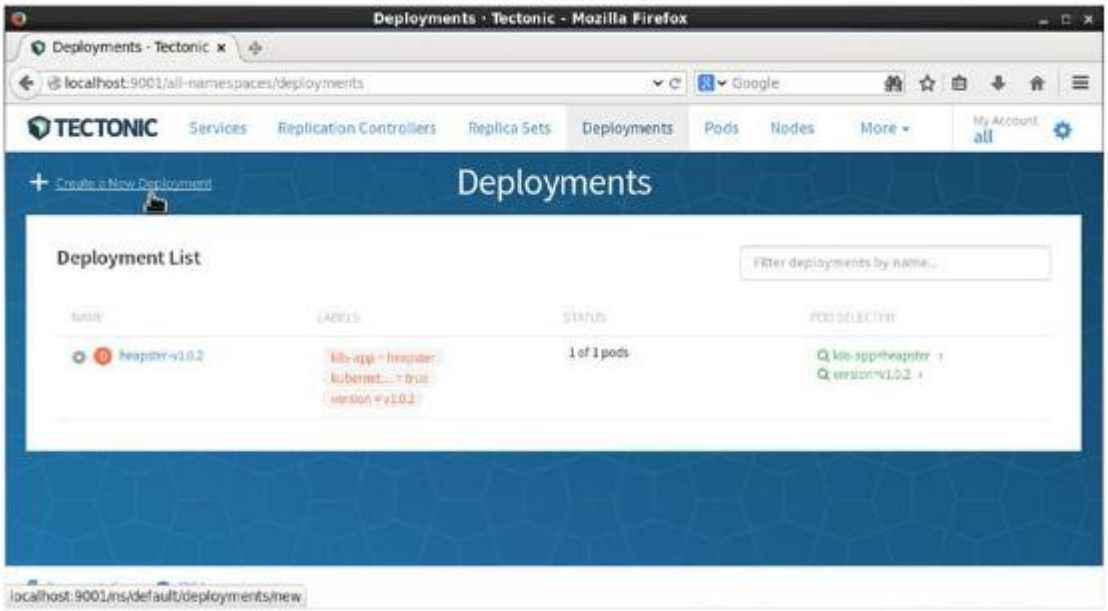
Penerusan port dari mesin lokal tempat perintah sebelumnya dijalankan ke instans pengontrol telah disiapkan. Akses Tectonic Console di URL http://localhost:9001 di browser pada mesin lokal seperti yang ditunjukkan pada Gambar 5.9. Port dapat berbeda jika port localhost yang berbeda diteruskan.



Gambar 5.9 Mengakses Tectonic Console

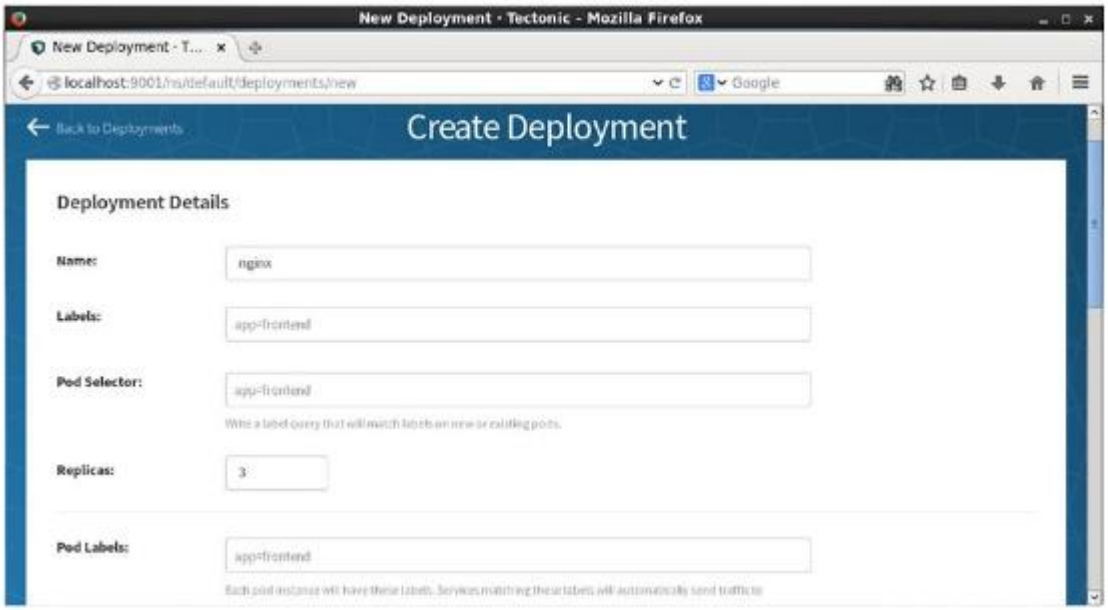
5.3 MENGGUNAKAN TECTONIC CONSOLE

Tectonic Console dapat digunakan untuk melihat berbagai objek Kubernetes, seperti deployment, replication controller, replica set, pod, dan layanan, atau untuk membuat objek Kubernetes baru. Untuk menampilkan deployment, klik tab Deployments. Untuk membuat deployment baru, klik tautan Create a New Deployment seperti yang ditunjukkan pada Gambar 5.10.



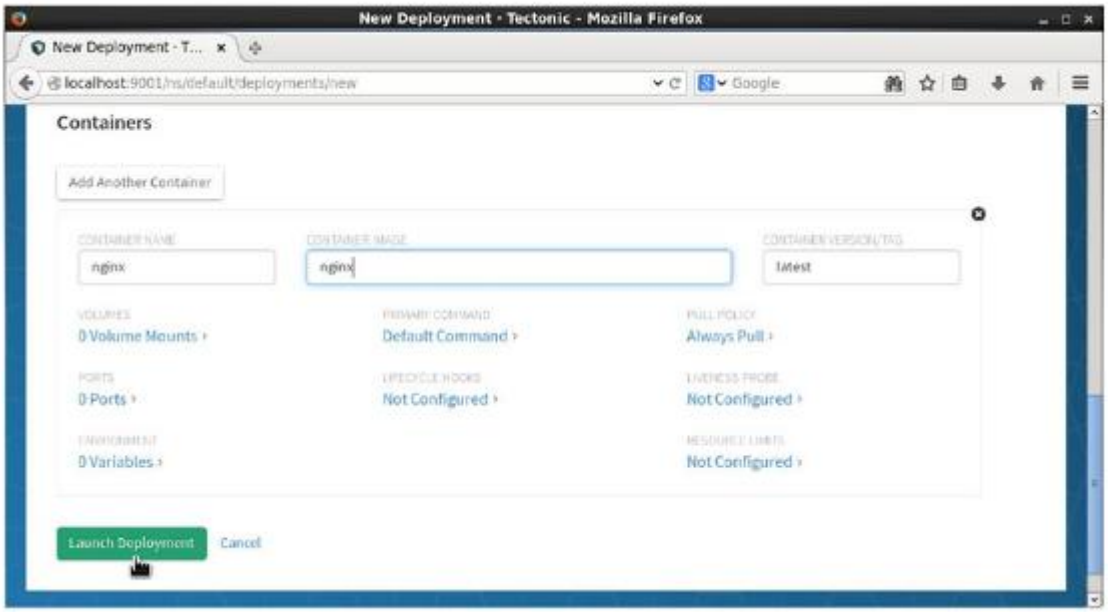
Gambar 5.10 Mulailah Dengan Mengeklik Tautan Buat Penerapan Baru

Sebagai contoh, buat penerapan untuk server nginx dengan menentukan label, pemilih pod, replika, dan label pod seperti yang ditunjukkan pada Gambar 5.11.



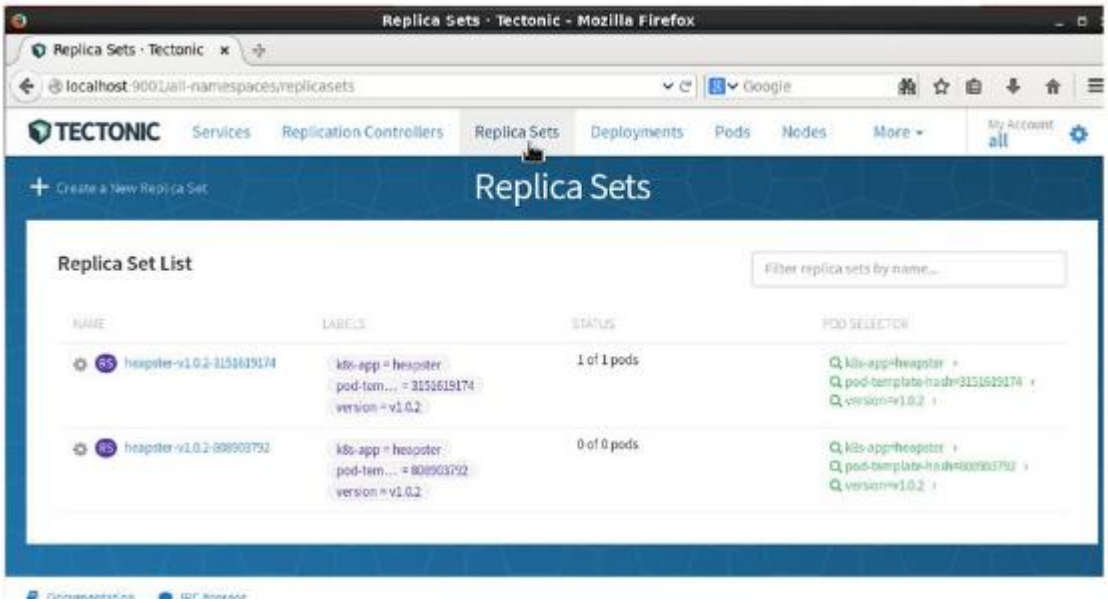
Gambar 5.11 Menentukan Detail Penerapan Untuk Aplikasi Nginx

Tentukan nama kontainer, gambar, dan versi/tag, lalu klik Luncurkan Penerapan seperti yang ditunjukkan pada Gambar 5.12.



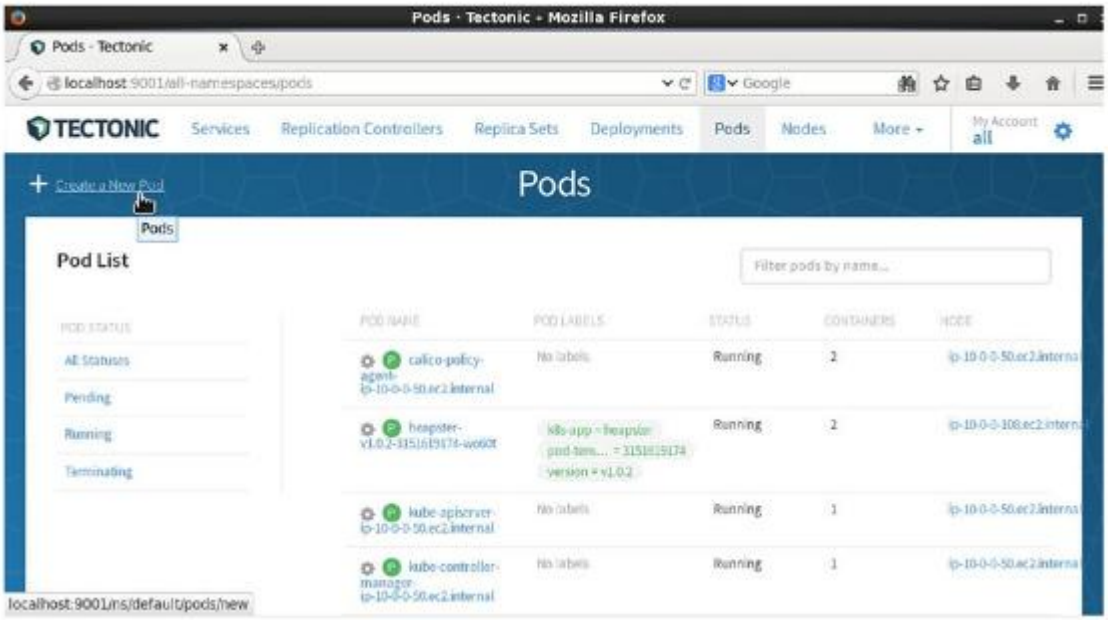
Gambar 5.12 Meluncurkan penerapan

Anda dapat mencantumkan kumpulan replika menggunakan tab Kumpulan Replika, seperti yang ditunjukkan pada Gambar 5.13.



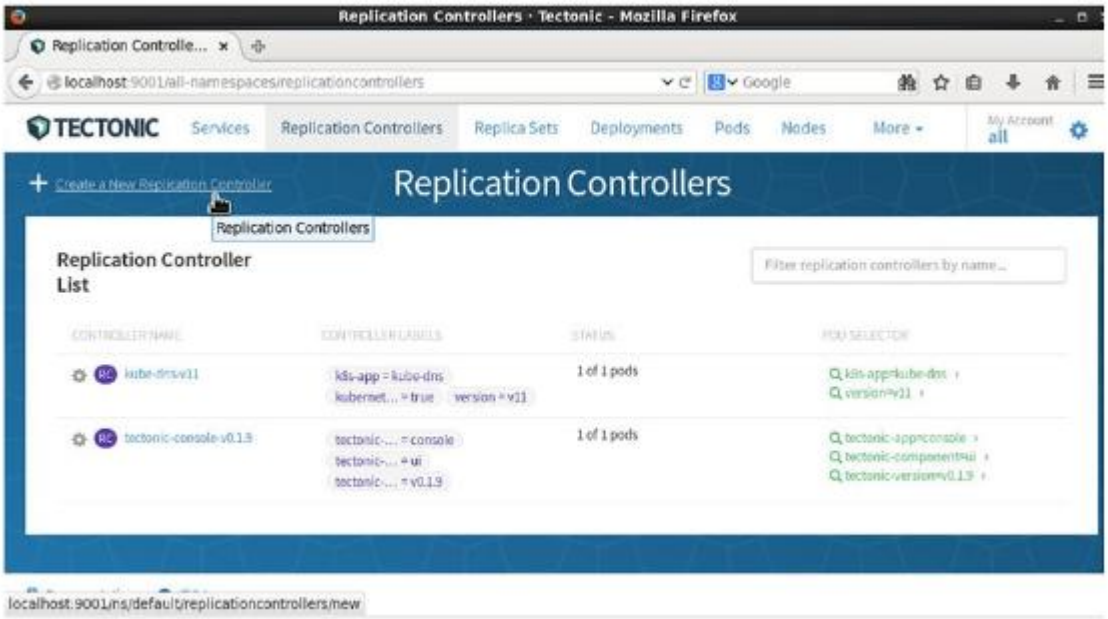
Gambar 5.13 Mencantumkan Kumpulan Replica

Untuk membuat daftar pod, klik tab Pod. Untuk membuat pod baru, klik tautan Buat Pod Baru seperti yang ditunjukkan pada Gambar 5.14.



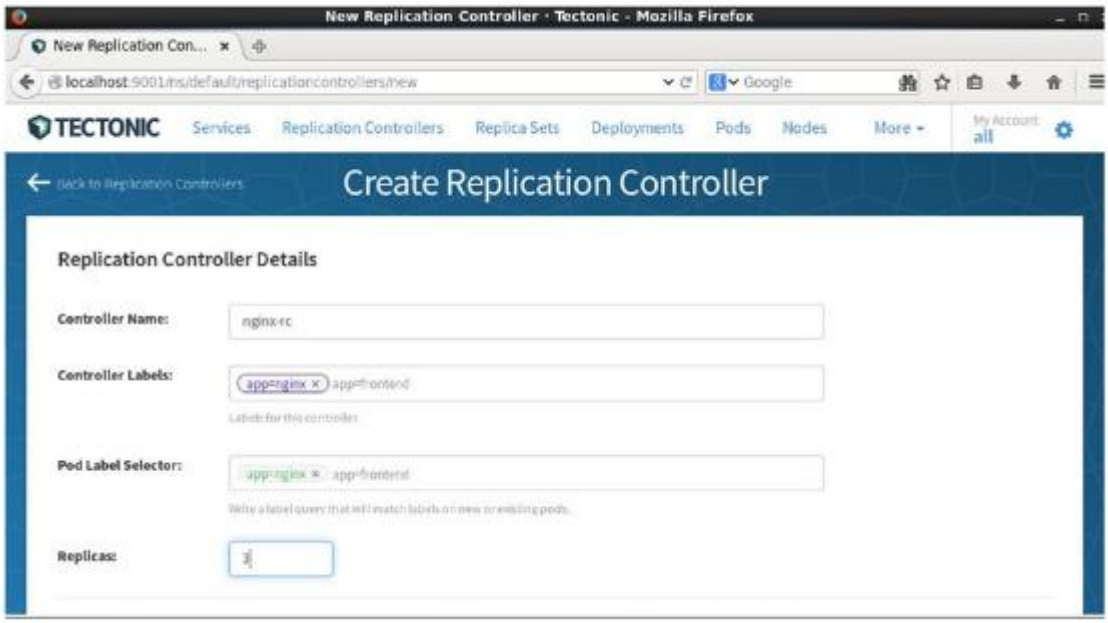
Gambar 5.14 Mulailah Dengan Mengklik Tautan Buat Pod Baru

Untuk membuat daftar pengontrol replikasi, klik tab Pengontrol Replikasi. Untuk membuat pengontrol replikasi baru, klik tautan Buat Pengontrol Replikasi Baru seperti yang ditunjukkan pada Gambar 5.15.



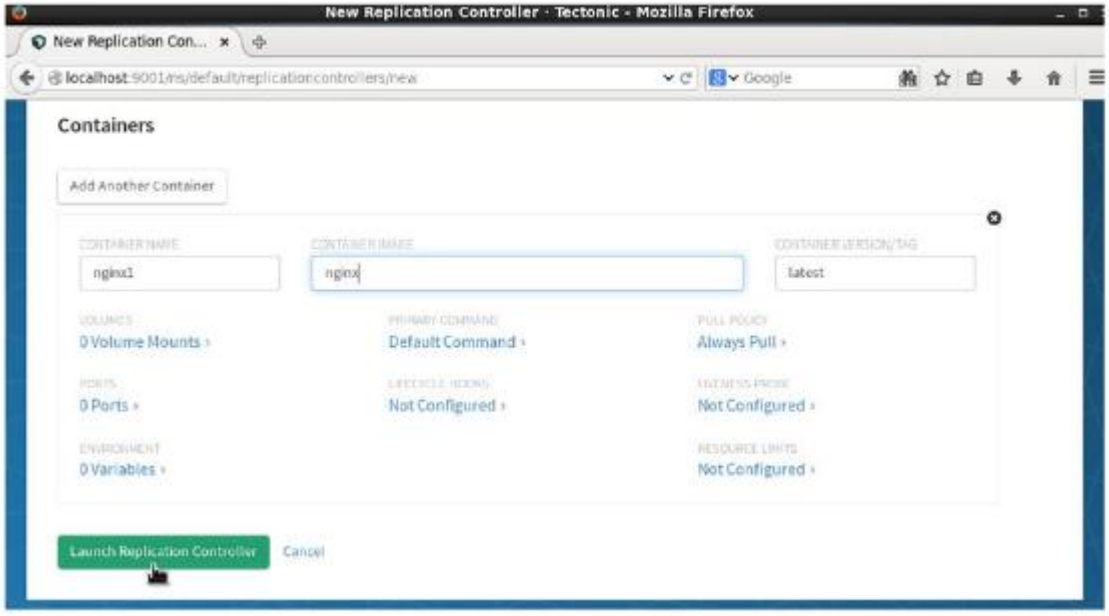
Gambar 5.15 Mulailah Dengan Mengklik Tautan Buat Pengontrol Replikasi Baru

Tentukan nama pengontrol, label pengontrol, pemilih label pod, dan replika seperti yang ditunjukkan pada Gambar 5.16.



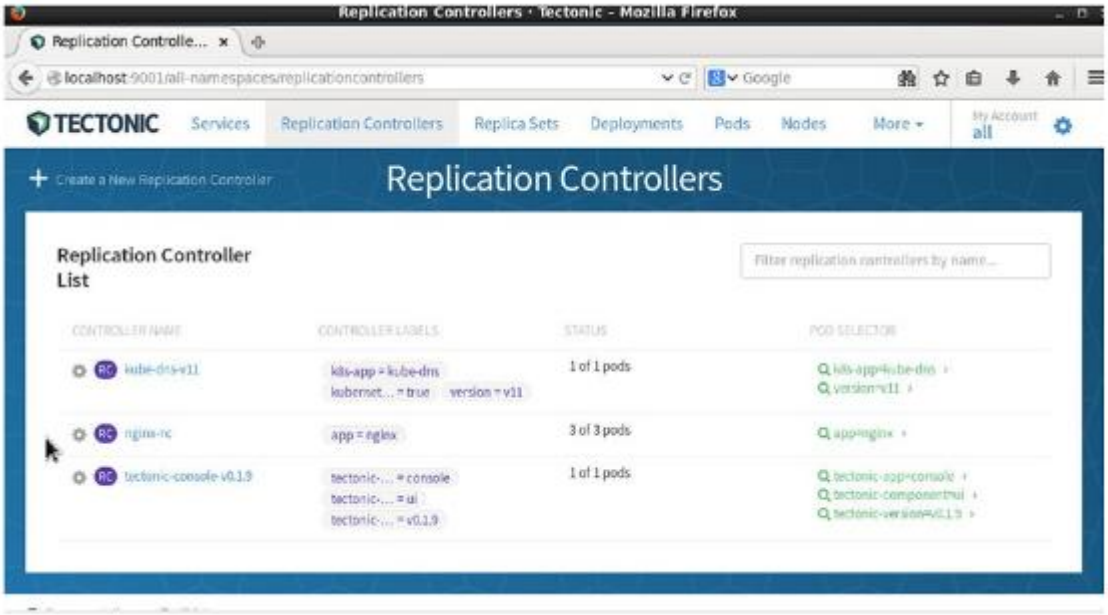
Gambar 5.16 Menentukan Detail Pengontrol Replikasi

Tentukan nama kontainer, gambar kontainer, dan versi/tag kontainer. Klik Luncurkan Pengontrol Replikasi seperti yang ditunjukkan pada Gambar 5.17.



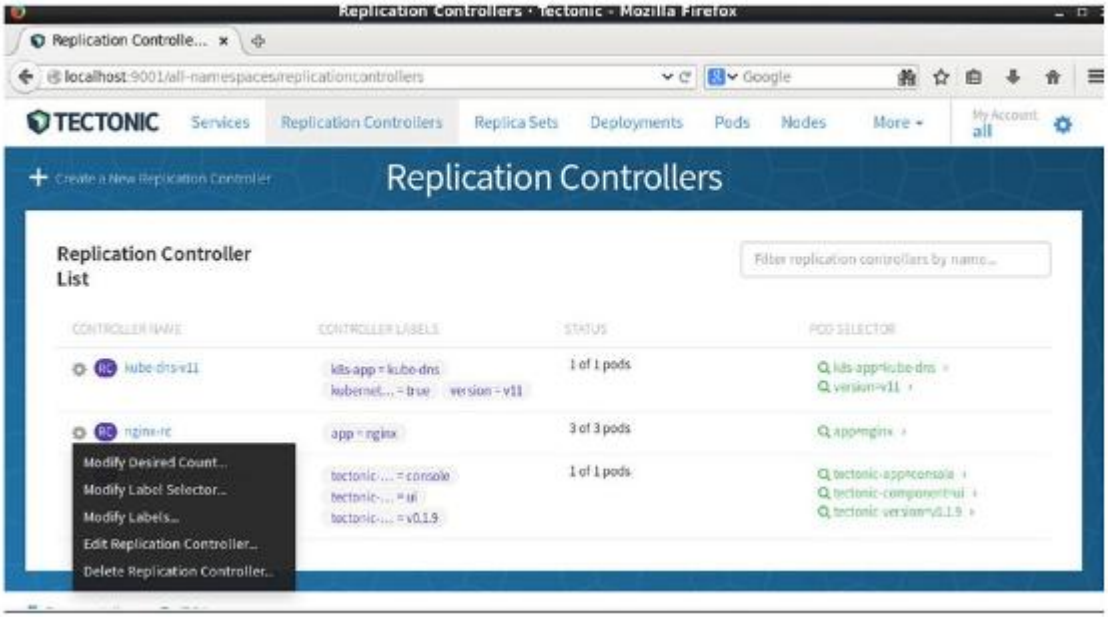
Gambar 5.17 Meluncurkan Pengontrol Replikasi

Pengontrol replikasi baru dibuat, seperti yang ditunjukkan pada Gambar 5.18.



Gambar 5.18 Pengontrol Replikasi Baru

Untuk mengubah pengaturan pengontrol replikasi, klik kanan RC dan pilih salah satu opsi yang ditunjukkan pada Gambar 5.19.

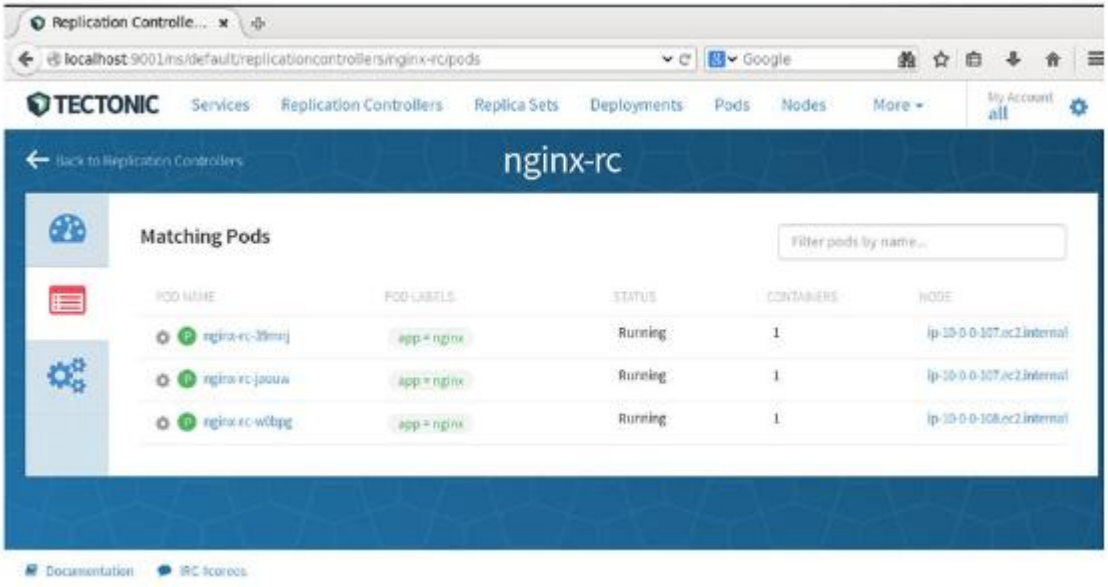


Gambar 5.19 Mengubah Atau Menghapus RC

Klik RC untuk mencantumkan detailnya. Anda dapat mencantumkan pod di RC dengan menggunakan tab Pod, seperti yang ditunjukkan pada Gambar 5.20.

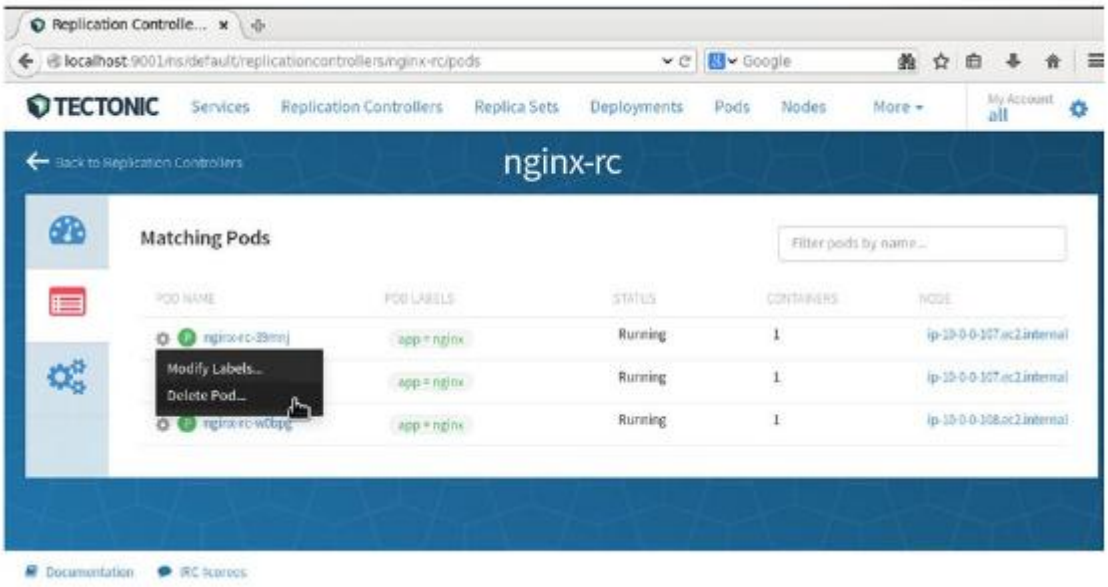
Gambar 5.20 Mencantumkan Pod Di RC

Pod yang dikelola oleh RC tercantum seperti yang ditunjukkan pada Gambar 5.21.



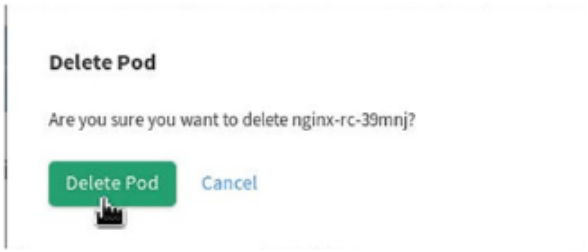
Gambar 5.21 Pod Di RC Nginx-Rc

Untuk mengubah label pod atau menghapusnya, klik kanan pod dan pilih salah satu opsi yang ditunjukkan pada Gambar 5.22. Klik Delete Pod untuk menghapus pod.



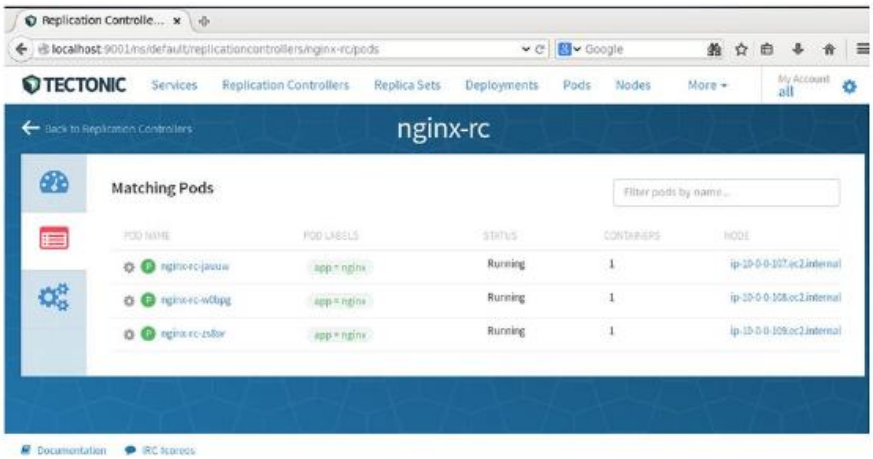
Gambar 5.22 Mengubah Label Pod Atau Menghapus Pod

Pada dialog konfirmasi, klik Delete Pod seperti yang ditunjukkan pada Gambar 5.23.



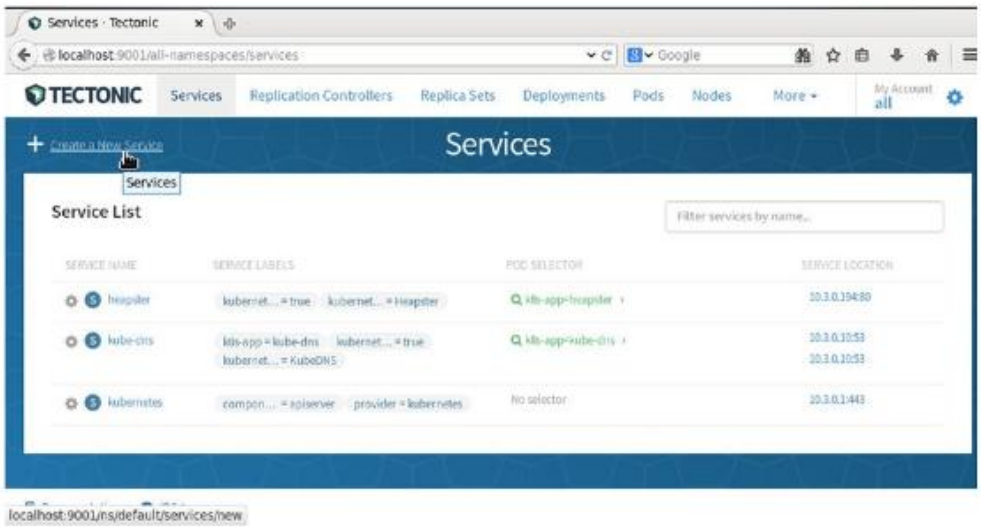
Gambar 5.23 Menghapus pod

Jika jumlah pod yang berjalan bukan jumlah replika yang ditentukan untuk RC, pod baru diluncurkan untuk pod yang dihapus, dan jumlah pod kembali menjadi 3, seperti yang ditunjukkan pada Gambar 5.24.



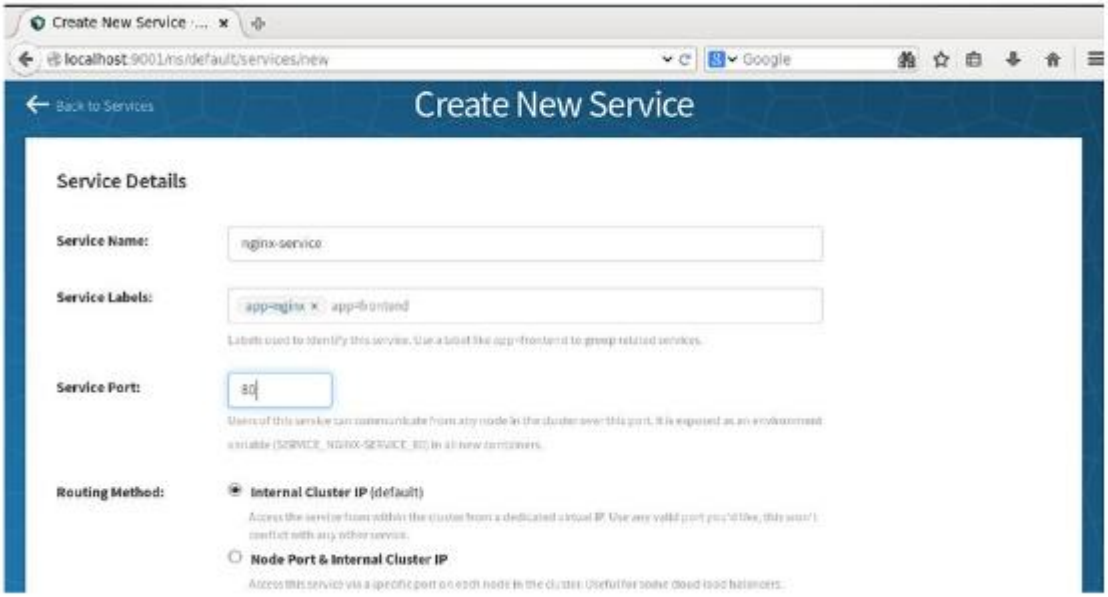
Gambar 5.24 Pod Yang Diluncurkan Kembali

Untuk mencantumkan layanan, klik tab Layanan. Untuk membuat layanan baru, klik tautan Buat Layanan Baru seperti yang ditunjukkan pada Gambar 5.25.



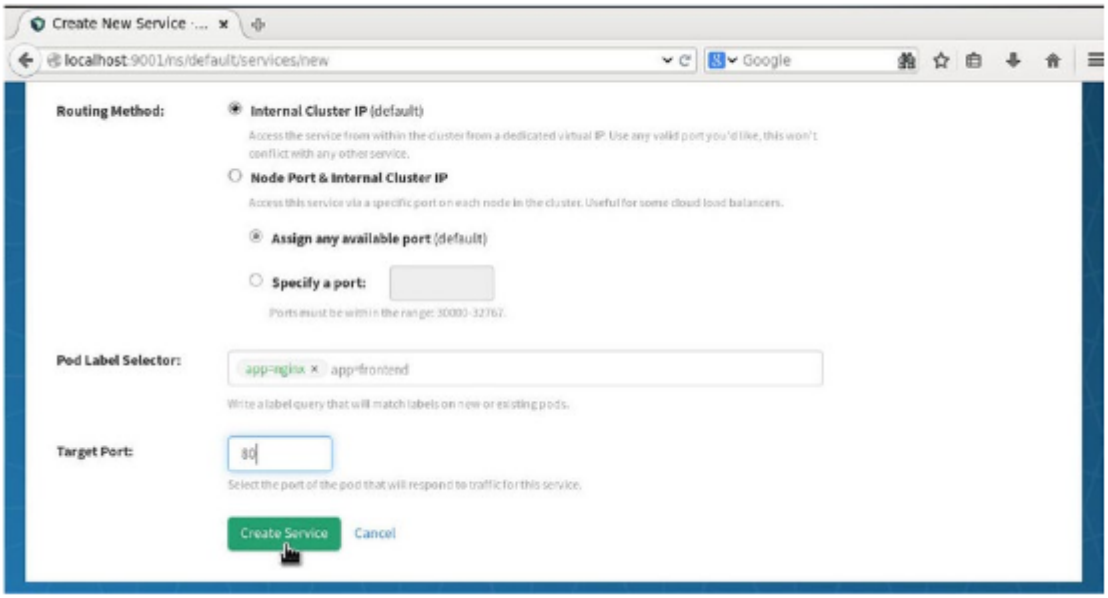
Gambar 5.25 Mulailah Dengan Mengklik Tautan Buat Layanan Baru

Tentukan detail layanan seperti nama layanan, label, port, dan metode perutean dalam formulir Buat Layanan Baru yang ditunjukkan pada Gambar 5.26.



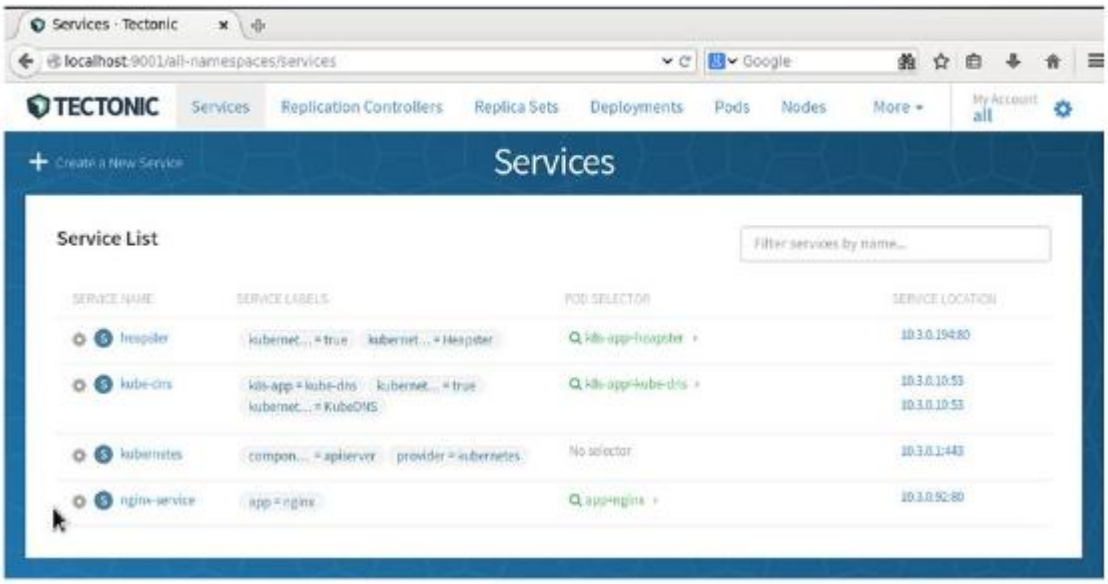
Gambar 5.26 Menentukan Detail Untuk Layanan Baru

Pilih Tetapkan Port yang Tersedia (default) untuk menetapkan port apa pun. Tentukan Pemilih Label Pod dan Port Target, lalu klik Buat Layanan seperti yang ditunjukkan pada Gambar 5.27.



Gambar 5.27 Membuat Layanan Baru

Layanan baru dibuat, seperti yang ditunjukkan pada Gambar 5.28.



Gambar 5.28 Layanan baru dibuat di Tectonic Console

Menghapus Tectonic Console

Untuk menghapus Tectonic Console, jalankan perintah berikut:

```
kubectl delete replicationcontrollers tectonic-console
```

Untuk menghapus Kubernetes Pull Secret, jalankan perintah berikut:

```
kubectl delete secrets coreos-pull-secret
```

Ringkasan

Dalam bab ini, kami memasang GUI Tectonic Console, bagian dari versi Starter gratis dari platform Kubernetes perusahaan komersial Tectonic. Kami mengakses konsol di browser dan membuat contoh pengontrol replikasi dan layanan. Terakhir, kami menghapus Tectonic Console. Dalam bab berikutnya, kami akan membahas penggunaan volume Kubernetes.

BAB 6
MENGGUNAKAN VOLUME

6.1 MENGELOLA DATA DAN VOLUME DALAM KUBERNETES PODS

Pod Kubernetes selalu dikaitkan dengan data, dan data dapat dijadikan bagian integral dari kontainer Docker melalui citra Docker-nya atau dipisahkan dari kontainer Docker.

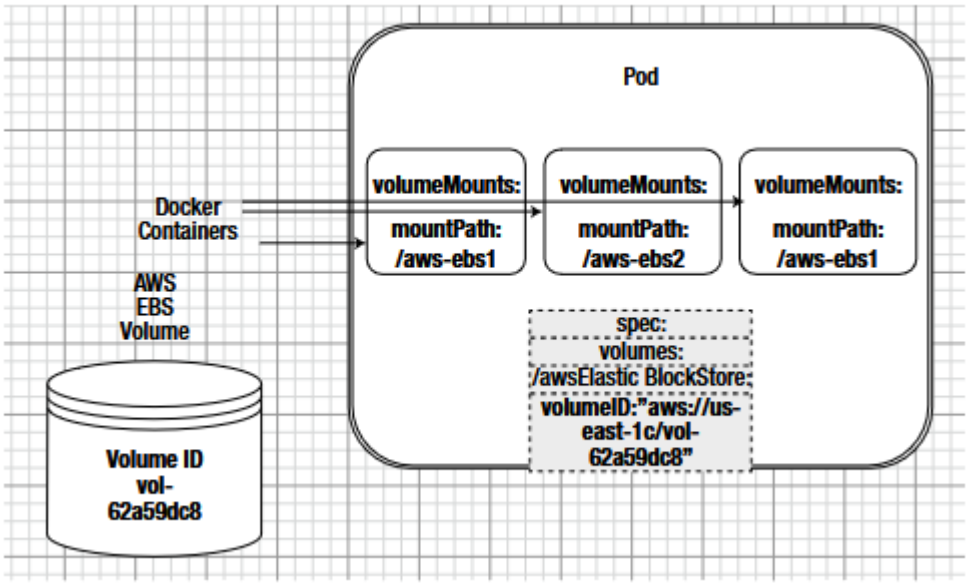
Masalah

Jika data (dalam file pada disk) dijadikan bagian integral dari kontainer Docker, masalah berikut dapat terjadi:

- Data tidak persisten. Data dihapus saat kontainer Docker dimulai ulang, yang juga dapat disebabkan oleh kerusakan kontainer.
- Data bersifat khusus kontainer dan tidak dapat dibagikan dengan kontainer lain.

Solusi

Salah satu prinsip desain modular adalah Prinsip Tanggung Jawab Tunggal (SRP). Volume Kubernetes menerapkan SRP dengan memisahkan data dari kontainer. Volume hanyalah direktori dengan atau tanpa data pada beberapa media, yang berbeda untuk jenis volume yang berbeda. Volume ditentukan dalam spesifikasi pod dan dibagikan ke berbagai kontainer di pod. Volume harus dipasang di setiap kontainer dalam spesifikasi pod secara independen, meskipun volume dapat dipasang di kontainer yang berbeda pada jalur file/direktori yang sama (atau berbeda). Kontainer dalam pod memiliki akses ke sistem file yang diwarisi dari citra Docker-nya dan sistem file dari volume Kubernetes.



Gambar 6.1 Menggunakan Volume Dalam Pod

Sistem file citra Docker masih berada di akar hierarki sistem file, dan volume hanya dapat dipasang pada jalur direktori dalam sistem file akar. Karena volume menyediakan akses

ke data di luar pod, volume yang dipasang di pod yang berbeda dapat berbagi data yang sama dari host atau penyimpanan eksternal lainnya seperti AWS EBS atau repositori GitHub. Tersedia dua jenis abstraksi atau plugin volume: Volume dan PersistentVolume. Sementara Volume digabungkan dengan pod, PersistentVolume disediakan pada kluster jaringan dan independen dari pod. Gambar 6.1 menunjukkan contoh penggunaan Volume Amazon EBS dalam pod.

Ikhtisar

Volume Kubernetes adalah unit penyimpanan yang dikaitkan dengan pod. Volume dapat dibagikan oleh replika pod atau didedikasikan untuk satu pod. Beberapa jenis volume didukung untuk berbagai jenis penyimpanan, seperti volume AWS, repositori GitHub, atau direktori pada host, dan masih banyak lagi. Berbagai jenis volume dijelaskan dalam Tabel 6.1.

Tabel 6.1 Jenis-jenis Volume

Jenis Volume	Keterangan
direktorikosong	Volume per pod yang awalnya kosong dan dibagikan oleh kontainer dalam pod. Setiap kontainer dapat memasang volume di jalur yang sama atau berbeda. Secara default, volume disimpan pada media yang mendukung mesin, yang dapat berupa SSD atau penyimpanan jaringan. Atau, media dapat diatur ke memori. Saat pod dihapus, volume juga dihapus, yang berarti volume tidak persisten.
jalurhost	Memasang berkas atau direktori dari sistem berkas node host ke dalam pod. Hanya dapat ditulis oleh root, data volume tetap ada meskipun pod dihapus. Semua kontainer di pod dapat mengakses volume. Dirancang hanya untuk pengujian node tunggal dan didukung dalam kluster multi-node.
gcePersistentDisk	Memasang Google Compute Engine Persistent Disk ke dalam pod. Konten GCE PD tidak dihapus jika pod dihapus dan volume dilepas. Hanya didukung untuk node jenis GCE VM dalam proyek GCE yang sama.
awsElasticBlockStore	Memasang volume AWS EBS ke dalam pod. Volume bersifat persisten, karena isinya tidak dihapus saat pod dihapus; volume tidak dipasang. Node tempat pod berjalan harus berupa instans Amazon EC2 di wilayah dan zona ketersediaan yang sama dengan volume EBS. Satu instans dapat memasang volume EBS.
Nfs	Volume persisten; memasang Sistem Berkas Jaringan (NFS) ke dalam sebuah pod.
flocker	Memasang kumpulan data Flocker ke dalam pod. Flocker adalah pengelola volume data kontainer berkluster sumber terbuka.
gitRepo	Mengkloning repositori Git ke dalam direktori kosong.

persistentVolumeClaim	Memasang PersistentVolume ke dalam pod.
azureFileVolume	Pemasangan azureFileVolume

Bab ini membahas topik-topik berikut:

- Menetapkan Lingkungan
- Membuat Volume AWS
- Menggunakan Volume awsElasticBlockStore
- Membuat Repo Git
- Menggunakan Volume gitRepo

Menetapkan Lingkungan

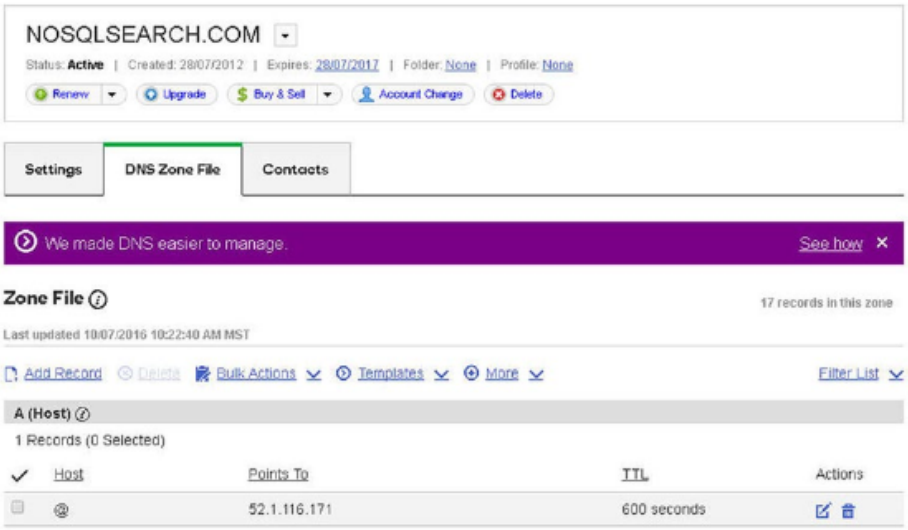
Kita akan membuat AWS CloudFormation di CoreOS untuk kluster Kubernetes. Untuk memulai, buat satu instans EC2 dari AMI Amazon Linux. Login SSH ke instans EC2 menggunakan alamat IP publik:

```
ssh -i "docker.pem" ec2-user@54.173.38.246
```

Buat CloudFormation yang terdiri dari satu pengontrol dan tiga node pekerja seperti yang ditunjukkan pada Gambar 6.2.

Gambar 6.2 Instans CloudFormation EC2

Dapatkan IP publik pengontrol kluster Kubernetes dan tambahkan rekaman A untuknya ke DNS publik tempat CloudFormation dibuat, seperti yang ditunjukkan pada Gambar 6.3.



Gambar 6.3 Menambahkan Rekaman A Untuk Domain

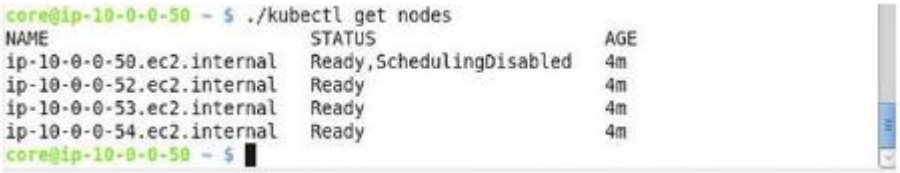
Login SSH ke instans pengontrol menggunakan IP publik:

```
ssh -i "kubernetes-coreos.pem" core@52.1.116.171
```

Instal biner kubectl dan daftarkan node dalam kluster:

```
./kubectl get nodes
```

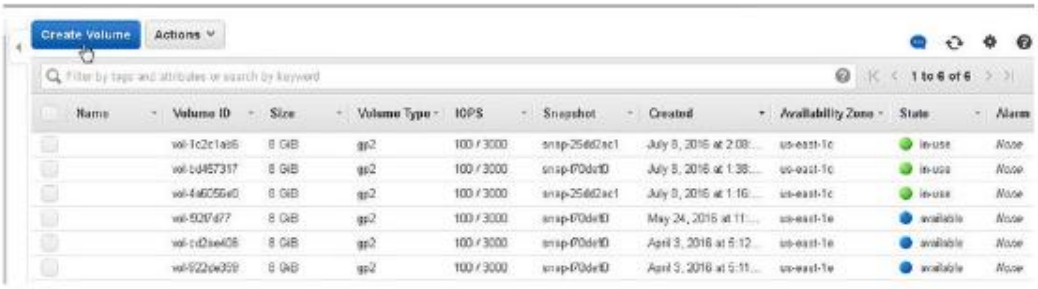
Node pengontrol tunggal dan tiga node pekerja dicantumkan seperti yang ditunjukkan pada Gambar 6.4.



Gambar 6.4 Mencantumkan Node Yang Sedang Berjalan

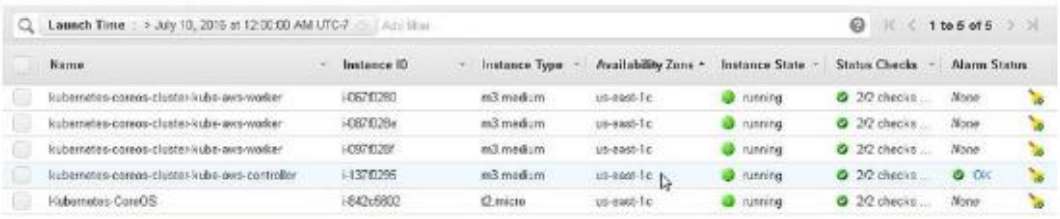
6.2 MEMBUAT VOLUME AWS

Volume awsElasticBlockStore memasang volume AWS EBS ke dalam pod. Di bagian ini, kita akan membuat volume AWS EBS. Klik Buat Volume di Konsol EC2 seperti yang ditunjukkan pada Gambar 6.5.



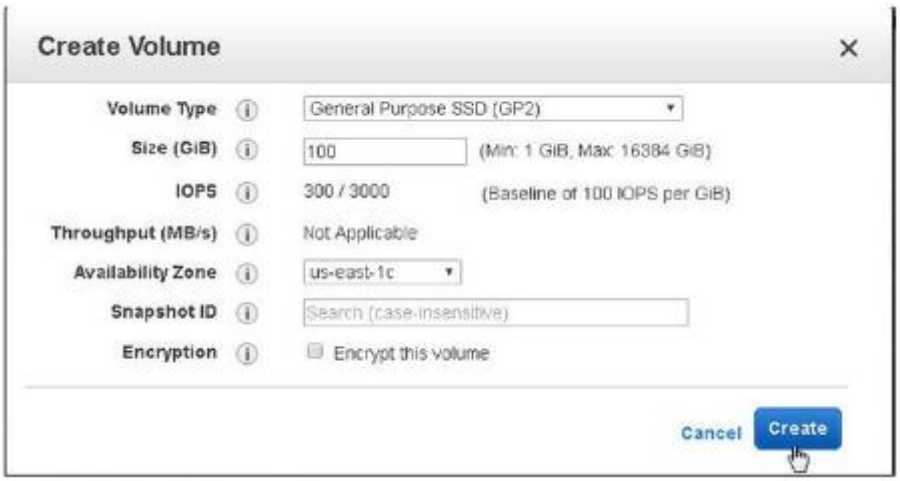
Gambar 6.5 Mulailah Dengan Mengklik Buat Volume

Volume EBS harus dibuat di zona ketersediaan yang sama dengan instans EC2 tempat pod akan memasang volume EBS. Zona ketersediaan diperoleh dari konsol EC2 dan adalah us-east-1c, seperti yang ditunjukkan pada Gambar 6.6.



Gambar 6.6 Memperoleh Zona Ketersediaan

Dalam dialog Buat Volume, tetapkan Jenis Volume sebagai SSD Tujuan Umum (GP2) dan Ukuran sebagai 100 GiB. Tetapkan Zona Ketersediaan sebagai us-east-1c dan klik Buat seperti yang ditunjukkan pada Gambar 6.7.



Gambar 6.7 Membuat Volume

Volume AWS EBS dibuat. Metode alternatif untuk membuat volume EBS adalah dengan perintah aws ec2 create-volume, sebagai berikut. Zona ketersediaan ditentukan dengan parameter perintah --availability-zone sebagai us-east-1c, sama seperti instans EC2 tempat

pod berjalan.

```
aws ec2 create-volume --availability-zone us-east-1c --size 10
--volume-type gp2
```

Volume AWS EBS dibuat, seperti yang ditunjukkan pada Gambar 6.8.

```
[ec2-user@ip-10-0-0-126 ~]$ aws ec2 create-volume --availability-zone us-east-1c
 --size 10 --volume-type gp2
{
    "AvailabilityZone": "us-east-1c",
    "Encrypted": false,
    "VolumeType": "gp2",
    "VolumeId": "vol-529fa7f8",
    "State": "creating",
    "Iops": 100,
    "SnapshotId": "",
    "CreateTime": "2016-07-10T18:04:31.554Z",
    "Size": 10
}
[ec2-user@ip-10-0-0-126 ~]$
```

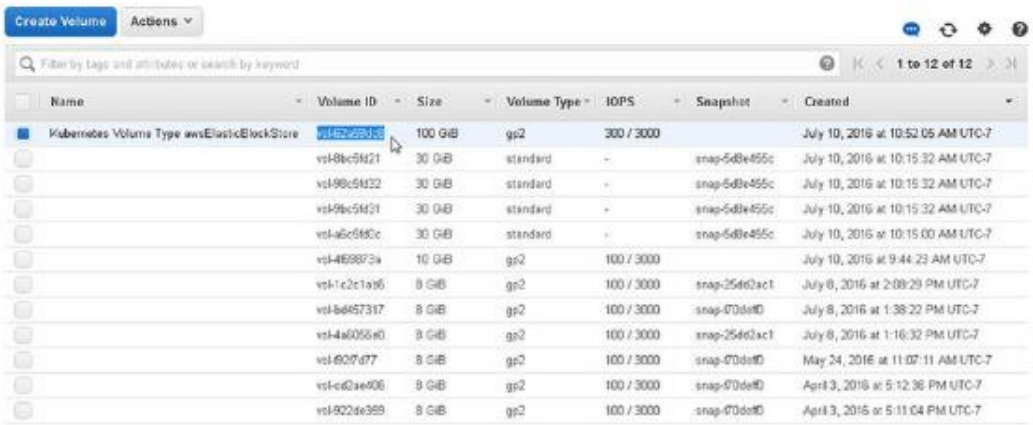
Gambar 6.8 Membuat Volume Pada Baris Perintah

6.3 MENGGUNAKAN VOLUME AWSELASTICBLOCKSTORE

Selanjutnya, kita akan menggunakan volume EBS dalam file spesifikasi ReplicationController. Buat file bernama `pod-aws.yaml`:

```
sudo vi pod-aws.yaml
```

Tentukan volume awsElasticBlockStore dengan kunci volumes dengan format volumeID yang ditetapkan ke aws://zone/volumeid. Dapatkan volumeID dari konsol EC2 seperti yang ditunjukkan pada Gambar 6.9.



Gambar 6.9 Memperoleh Volumeid

Tentukan `fsType` sebagai `ext4` dan nama volume sebagai `aws-volume`:

```
volumes:
  - awsElasticBlockStore:
      fsType: ext4
      volumeID: "aws://us-east-1c/vol-62a59dc8"
    name: aws-volume
```

Definisi volume sebelumnya dipasang dengan kunci volumeMounts. File pod-aws.yaml tampak seperti ini:

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: nginx
  name: nginx-rc
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      containers:
      - image: nginx
        name: nginx
        volumeMounts:
        - mountPath: /aws-ebs
          name: aws-volume
      volumes:
      - awsElasticBlockStore:
          fsType: ext4
          volumeID: "aws://us-east-
```

Simpan file pod-aws.yaml dengan :wq seperti yang ditunjukkan pada Gambar 6-10.

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: nginx
  name: nginx-rc
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        -
          image: nginx
          name: nginx
          volumeMounts:
            -
              mountPath: /aws-ebs
              name: aws-volume
      volumes:
        -
          awsElasticBlockStore:
            fsType: ext4
            volumeID: "aws://us-east-1c/vol-62a59dc8"
          name: aws-volume

~
~
:wq
```

Gambar 6.10 Pod-Aws.Yaml

Buat pengontrol replikasi dengan perintah kubectl create:

```
./kubectl create -f pod-aws.yaml
```

Daftar deployment, pengontrol replikasi, dan pod:

```
./kubectl get deployments
./kubectl get rc
./kubectl get pods
```

Pengontrol replikasi dibuat. Pod mungkin awalnya terdaftar sebagai belum siap, seperti yang ditunjukkan pada Gambar 6.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-aws.yaml
replicationcontroller "nginx-rc" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME       DESIRED   CURRENT   AGE
nginx-rc   1         1         14s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME             READY     STATUS              RESTARTS   AGE
nginx-rc-a3mih   0/1       ContainerCreating   0          28s
server           1/1       Running             0          31m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME             READY     STATUS              RESTARTS   AGE
nginx-rc-a3mih   0/1       ContainerCreating   0          32s
server           1/1       Running             0          31m
```

Gambar 6.11. Pod Dibuat Tetapi Belum Berjalan Dan Siap

Dengan menggunakan nama pod, jelaskan pod tersebut dengan perintah berikut:

```
kubectl describe pod nginx-rc-a3mih
```

Deskripsi pod tercantum. Volume AWSElasticBlockStore juga harus tercantum seperti yang ditunjukkan pada Gambar 6.12.

```
Volumes:
  aws-volume:
    Type:       AWSElasticBlockStore (a Persistent Disk resource in AWS)
    VolumeID:   aws://us-east-1c/vol-62a59dc8
    FSType:     ext4
    Partition:  0
    ReadOnly:   false
  default-token-nx5cy:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-nx5cy
QoS Tier:       BestEffort
```

Gambar 6.12 Deskripsi Volume

Cantumkan pod lagi setelah jeda (hingga satu menit), dan pod akan berjalan dan siap seperti yang ditunjukkan pada Gambar 6.13.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
nginx-rc-a3mih  1/1       Running   0          2m
server          1/1       Running   0          32m
core@ip-10-0-0-50 ~ $
```

Gambar 6.13 Pod Berjalan Dan Siap

Dengan menggunakan perintah kubectl exec, mulai shell bash interaktif. Cantumkan file dan direktori dengan `ls –l`, dan direktori aws-ebs akan tercantum; ini adalah jalur pemasangan untuk volume seperti yang ditunjukkan pada Gambar 6.14.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
nginx-rc-a3mih  1/1       Running   0          2m
server          1/1       Running   0          32m
core@ip-10-0-0-50 ~ $
core@ip-10-0-0-50 ~ $ ./kubectl exec nginx-rc-a3mih -i -t -- bash -il
root@nginx-rc-a3mih:/# ls -l
total 132
drwxr-xr-x.  3 root root 4096 Jul 10 17:56 aws-ebs
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 bin
drwxr-xr-x.  2 root root 4096 Mar 13 23:46 boot
drwxr-xr-x.  5 root root  380 Jul 10 17:56 dev
drwxr-xr-x.  1 root root 4096 Jul 10 17:56 etc
drwxr-xr-x.  2 root root 4096 Mar 13 23:46 home
drwxr-xr-x.  9 root root 4096 Jul 10 17:26 lib
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 lib64
drwxr-xr-x.  2 root root 4096 May 23 17:51 media
drwxr-xr-x.  2 root root 4096 May 23 17:51 mnt
drwxr-xr-x.  2 root root 4096 May 23 17:51 opt
dr-xr-xr-x. 94 root root    0 Jul 10 17:56 proc
drwx------.  2 root root 4096 Jul 10 17:26 root
drwxr-xr-x.  1 root root 4096 Jul 10 17:56 run
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 sbin
drwxr-xr-x.  2 root root 4096 May 23 17:51 srv
dr-xr-xr-x. 13 root root    0 Jul 10 17:16 sys
drwxrwxrwt.  2 root root 4096 Jun  1 18:00 tmp
drwxr-xr-x. 10 root root 4096 Jul 10 17:26 usr
drwxr-xr-x.  1 root root 4096 Jul 10 17:26 var
root@nginx-rc-a3mih:/#
```

Gambar 6.14 Memulai Shell Interaktif Dan Membuat Daftar File

Ubah direktori (cd) ke direktori /aws-ebs dan buat daftar isinya. File yang dibuat secara default akan dicantumkan seperti yang ditunjukkan pada Gambar 6.15.

```
root@nginx-rc-a3mih:/# cd /aws-ebs
root@nginx-rc-a3mih:/aws-ebs# ls -l
total 16
drwx------. 2 root root 16384 Jul 10 17:56 lost+found
root@nginx-rc-a3mih:/aws-ebs#
```

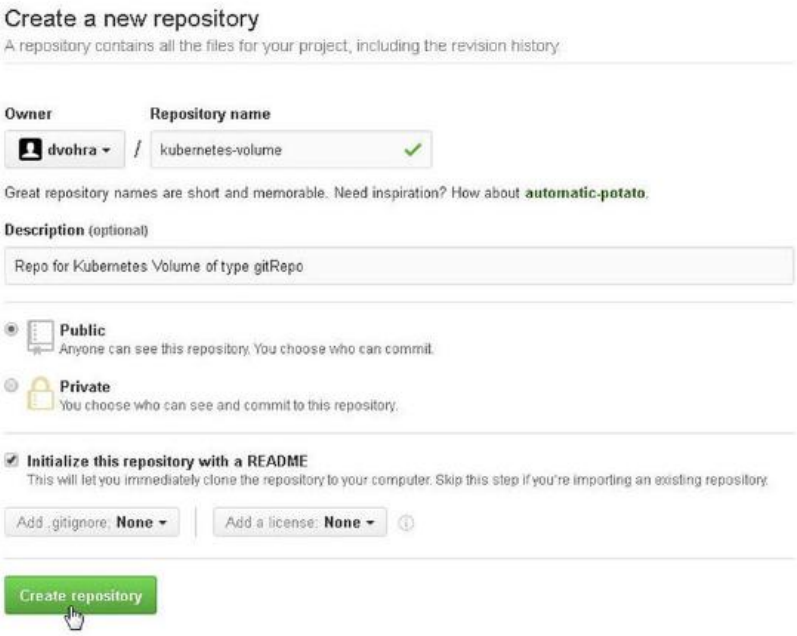
Gambar 6.15 Membuat Daftar File Default Di Direktori /Aws-Ebs

Membuat Repo Git

Untuk jenis volume gitRepo, kita perlu membuat repositori Git jika belum ada. Buat akun GitHub dan klik Repositori Baru seperti yang ditunjukkan pada Gambar 6.16.

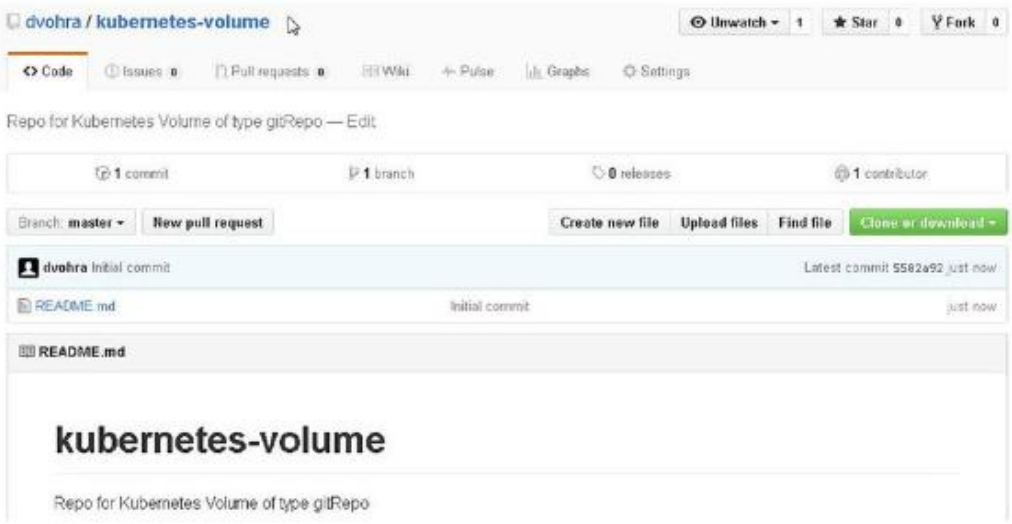
Gambar 6.16 Memilih Opsi Repositori Baru

Di jendela Buat Repositori Baru, tentukan nama Repositori, pilih opsi Repositori Publik, pilih Inisialisasi Repositori Ini dengan README, dan klik Buat Repositori seperti yang ditunjukkan pada Gambar 6.17.



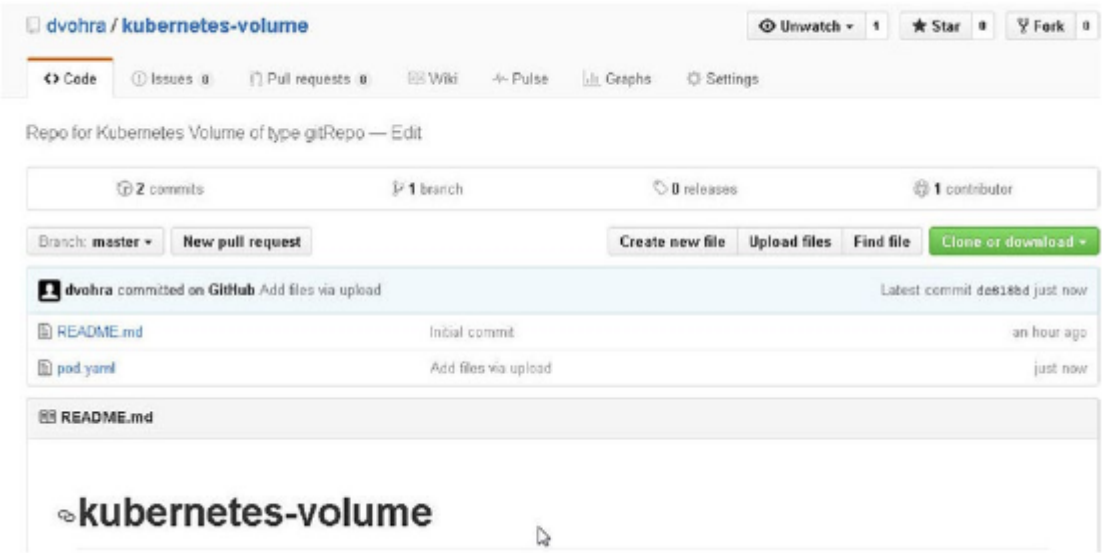
Gambar 6.17 Membuat Repositori

Repositori baru dibuat, seperti yang ditunjukkan pada Gambar 6.18.



Gambar 6.18 Repositori Kubernetes-Volume

Secara opsional, tambahkan beberapa file (pod.yaml) ke repositori, seperti yang ditunjukkan pada Gambar 6.19.

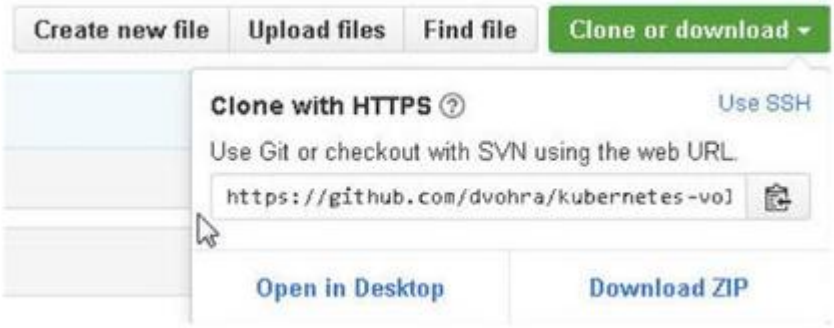


Gambar 6.19 Menambahkan File Ke Repositori

Repo kubernetes-volume harus dicantumkan dalam akun pengguna, seperti yang ditunjukkan pada Gambar 6.20.

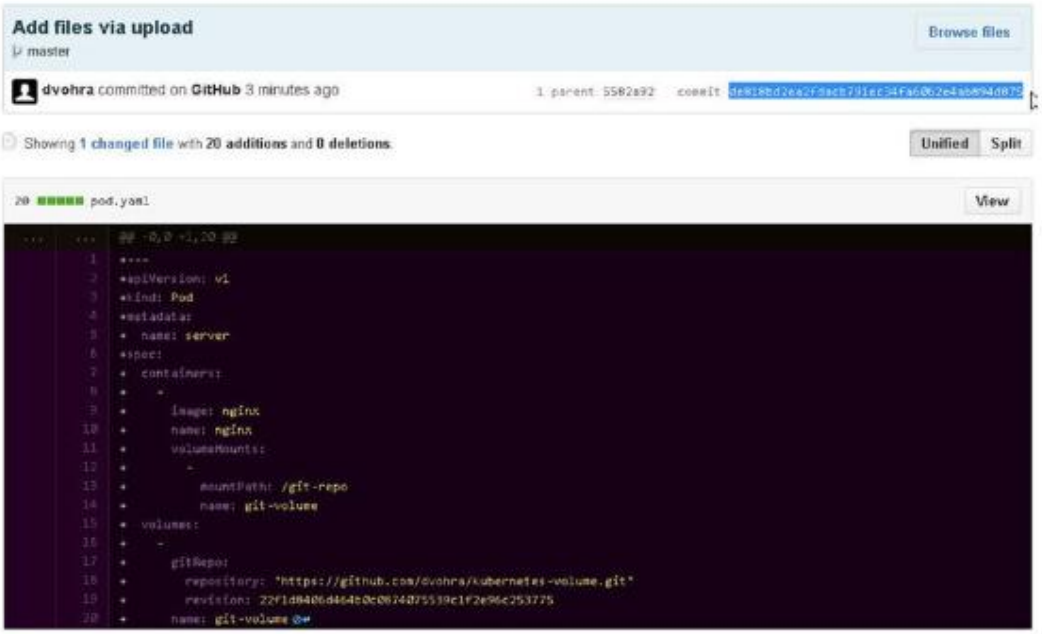
Gambar 6.20 Mencantumkan Repositori Kubernetes-Volume

Dapatkan URL web HTTPS untuk repo Git seperti yang ditunjukkan pada Gambar 6.21. Kita akan menggunakan URL web untuk menentukan volume dalam berkas spesifikasi pod.



Gambar 6.21 Memperoleh URL Web Untuk Repositori

Kita juga memerlukan nomor revisi komit, yang dapat diperoleh dari repo GitHub, seperti yang ditunjukkan pada Gambar 6.22.



Gambar 6.22 Memperoleh Nomor Revisi Komit

6.4 MENGGUNAKAN VOLUME GITREPO

Buat file spesifikasi pod pod.yaml untuk menggunakan volume gitRepo:

```
sudo vi pod.yaml
```

Salin daftar berikut ke pod.yaml. String repositori dan revisi dibiarkan kosong dalam daftar berikut, dan nilai yang diperoleh dari repositori pengguna harus diganti.

```
apiVersion: v1
kind: Pod
metadata:
  name: server
spec:
  containers:
  - image: nginx
    name: nginx
    volumeMounts:
    - mountPath: /git-repo
      name: git-volume
volumes:
- name: git-volume
  gitRepo:
    repository: ""
    revision: ""
```

Pod.yaml yang dihasilkan ditampilkan di editor vi dengan repositori dan revisi yang ditambahkan seperti yang ditunjukkan pada Gambar 6.23; repositori dan revisi akan berbeda untuk setiap pengguna.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: server
spec:
  containers:
    -
      image: nginx
      name: nginx
      volumeMounts:
        -
          mountPath: /git-repo
          name: git-volume
  volumes:
    -
      gitRepo:
        repository: "https://github.com/dvohra/kubernetes-volume.git"
        revision: de818bd2ea2fdacb791ec34fa6062e4ab894d875
      name: git-volume
~
~
:wq
```

Gambar 6.23 File pod.yaml

Buat pod dengan perintah kubectl create:

```
./kubectl create -f pod.yaml
```

Daftar pengontrol replikasi dan pod:

```
./kubectl get rc
./kubectl get pods
```

Seperti yang ditunjukkan oleh output dari perintah sebelumnya yang ditunjukkan pada Gambar 6.24, pod "server" dibuat dan dimulai. Awalnya pod mungkin terdaftar sebagai tidak berjalan, tetapi setelah beberapa detik pod akan berjalan, seperti yang juga ditunjukkan pada Gambar 6.24.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod.yaml
pod "server" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
server    0/1       ContainerCreating   0          22s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
server    1/1       Running   0          35s
core@ip-10-0-0-50 ~ $
```

Gambar 6.24 Membuat Pod

Jelaskan pod:

```
kubectl describe pod server
```

Volume gitRepo juga harus dicantumkan dalam deskripsi, seperti yang ditunjukkan pada Gambar 6.25.

```
Volumes:
  git-volume:
    Type:       GitRepo (a volume that is pulled from git when the pod is create
d)
    Repository: https://github.com/dvohra/kubernetes-volume.git
    Revision:   de818bd2ea2fdacb791ec34fa6062e4ab894d875
  default-token-nx5cy:
```

Gambar 6.25 Mencantumkan Deskripsi Volume

Jalankan shell interaktif pada pod "server". Cantumkan direktori dan file, dan Anda akan melihat jalur direktori git-repo tempat volume gitRepo dipasang tercantum seperti yang ditunjukkan pada Gambar 6.26.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
server    1/1       Running   0          13m
core@ip-10-0-0-50 ~ $ ./kubectl exec server -i -t -- bash -il
root@server:/# ls -l
total 136
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 bin
drwxr-xr-x.  2 root root 4096 Mar 13 23:46 boot
drwxr-xr-x.  5 root root  380 Jul 10 17:26 dev
drwxr-xr-x.  1 root root 4096 Jul 10 17:26 etc
drwxrwxrwx.  3 root root 4096 Jul 10 17:25 git-repo
drwxr-xr-x.  2 root root 4096 Mar 13 23:46 home
drwxr-xr-x.  9 root root 4096 Jul 10 17:26 lib
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 lib64
drwxr-xr-x.  2 root root 4096 May 23 17:51 media
drwxr-xr-x.  2 root root 4096 May 23 17:51 mnt
drwxr-xr-x.  2 root root 4096 May 23 17:51 opt
dr-xr-xr-x. 85 root root    0 Jul 10 17:26 proc
drwx------.  2 root root 4096 Jul 10 17:26 root
drwxr-xr-x.  1 root root 4096 Jul 10 17:26 run
drwxr-xr-x.  2 root root 4096 Jul 10 17:26 sbin
drwxr-xr-x.  2 root root 4096 May 23 17:51 srv
dr-xr-xr-x. 13 root root    0 Jul 10 17:16 sys
drwxrwxrwt.  2 root root 4096 Jun  1 18:00 tmp
drwxr-xr-x. 10 root root 4096 Jul 10 17:26 usr
drwxr-xr-x.  1 root root 4096 Jul 10 17:26 var
root@server:/#
```

Gambar 6.26 Memulai Shell Interaktif

Ubah direktori (cd) ke direktori git-repo. Cantumkan direktori, dan direktori kubernetes-volume akan dicantumkan, seperti yang ditunjukkan pada Gambar 6.27.

```
root@server:/# cd /git-repo
root@server:/git-repo# ls -l
total 8
drwxr-xr-x. 3 root root 4096 Jul 10 17:25 kubernetes-volume
root@server:/git-repo#
```

Gambar 6.27 Mencantumkan Direktori Kubernetes-Volume

Ubah direktori (cd) ke direktori kubernetes-volume. Cantumkan direktori dan file pod.yaml

pada repo Git akan dicantumkan, seperti yang ditunjukkan pada Gambar 6.28.

Gambar 6.28 Mencantumkan File Dalam Direktori Kubernetes-Volume

Ringkasan

Dalam bab ini, kami memperkenalkan berbagai jenis volume Kubernetes dan kemudian menggunakan dua volume ini, volume awsElasticBlockStore dan volume gitRepo. Untuk awsElasticBlockStore, volume AWS harus dibuat, dan untuk volume gitRepo, repo Git harus dibuat. Dalam bab berikutnya, kami akan membahas penggunaan berbagai jenis layanan.

BAB 7
MENGGUNAKAN LAYANAN

7.1 MENGELOLA LAYANAN DAN PENGONTROL REPLIKASI DALAM KUBERNETES

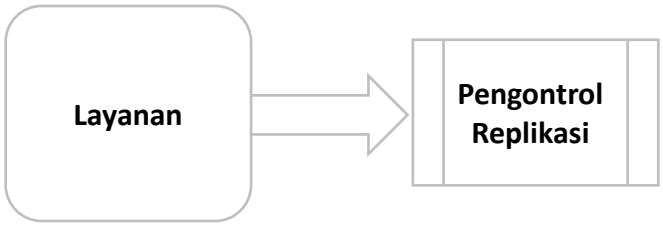
Layanan Kubernetes adalah abstraksi yang melayani sekumpulan pod. Pod yang didefinisikan atau direpresentasikan oleh layanan dipilih menggunakan pemilih label yang ditentukan dalam spesifikasi layanan. Ekspresi pemilih label layanan harus disertakan dalam label pod agar layanan dapat merepresentasikan pod. Misalnya, jika ekspresi pemilih layanan adalah "app=hello-world", label pod harus menyertakan label "app=hello-world" agar layanan dapat merepresentasikan pod.

Layanan diakses di satu atau beberapa titik akhir yang disediakan oleh layanan. Jumlah titik akhir yang tersedia sama dengan jumlah replika pod untuk pengontrol penyebaran/replikasi. Agar dapat mengakses layanan di luar klusternya, layanan harus diekspos di alamat IP eksternal. Kolom ServiceType menentukan cara layanan diekspos. Secara default ServiceType adalah ClusterIP, yang mengekspos layanan hanya di dalam kluster dan bukan di IP eksternal. ServiceType lainnya adalah NodePort dan LoadBalancer, yang mengekspos layanan pada IP eksternal.

Masalah

Layanan adalah contoh klasik dari Prinsip Tanggung Jawab Tunggal (SRP). Pertimbangkan alternatif bahwa layanan digabungkan erat dengan pengontrol replikasi (RC) seperti yang ditunjukkan pada Gambar 7.1. Masalah berikut akan muncul.

- Jika pengontrol replikasi atau layanan dimodifikasi, yang lain juga harus dimodifikasi, karena keduanya saling bergantung. Jika pengontrol replikasi dihapus dan diganti dengan yang lain, layanan juga perlu diganti.
- Untuk tim DevOp yang "berfungsi tinggi", biasanya aplikasi memiliki beberapa jalur rilis, yang bisa harian atau mingguan. Untuk beberapa aplikasi rilis, biasanya ada beberapa versi pengontrol replikasi yang dapat hidup berdampingan. Beberapa RC tidak dapat hidup berdampingan dengan kopling pengontrol/layanan.



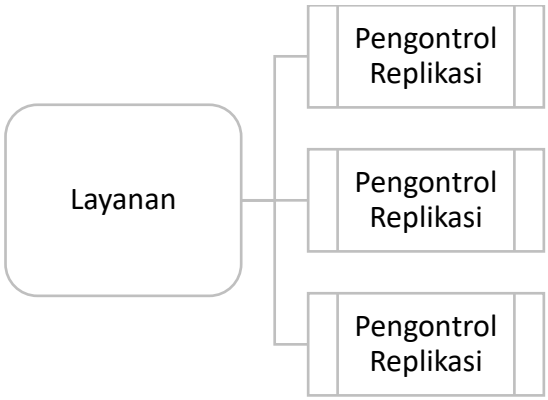
Gambar 7.1 Mencantumkan Node Dalam Kluster Kubernetes

Masalah lain yang terkait dengan layanan adalah bahwa ketika RC baru dibuat, pod tidak langsung dimulai dan memerlukan waktu tertentu (yang bisa jadi beberapa detik). Layanan yang mewakili RC perlu mengetahui kapan kontainer dalam pod siap sehingga layanan dapat

mengarahkan lalu lintas ke pod.

Solusi

Layanan adalah objek REST yang mirip dengan pod dan menyediakan manfaat berorientasi objek seperti modularitas atau pengemasan, abstraksi, dan penggunaan ulang. Memisahkan layanan dari pengontrol mengimplementasikan SRP, dan layanan atau pengontrol dapat dimodifikasi atau dihapus tanpa harus memodifikasi atau menghapus yang lain. Beberapa pengontrol replikasi dapat disimpan tanpa batas waktu, seperti yang ditunjukkan pada Gambar 7.2, untuk memenuhi persyaratan tim DevOps. Pengontrol replikasi hanya mengelola pod, dan layanan hanya mengekspos titik akhir ke pod akses. Memisahkan pengontrol dan layanan adalah pola desain manajemen.



Gambar 7.2 Layanan Yang Terkait Dengan Beberapa Pengontrol Replikasi

Pola manajemen lain yang digunakan dalam layanan adalah pemeriksaan kesiapan. Pemeriksaan kesiapan digunakan untuk mengetahui apakah kontainer pod siap menerima lalu lintas.

Tinjauan Umum

Tabel 7.1 menjelaskan berbagai ServiceType.

Tabel 7.1 Jenis Layanan

Jenis Layanan	IP Eksternal	Keterangan
ClusterIP	Tidak	Defaultnya; hanya mengekspos layanan dari dalam kluster saja.
NodePort	Ya	Selain mengekspos layanan dalam suatu kluster, mengekspos layanan pada setiap node dalam kluster pada port tertentu di URL <NodeIP>:NodePort.
Penyeimbang Beban	Ya	Selain mengekspos layanan dalam klaster dan pada setiap node di klaster, mengekspos layanan pada IP LoadBalancer eksternal.

Dalam bab ini, kita akan membahas masing-masing ServiceTypes dengan sebuah contoh. Bab

ini memiliki bagian-bagian berikut:

1. Menetapkan Lingkungan
2. Membuat Layanan ClusterIP
3. Membuat Layanan NodePort
4. Membuat Layanan LoadBalancer

Menetapkan Lingkungan

Membuat instans AWS EC2 dari Amazon Linux AMI. Masuk ke instans EC2 melalui SSH menggunakan alamat IP publik:

```
ssh -i "docker.pem" ec2-user@107.23.131.161
```

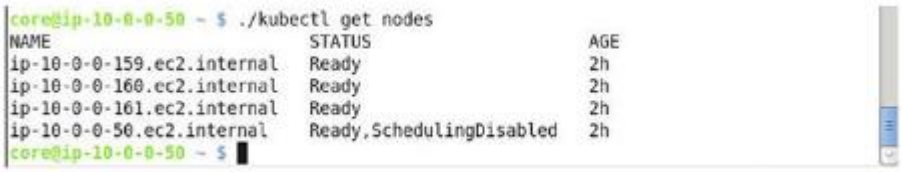
Membuat CoreOS AWS CloudFormation untuk kluster Kubernetes. Tambahkan rekaman A untuk instans pengontrol kluster 135uberne DNS 135ubern untuk CloudFormation dan log masuk SSH ke instans pengontrol:

```
ssh -I "135ubernetes-coreos.pem" core@52.203.239.87
```

Instal biner kubectl dan daftarkan node:

```
./kubectl get nodes
```

Node dalam kluster harus dicantumkan, seperti yang ditunjukkan pada Gambar 7.3.



Gambar 7.3 Mencantumkan Node Dalam Kluster Kubernetes

7.2 MEMBUAT LAYANAN CLUSTERIP

Di bagian ini, kita akan membuat layanan bertipe ClusterIP, yang merupakan tipe layanan default. Pertama, buat deployment menggunakan image Docker tutum/hello-world dengan tiga replika:

```
./kubectl run hello-world --image=tutum/hello-world --
replicas=3 --port=80
```

Selanjutnya, daftarkan deployment:

```
./kubectl get deployments
```

Deployment hello-world dibuat dan dicantumkan seperti yang ditunjukkan pada Gambar 7.4.

Gambar 7.4 Membuat Dan Mencantumkan Deployment

Daftar pod:

```
./kubectl get pods
```

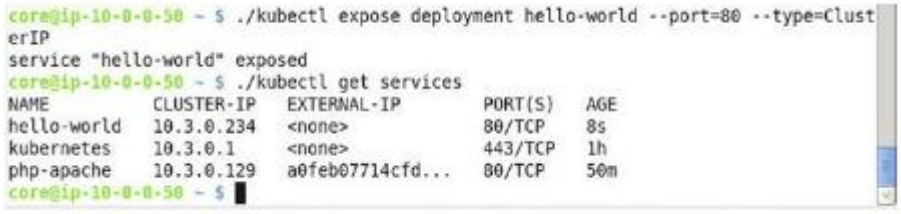
Ketiga replika pod dicantumkan. Paparkan deployment sebagai layanan bertipe ClusterIP, yang merupakan default, tetapi juga dapat ditetapkan secara eksplisit.

```
./kubectl exposed deployment hello-world --port=80 --
type=ClusterIP
```

Daftar layanan:

```
./kubectl get services
```

Layanan hello-world harus dicantumkan sebagai tambahan layanan kubernetes dan layanan lainnya, seperti yang ditunjukkan pada Gambar 7.5.



Gambar 7.5 Membuat Dan Mencantumkan Layanan

Jelaskan layanan:

```
./kubectl describe svc hello-world
```

Deskripsi layanan mencakup jenis layanan sebagai ClusterIP dan tiga titik akhir untuk layanan tersebut, seperti yang ditunjukkan pada Gambar 7.6.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc hello-world
Name:                   hello-world
Namespace:              default
Labels:                 run=hello-world
Selector:               run=hello-world
Type:                   ClusterIP
IP:                     10.3.0.234
Port:                   <unset> 80/TCP
Endpoints:              10.2.12.7:80,10.2.49.7:80,10.2.83.6:80
Session Affinity:       None
No events.

core@ip-10-0-0-50 ~ $
```

Gambar 7.6 Menjelaskan Layanan Hello-World

Layanan dapat diakses di clusterIP dan setiap titik akhir layanan. Pertama, akses IP klaster dengan perintah curl cluster-ip. Cluster-ip adalah 10.3.0.234, jadi akses layanan di curl 10.3.0.234. Markup HTML untuk layanan ditampilkan seperti yang ditunjukkan pada Gambar 7.7.

```
core@ip-10-0-0-50 ~ $ curl 10.3.0.234
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-xyqhi</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
                                        </body>
</html>
core@ip-10-0-0-50 ~ $
```

Gambar 7.7 Memanggil Titik Akhir Layanan Dengan Curl

Demikian pula, panggil layanan di titik akhir layanan 10.2.12.7 seperti yang ditunjukkan pada Gambar 7.8. Markup HTML untuk layanan tersebut ditampilkan.

Gambar 7.8 Memanggil Titik Akhir Layanan Yang Berbeda

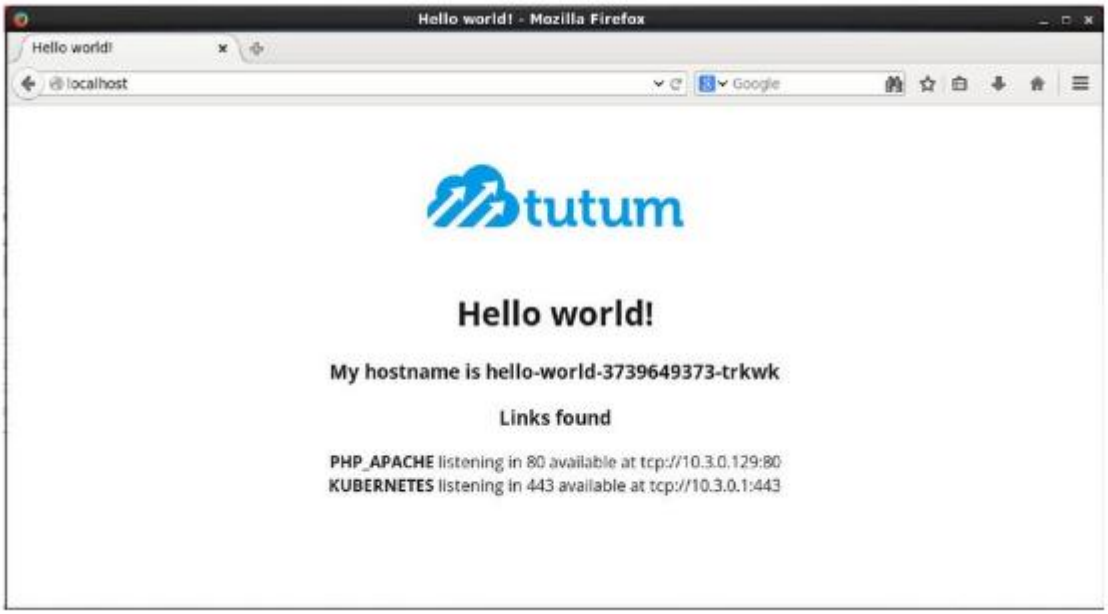
Untuk memanggil layanan di peramban web, tetapkan penerusan porta dari mesin lokal. Pertama, salin pasangan kunci yang digunakan untuk mengakses instans pengontrol kluster ke mesin lokal:

```
scp -i docker.pem ec2-user@ec2-107-23-131-161.compute-1.amazonaws.com:~/kubernetes-coreos. pem ~/kubernetes-coreos.pem
```

Kemudian atur penerusan port dari mesin lokal locahost:80 ke IP kluster layanan pada instans pengontrol:

```
ssh -i kubernetes-coreos.pem -f -nNT -L 80:10.3.0.234:80 core@ec2-52-203-239-87.compute-1. amazonaws.com
```

Penerusan port dari mesin lokal localhost:80 ke cluster-ip:80 telah ditetapkan. Panggil layanan di browser web di http://localhost, seperti yang ditunjukkan pada Gambar 7.9.



Gambar 7.9 Memanggil Layanan Di Browser

7.3 Membuat Layanan NodePort

Di bagian ini, kita akan mengekspos deployment hello-world yang sama sebagai layanan bertipe NodePort. Pertama, hapus layanan hello-world:

```
./kubectl delete svc hello-world
```

Kemudian, ekspos deployment hello-world sebagai layanan bertipe NodePort:

```
./kubectl exposed rc hello-world --port=80 --type=NodePort
```

Layanan bertipe NodePort diekspos, seperti yang ditunjukkan pada Gambar 7.10.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete svc hello-world
service "hello-world" deleted
core@ip-10-0-0-50 ~ $ ./kubectl expose deployment hello-world --port=80 --type=NodeP
ort
service "hello-world" exposed
```

Gambar 7.10 Membuat Layanan Bertipe Nodeport

Daftarkan layanan, dan layanan hello-world akan ditampilkan. Selain cluster-ip, ip eksternal <nodes> juga dicantumkan, seperti yang ditunjukkan pada Gambar 7.11. Tidak seperti cluster-ip, ip <nodes> bukanlah IP literal dan menunjukkan bahwa layanan tersebut diekspos di setiap node dalam cluster.

```
core@ip-10-0-0-50 ~ $ ./kubectl get svc
NAME          CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
hello-world   10.3.0.125   <nodes>       80/TCP    42s
kubernetes    10.3.0.1     <none>        443/TCP   2h
core@ip-10-0-0-50 ~ $
```

Gambar 7.11 Mencantumkan Layanan Tipe Nodeport

Akses layanan di cluster-ip seperti untuk layanan tipe ClusterIP, dan markup HTML akan ditampilkan seperti yang ditunjukkan pada Gambar 7.12.

```
core@ip-10-0-0-50 ~ $ curl 10.3.0.125
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-b8gar</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
                                </body>
</html>
```

Gambar 7.12 Memanggil Layanan Di Cluster-IP

Cantumkan node, seperti yang ditunjukkan pada Gambar 7.13.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                         STATUS                     AGE
ip-10-0-0-159.ec2.internal   Ready                      2h
ip-10-0-0-160.ec2.internal   Ready                      2h
ip-10-0-0-161.ec2.internal   Ready                      2h
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   2h
core@ip-10-0-0-50 ~ $
```

Gambar 7.13 Mencantumkan Node Dalam Kluster Kubernetes

Jelaskan layanan untuk mencantumkan NodePort (32624), yang merupakan port pada setiap node tempat layanan tersebut diekspos, seperti yang ditunjukkan pada Gambar 7.14.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc hello-world
Name:                   hello-world
Namespace:              default
Labels:                 run=hello-world
Selector:               run=hello-world
Type:                   NodePort
IP:                     10.3.0.125
Port:                   <unset> 80/TCP
NodePort:               <unset> 32624/TCP
Endpoints:              10.2.12.3:80,10.2.49.2:80,10.2.49.3:80
Session Affinity:       None
No events.

core@ip-10-0-0-50 ~ $
```

Gambar 7.14 Mendeskripsikan layanan NodePort

Panggil layanan pada node pekerja dengan URL Node-IP:NodePort, dan markup HTML yang sama harus dicantumkan, seperti yang ditunjukkan pada Gambar 7.15.

```
core@ip-10-0-0-50 ~ $ curl  http://ip-10-0-0-159.ec2.internal:32624
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-5ll92</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
```

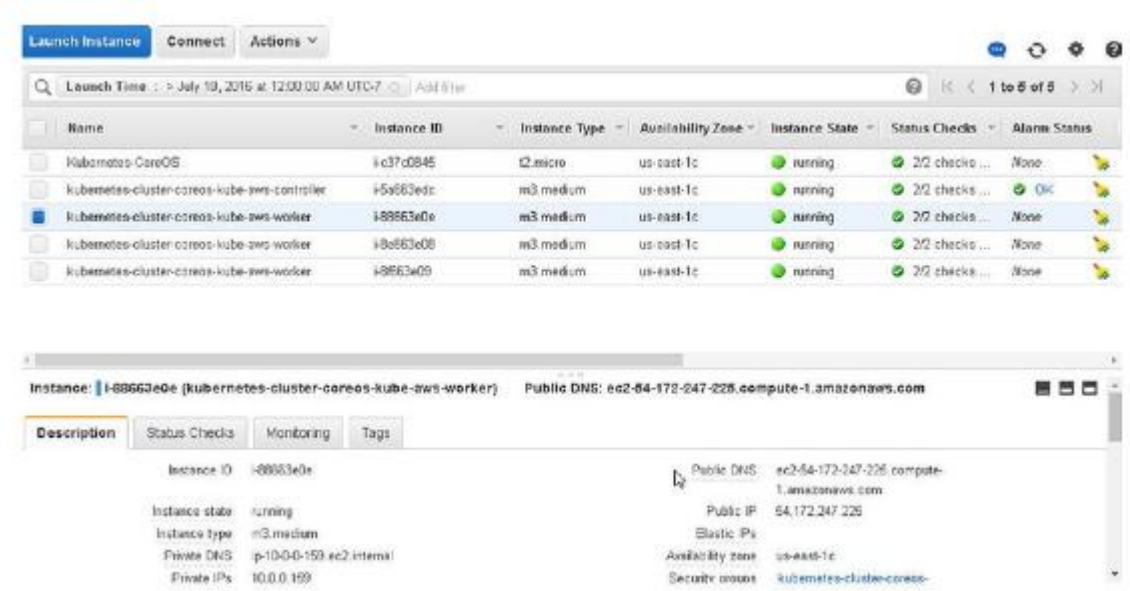
Gambar 7.15 Memanggil Layanan Di Node Pekerja

Demikian pula, layanan dapat dipanggil di node induk menggunakan port node yang sama, seperti yang ditunjukkan pada Gambar 7.16.

```
core@ip-10-0-0-50 ~ $ curl http://ip-10-0-0-50.ec2.internal:32624
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-5ll92</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
```

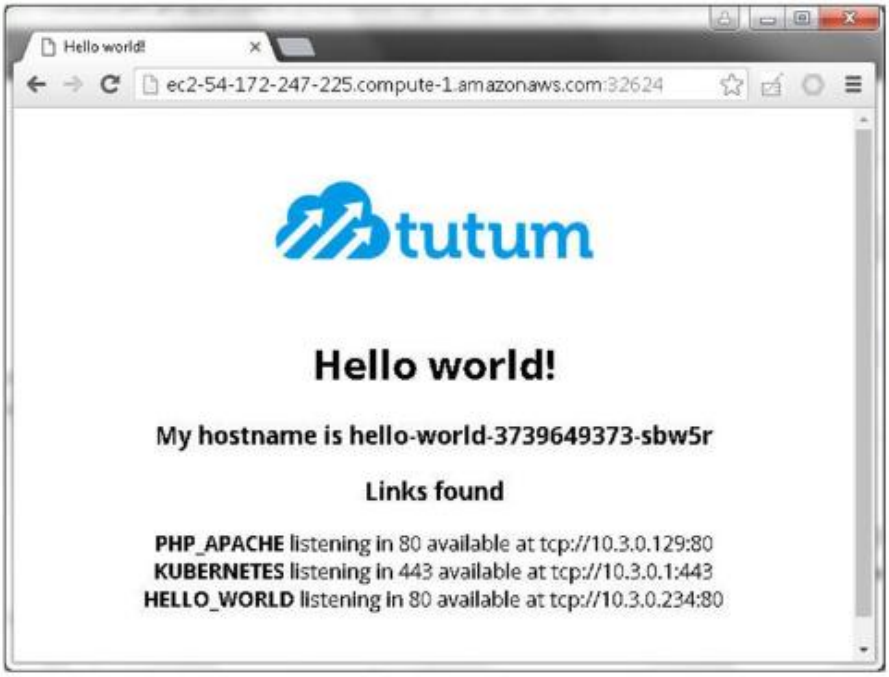
Gambar 7.16 Memanggil Layanan Di Simpul Induk

Untuk memanggil layanan di peramban web, kita tidak perlu mengatur penerusan porta. Dapatkan nama DNS publik dari simpul tempat memanggil layanan seperti yang ditunjukkan pada Gambar 7.17.



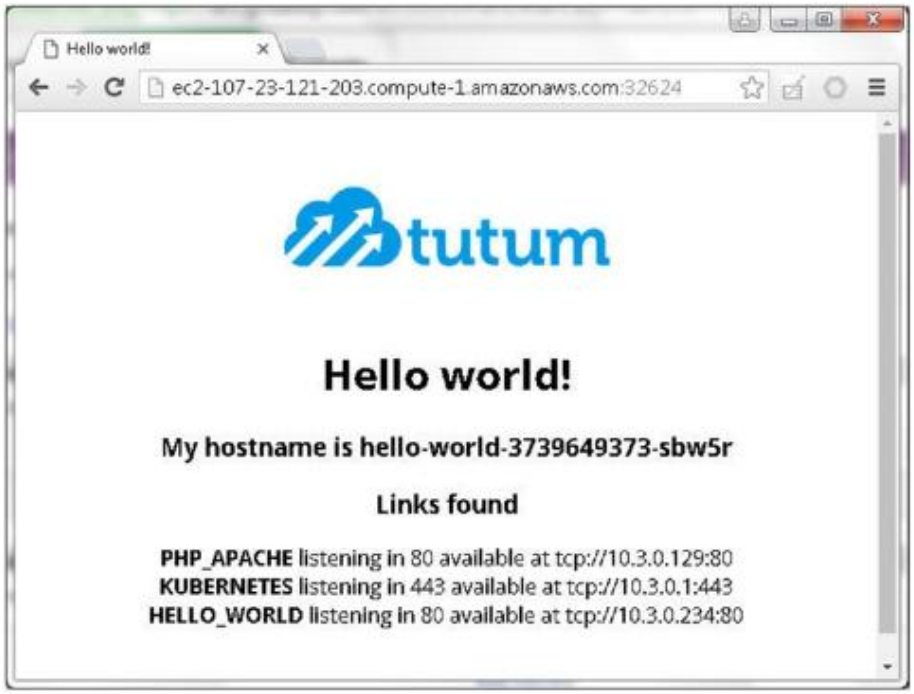
Gambar 7.17 Memperoleh DNS Public

Panggil layanan di peramban web di URL http://public-dns:32624 seperti yang ditunjukkan pada Gambar 7.18.



Gambar 7.18 Memanggil Layanan Di Peramban

Demikian pula, dapatkan public-dns untuk node pekerja lainnya. Panggil layanan di http://public-dns:node-port untuk node tersebut seperti yang ditunjukkan pada Gambar 7.19.



Gambar 7.19 Memanggil Layanan Di Node Pekerja Lain

Selain cluster-ip:80 dan node-ip:node-port, layanan juga dapat dipanggil di setiap titik akhir layanan seperti yang ditunjukkan untuk salah satu titik akhir pada Gambar 7.20.

```
core@ip-10-0-0-50 ~ $ curl 10.2.12.3:80
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-sbw5r</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
```

Gambar 7.20 Memanggil Layanan Di Titik Akhir

7.4 MEMBUAT LAYANAN LOADBALANCER

Di bagian ini, kita akan memaparkan penerapan yang sama sebagai layanan bertipe LoadBalancer. Hapus layanan hello-world dan paparkan penerapan hello-world sebagai layanan bertipe LoadBalancer:

```
./kubectl exposed deployment hello-world --port=80 --
type=LoadBalancer
```

Selanjutnya, layanan hello-world yang tercantum harus memaparkan IP eksternal selain cluster-ip seperti yang ditunjukkan pada Gambar 7.21.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete svc hello-world
service "hello-world" deleted
core@ip-10-0-0-50 ~ $ ./kubectl expose deployment hello-world --port=80 --type=LoadB
alancer
service "hello-world" exposed
core@ip-10-0-0-50 ~ $ ./kubectl get svc
NAME          CLUSTER-IP   EXTERNAL-IP        PORT(S)   AGE
hello-world   10.3.0.142   a85ad84414d08...   80/TCP    9s
kubernetes    10.3.0.1     <none>             443/TCP   2h
core@ip-10-0-0-50 ~ $
```

Gambar 7.21 Membuat Layanan Loadbalancer

Layanan dipanggil di cluster-internal cluster-ip untuk semua jenis layanan Kubernetes, seperti yang ditunjukkan pada Gambar 7.22.

Gambar 7.22 Memanggil Layanan Di Cluster-IP

Dapatkan IP eksternal, LoadBalancer Ingress, tempat layanan diekspos dengan perintah berikut:

```
./kubectl describe services hello-world | grep "LoadBalancer
Ingress"
```

LoadBalancer Ingress tercantum seperti yang ditunjukkan pada Gambar 7.23.

Gambar 7.23 Memperoleh Loadbalancer Ingress

LoadBalancer Ingress juga dapat diperoleh dari deskripsi layanan, seperti yang ditunjukkan pada Gambar 7.24.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc hello-world
Name:                   hello-world
Namespace:              default
Labels:                 run=hello-world
Selector:               run=hello-world
Type:                   LoadBalancer
IP:                     10.3.0.142
LoadBalancer Ingress:   a85ad84414d0811e699c50a558d3101d-1977318300.us-east-1.elb.am
azonaws.com
Port:                   <unset> 80/TCP
NodePort:               <unset> 31653/TCP
Endpoints:              10.2.12.3:80,10.2.49.2:80,10.2.49.3:80
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubobjectPath   Type
Reason                  Message
  ---------     --------        -----   ----                    -------------   ----
----    ------                  -------
  5m            5m              1       {service-controller }                   Norm
al              CreatingLoadBalancer    Creating load balancer
  5m            5m              1       {service-controller }                   Norm
al              CreatedLoadBalancer     Created load balancer


core@ip-10-0-0-50 ~ $
```

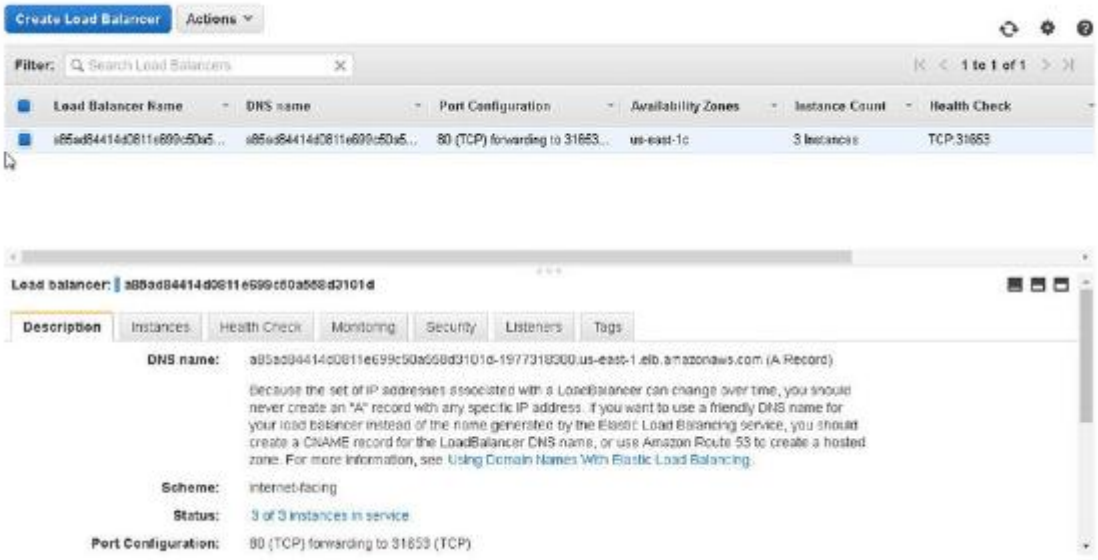
Gambar 7.24 Loadbalancer Ingress Juga Tercantum Dalam Deskripsi Layanan

Panggil layanan di IP LoadBalancer Ingress seperti yang ditunjukkan pada Gambar 7.25.

```
core@ip-10-0-0-50 ~ $ curl a85ad84414d0811e699c50a558d3101d-1977318300.us-east-1.elb
.amazonaws.com
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-5ll92</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
```

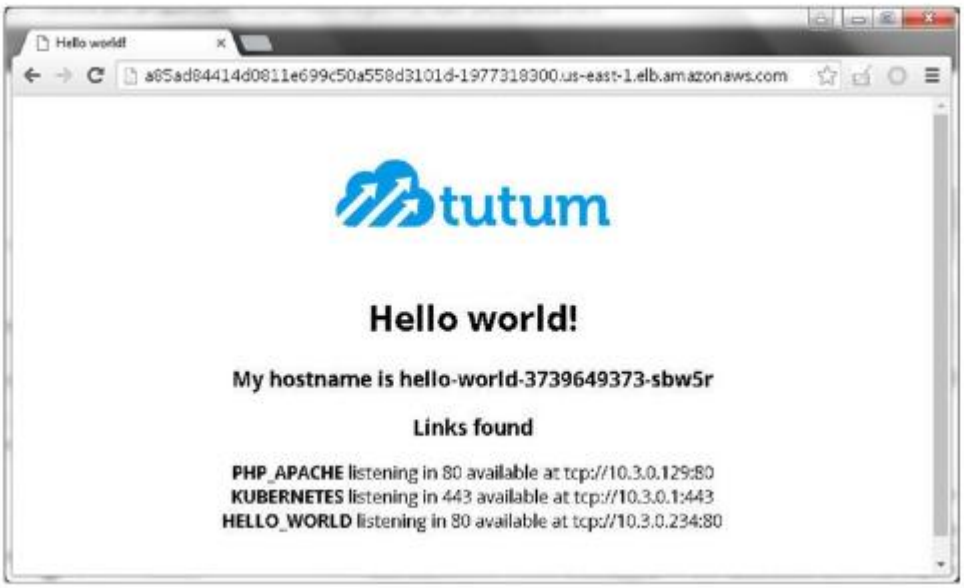
Gambar 7.25 Memanggil Layanan Di Loadbalancer Ingress

Penyeimbang beban AWS dibuat untuk layanan berjenis LoadBalancer. LoadBalancer Ingress adalah DNS publik penyeimbang beban, seperti yang ditunjukkan di Konsol EC2 pada Gambar 7.26.



Gambar 7.26 DNS Publik Loadbalancer

Untuk memanggil layanan berjenis LoadBalancer, akses DNS publik di browser seperti yang ditunjukkan pada Gambar 7.27.



Gambar 7.27 Memanggil DNS Publik Di Browser

Selain cluster-ip dan DNS publik penyeimbang beban, layanan juga dapat dipanggil di salah satu titik akhir layanan, seperti yang ditunjukkan untuk salah satu titik akhir pada Gambar 7.28.

```
core@ip-10-0-0-50 ~ $ curl 10.2.12.3
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='s
tylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans","Helvetica Neue",Helvetica,Arial,sans-serif
;
        }

        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is hello-world-3739649373-sbw5r</h3>                    <h3>
Links found</h3>
                                        <b>PHP_APACHE</b> listening in 80 available
at tcp://10.3.0.129:80<br />
                                                <b>KUBERNETES</b> listening in 443 a
vailable at tcp://10.3.0.1:443<br />
                                                <b>HELLO_WORLD</b> listening in 80 a
vailable at tcp://10.3.0.234:80<br />
                                </body>
```

Gambar 7.28 Memanggil Titik Akhir Layanan

Ringkasan

Dalam bab ini, kami memperkenalkan berbagai jenis layanan Kubernetes. Jenis layanan ClusterIP adalah default dan dipanggil pada IP klaster. Jenis layanan NodePort juga diekspos pada setiap node dalam klaster selain IP klaster. Jenis layanan LoadBalancer diekspos pada DNS AWS LoadBalancer selain diekspos pada IP klaster, dan setiap node dalam klaster. Selanjutnya, kami membuat contoh penerapan dan pod untuk setiap jenis layanan dan memanggilnya dari IP atau DNS tempat layanan tersebut dapat dipanggil. Dalam bab berikutnya, kami akan membahas penggunaan pembaruan bergulir.

BAB 8
MENGGUNAKAN PEMBARUAN BERGULIR

8.1 PEMBARUAN BERGULIR PADA PENGONTROL REPLIKASI DI KUBERNETES

Umumnya, spesifikasi pengontrol replikasi atau citra kontainer diperbarui. Jika pengontrol replikasi dibuat dari citra atau berkas definisi sebelumnya, pengontrol replikasi perlu diperbarui.

Masalah

Jika citra Docker atau spesifikasi pengontrol untuk pengontrol replikasi telah diperbarui saat pod pengontrol replikasi berjalan, pengontrol replikasi perlu dihapus dan pengontrol replikasi baru dibuat berdasarkan citra Docker atau spesifikasi pengontrol yang diperbarui. Mematikan aplikasi akan menyebabkan aplikasi menjadi tidak tersedia. Salah satu praktik terbaik pengembangan perangkat lunak DevOps dan Agile adalah Penerapan Berkelanjutan. Tujuan Penerapan Berkelanjutan adalah meminimalkan waktu tunggu antara rilis/pembuatan aplikasi baru yang sedang dikembangkan dan digunakan dalam produksi.

Solusi

Selama pembaruan bergulir, pod untuk pengontrol replikasi versi sebelumnya dihentikan dan pod untuk pengontrol baru dimulai. Pod versi sebelumnya dimatikan menggunakan mekanisme "graceful termination", yang menyediakan panggilan balik ke kontainer sejumlah waktu yang dapat dikonfigurasi sebelum kontainer dihentikan untuk memungkinkan kontainer dimatikan dengan anggun, yang menyiratkan bahwa status kontainer dalam memori dipertahankan dan koneksi terbuka ditutup. Mekanisme "graceful termination" yang digunakan Kubernetes adalah pola desain manajemen kontainer tunggal.

Gambar 8.1 Pembaruan Bergulir Dari RC1 Ke RC2

Seperti yang ditunjukkan pada Gambar 8-1, untuk pembaruan bergulir pengontrol replikasi RC1 ke RC2, RC1 awalnya memiliki tiga pod dan RC2 tidak memiliki pod. Pada fase berikutnya, RC1 memiliki dua pod dan RC2 satu pod. Pada fase ketiga, RC1 memiliki satu pod dan RC2 dua pod. Ketika pembaruan bergulir selesai, RC1 tidak memiliki pod dan RC2 tiga pod. Pembaruan bergulir dilakukan satu pod pada satu waktu.

Pola manajemen lain yang digunakan dalam pembaruan bergulir adalah pemisahan pengontrol/layanan, yang mengikuti Prinsip Tanggung Jawab Tunggal. Jika pengontrol dan layanan saling terkait erat, beberapa pengontrol tidak dapat dikaitkan dengan satu layanan karena pengontrol replikasi baru dibuat dan yang lama dihapus. Salah satu persyaratan pembaruan bergulir adalah beberapa pengontrol dikaitkan dengan layanan sementara RC versi lama dihapus (satu pod pada satu waktu seperti yang dibahas dalam bab ini) dan pod untuk RC baru dimulai.

Ikhtisar

Kubernetes menyediakan mekanisme yang dengannya pengontrol replikasi yang sedang berjalan dapat diperbarui ke citra atau spesifikasi yang lebih baru saat sedang berjalan—yang disebut pembaruan bergulir. Pengontrol replikasi atau layanan yang mewakili pengontrol replikasi tidak menjadi tidak tersedia setiap saat. RC diperbarui satu pod pada satu waktu sehingga pada waktu tertentu jumlah pod di RC berada pada tingkat replikasi yang ditentukan. Dalam bab ini, kita akan menggunakan pembaruan bergulir untuk memperbarui pengontrol replikasi dan penyebaran saat pengontrol replikasi atau penyebaran sedang berjalan. Topik meliputi yang berikut:

- Menetapkan lingkungan
- Pembaruan bergulir dengan file definisi RC
- Pembaruan bergulir dengan memperbarui citra kontainer
- Mengembalikan pembaruan
- Hanya menggunakan file atau citra
- Pembaruan bergulir pada penyebaran dengan file penyebaran

Mengatur Lingkungan

Buat instans Amazon EC2 berdasarkan Amazon Linux AMI. Dapatkan IP publik instans dan log masuk SSH ke instans:

```
ssh -i "docker.pem" ec2-user@54.87.191.230
```

Buat kluster Kubernetes menggunakan AWS CloudFormation yang terdiri dari satu pengontrol dan tiga node pekerja yang menjalankan CoreOS, seperti yang ditunjukkan pada Gambar 8.2.

Gambar 8.2 Instans CloudFormation EC2

Setelah memulai kluster dan menyiapkan rekaman A untuk alamat IP instans pengontrol dalam nama DNS publik untuk CloudFormation, log masuk SSH ke instans pengontrol:

```
ssh -i "kubernetes-coreos.pem" core@52.205.169.82
```

Instans CoreOS pengontrol login seperti yang ditunjukkan pada Gambar 8.3.

```
[ec2-user@ip-10-0-0-126 ~]$ ssh -i "kubernetes-coreos.pem"  core@52.205.169.82
The authenticity of host '52.205.169.82 (52.205.169.82)' can't be established.
ECDSA key fingerprint is 37:1a:80:3e:d9:f4:d6:97:af:8a:a3:36:44:cf:bc:58.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.205.169.82' (ECDSA) to the list of known hosts.
CoreOS stable (1010.6.0)
Update Strategy: No Reboots
core@ip-10-0-0-50 ~ $
```

Gambar 8.3 Login SSH Ke Instance Kontroler

Instal biner kubectl dan daftarkan node:

```
./kubectl get nodes
```

Node kontroler tunggal dan tiga node pekerja tercantum seperti yang ditunjukkan pada Gambar 8.4.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-196.ec2.internal    Ready                      14m
ip-10-0-0-197.ec2.internal    Ready                      14m
ip-10-0-0-198.ec2.internal    Ready                      14m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   14m
core@ip-10-0-0-50 ~ $
```

Gambar 8.4 Mencantumkan Node

Perintah kubectl rolling-update digunakan untuk melakukan pembaruan bergulir. Sintaks untuk perintah rolling-update adalah sebagai berikut.

```
kubectl rolling-update NAMA_KONTROLER_LAMA
([NAMA_KONTROLER_BARU] --image=GAMBAR_KONTROLER_BARU | -f
SPESIFIKASI_KONTROLER_BARU)
```

Nama kontainer lama harus ditentukan, dan jika nama pengontrol baru akan ditetapkan ke RC yang diperbarui, nama pengontrol baru dapat ditentukan. Baik citra kontainer baru maupun spesifikasi kontainer baru harus ditentukan sebagai parameter perintah. Selanjutnya, kita akan membahas tentang melakukan pembaruan bergulir menggunakan masing-masing metode; citra kontainer baru dan spesifikasi pengontrol baru.

8.2 PEMBARUAN BERGULIR DENGAN BERKAS DEFINISI RC

Di bagian ini, kita akan membahas pembaruan bergulir dari pengontrol replikasi yang ada dengan menyediakan berkas definisi RC ke perintah kubectl rolling-update. Persyaratan berikut berlaku untuk pembaruan bergulir RC.

1. Pengontrol replikasi baru harus berada dalam namespace yang sama.
2. Nama pengontrol replikasi baru dalam berkas definisi tidak boleh sama dengan pengontrol replikasi yang ada yang sedang diperbarui.
3. Pengontrol replikasi baru harus menentukan setidaknya satu kunci yang cocok dengan nilai yang tidak sama di bidang pemilih.

Pertama, buat pengontrol replikasi yang akan diperbarui. Berkas definisi RC berikut mysql.yaml membuat RC yang disebut mysql dan menentukan tiga replika. Buat berkas definisi mysql.yaml dalam editor vi:

```
sudo vi mysql.yaml
```

Salin kode sumber berikut ke berkas definisi:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    matchLabels: # Gunakan matchLabels untuk Deployment
      app: mysql-app
      deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
```

```
        deployment: v1
    spec:
      containers:
      - name: mysql
        image: mysql
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: mysql
```

Buat pengontrol replikasi dengan perintah berikut:

```
kubectl create –f mysql.yaml
```

RC bernama mysql dengan tiga replika harus dibuat. Selanjutnya, ubah file mysql.yaml berdasarkan persyaratan yang tercantum sebelumnya. mysql.yaml berikut menentukan nama RC yang berbeda dan nilai yang berbeda untuk kunci penyebaran di pemilih. Secara opsional, tag gambar Docker dapat berbeda.

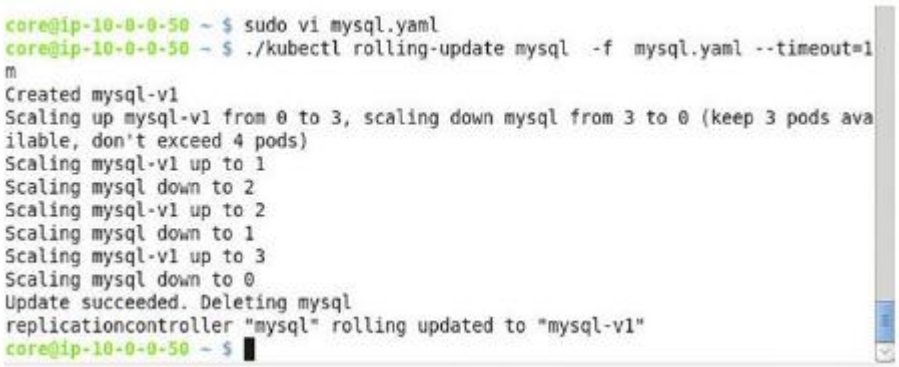
```
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql-v1
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v2
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v2
spec:
  containers:
  - name: mysql
    image: mysql:5.5
    ports:
    - containerPort: 3306
    env:
    - name: MYSQL_ROOT_PASSWORD
```

```
      value: mysql
```

Selanjutnya, lakukan pembaruan bergulir ke RC yang disebut mysql menggunakan file definisi mysql.yaml. Secara opsional, tentukan batas waktu untuk pembaruan bergulir. Ketika batas waktu ditetapkan, pod yang diperbarui ke citra atau spesifikasi baru tidak dibatalkan setelah batas waktu berlalu dan pembaruan bergulir telah berakhir. Pembaruan bergulir dapat dilakukan lagi dan dilanjutkan dari pembaruan sebelumnya.

```
kubectl rolling-update mysql -f mysql.yaml --timeout=1m
```

RC mysql diperbarui ke mysql-v1 seperti yang ditunjukkan pada Gambar 8-5. Selanjutnya RC mysql dihapus.



Gambar 8.5 Pembaruan Bergulir Pengontrol Replikasi

Hapus RC mysql-v1, karena kita akan menggunakan nama RC yang sama di bagian berikutnya:

```
kubectl delete rc mysql-v1
```

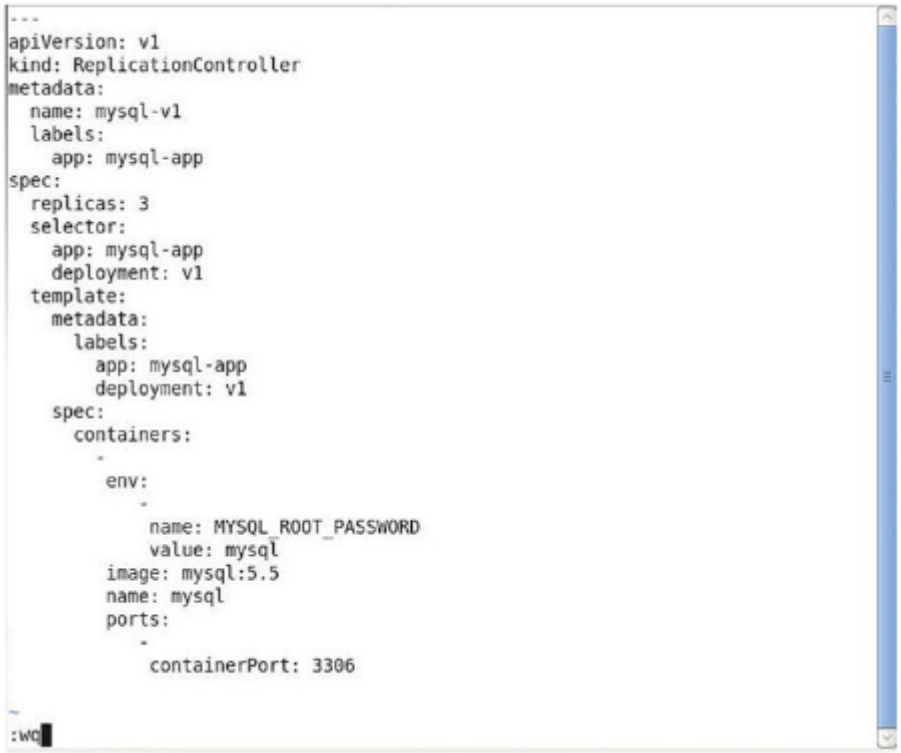
8.3 ROLLING UPDATE DENGAN MEMPERBARUI CITRA KONTAINER

Pada bagian ini, kita akan memperbarui RC dengan memperbarui citra Docker. Pertama, buat RC menggunakan file definisi mysql.yaml berikut:

```
apiVersion: v1
kind: ReplicationController # Sebaiknya gunakan Deployment
metadata:
  name: mysql-v1
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
```

```
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
      - name: mysql
        image: mysql:5.5 # Sebaiknya gunakan versi yang lebih
baru dan tag yang spesifik
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "mysecretpassword" # Ganti dengan password yang
SANGAT kuat!
```

Salin daftar sebelumnya ke file mysql.yaml dalam editor vi seperti yang ditunjukkan pada Gambar 8.6.



Gambar 8.6 Berkas Mysql.Yaml Dalam Editor Vi

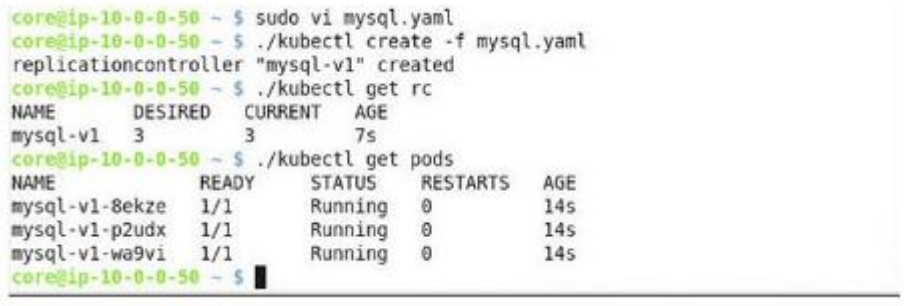
Jalankan perintah berikut untuk membuat RC:

```
kubectl create –f mysql.yaml
```

Daftar RC dan pod:

```
kubectl get rc kubectl get pods
```

RC mysql-v1 dibuat dan dicantumkan. Ketiga pod juga dicantumkan seperti yang ditunjukkan pada Gambar 8.7.

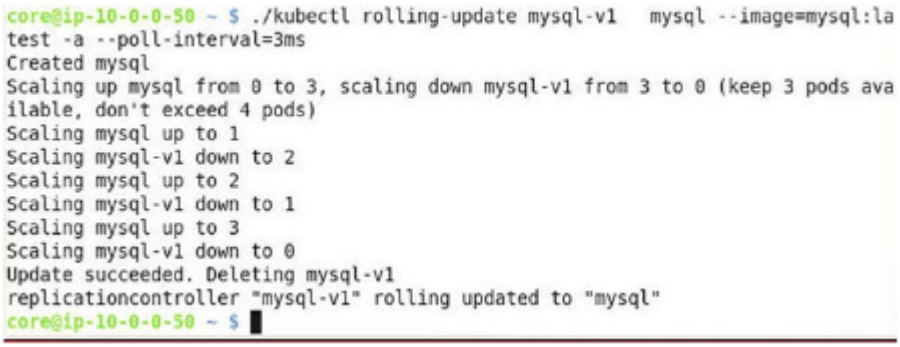


Gambar 8.7 Membuat RC Dan Mencantumkan Pod

Perintah berikut membuat pembaruan bergulir pada RC dengan tag gambar baru dan nama RC baru. Opsi `–a` menampilkan semua label, dan `--poll-interval` menentukan interval antara polling pengontrol replikasi untuk status setelah pembaruan.

```
kubectl rolling-update mysql-v1 mysql --image=mysql:latest -a --poll-interval=3ms
```

`RC mysql-v1` diperbarui secara bergulir ke mysql seperti yang ditunjukkan pada Gambar 8.8. Selanjutnya `RC mysql-v1` dihapus.



Gambar 8.8 Pembaruan Bergulir Ke RC Menggunakan Gambar Docker

Setelah pembaruan, daftarkan RC dan pod:

```
kubectl get rc kubectl get pods
```

RC, mysql, dan pod yang berbeda sekarang terdaftar, seperti yang ditunjukkan pada Gambar 8.9.

Gambar 8.9 Mencantumkan RC Dan Pod

Pembaruan bergulir pada RC tidak harus menggunakan nama RC baru. Sebagai contoh, lakukan pembaruan bergulir dengan tag gambar baru dan nama RC yang sama.

```
kubectl rolling-update mysql --image=mysql:5.6
```

Untuk tujuan memperbarui RC, RC sementara dibuat dan pembaruan diterapkan ke RC seperti yang ditunjukkan pada Gambar 8.10. Selanjutnya, RC mysql asli dihapus dan RC sementara diganti nama menjadi mysql, sehingga nama RC tetap sama.

Gambar 8.10 Rolling Update Menggunakan Nama RC Yang Sama

Rolling update tidak harus menggunakan image Docker yang sama. Sebagai contoh, gunakan image yang berbeda, postgres, untuk memperbarui RC yang disebut mysql dan berdasarkan image mysql. Perintah berikut memperbarui image mysql ke image postgresql menggunakan `image=postgres`:

```
kubectl rolling-update mysql postgresql --image=postgres
```

RC mysql diperbarui ke RC postgresql seperti yang ditunjukkan pada Gambar 8.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update mysql postgresql  --image=postgre
s
Created postgresql
Scaling up postgresql from 0 to 3, scaling down mysql from 3 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling postgresql up to 1
Scaling mysql down to 2
Scaling postgresql up to 2
Scaling mysql down to 1
Scaling postgresql up to 3
Scaling mysql down to 0
Update succeeded. Deleting mysql
replicationcontroller "mysql" rolling updated to "postgresql"
core@ip-10-0-0-50 ~ $
```

Gambar 8.11 Rolling Update Menggunakan Image Docker Yang Berbeda

Daftar RC dan pod:

```
kubectl get rc kubectl get pods
```

RC dan pod yang berbeda harus dicantumkan, seperti yang ditunjukkan pada Gambar 8.12. Tidak hanya RC yang diperbarui, pod juga menjalankan perangkat lunak yang berbeda.

```
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME         DESIRED   CURRENT   AGE
postgresql   3         3         4m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
postgresql-gh0ix   1/1       Running   0          4m
postgresql-h40u4   1/1       Running   0          2m
postgresql-vdde7   1/1       Running   0          1m
core@ip-10-0-0-50 ~ $
```

Gambar 8.12 Mencantumkan RC Dan Pod Yang Diperbarui

Pembaruan bergulir kembali ke RC berbasis citra mysql dapat dilakukan dengan menjalankan perintah yang serupa. Kemudian cantumkan RC dan pod baru untuk RC baru:

```
kubectl rolling-update postgresql mysql --image=mysql kubectl
get rc
kubectl get pods
```

RC postgresql diperbarui ke RC berbasis citra mysql yang bernama mysql, seperti yang ditunjukkan pada Gambar 8.13. RC dan pod baru tercantum.

```
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update postgresql mysql  --image=mysql
Created mysql
Scaling up mysql from 0 to 3, scaling down postgresql from 3 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling mysql up to 1
Scaling postgresql down to 2
Scaling mysql up to 2
Scaling postgresql down to 1
Scaling mysql up to 3
Scaling postgresql down to 0
Update succeeded. Deleting postgresql
replicationcontroller "postgresql" rolling updated to "mysql"
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME      DESIRED   CURRENT   AGE
mysql     3         3         6m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
mysql-7zway   1/1       Running   0          3m
mysql-l9l4r   1/1       Running   0          6m
mysql-u92yw   1/1       Running   0          5m
core@ip-10-0-0-50 ~ $ ./kubectl describe rc mysql
Name:           mysql
Namespace:      default
Image(s):       mysql
Selector:       app=mysql-app,deployment=b96c41ab125432331f3058c0d774809f
Labels:         app=mysql-app
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                  Message
```

Gambar 8.13 Pembaruan Bergulir Kembali Ke RC

Pembaruan berkelanjutan dapat dihentikan saat sedang berlangsung dan dilanjutkan nanti. Sebagai contoh, buat file definisi ReplicationController mysql.yaml yang ditunjukkan pada Gambar 8.14.

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql-v1
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              value: mysql
          image: mysql:5.5
          name: mysql
          ports:
            -
              containerPort: 3306

~
:wq
```

Gambar 8.14 File Definisi Mysql.Yaml

Buat RC dengan perintah berikut:

```
kubectl create –f mysql.yaml
```

Daftar pod:

```
kubectl get pods
```

`RC mysql-v1` dibuat, dan pod dicantumkan seperti yang ditunjukkan pada Gambar 8.15.

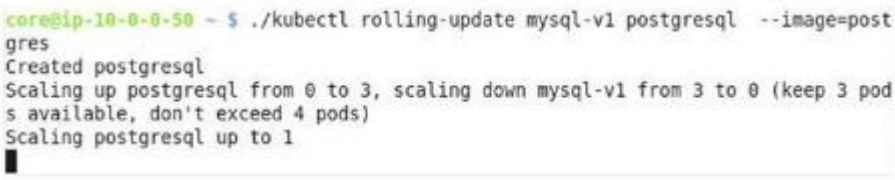
```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
replicationcontroller "mysql-v1" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME             READY     STATUS    RESTARTS   AGE
mysql-v1-2blf9   1/1       Running   0          7s
mysql-v1-6zmzb   1/1       Running   0          7s
mysql-v1-zv14x   1/1       Running   0          7s
core@ip-10-0-0-50 ~ $
```

Gambar 8.15 Membuat RC Dan Mencantumkan Pod

Lakukan pembaruan bergulir dari RC mysql-v1 ke RC baru yang disebut postgresql menggunakan citra Docker baru,
postgres:

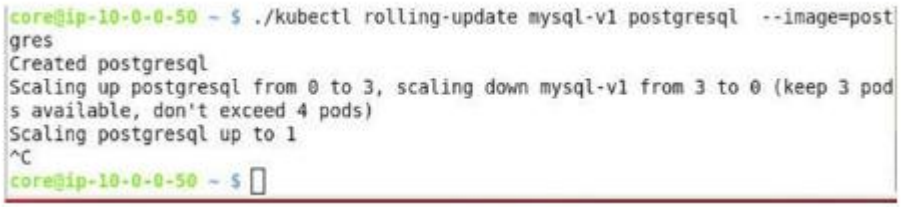
```
kubectl rolling-update mysql-v1 postgresql --image=postgres
```

Pembaruan bergulir dimulai seperti yang ditunjukkan pada Gambar 8.16.



Gambar 8.16 Memulai Pembaruan Bergulir

Saat pembaruan bergulir sedang berjalan, hentikan pembaruan dengan `^C` seperti yang ditunjukkan pada Gambar 8.17.

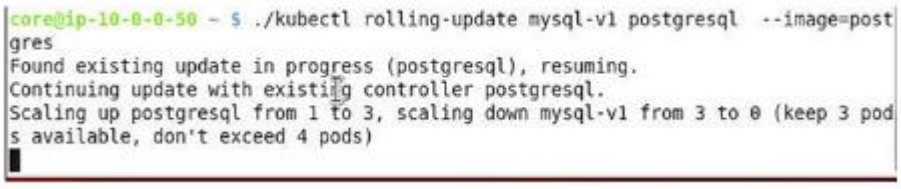


Gambar 8.17 Menghentikan Pembaruan Bergulir

Untuk melanjutkan pembaruan bergulir, jalankan perintah yang sama lagi:

```
kubectl rolling-update mysql-v1 postgresql --image=postgres
```

Seperti yang ditunjukkan pada output pada Gambar 8.18, pembaruan yang ada ditemukan dan dilanjutkan.



Gambar 8.18 Melanjutkan Pembaruan Bergulir

Pembaruan bergulir diselesaikan menggunakan pembaruan yang sudah ada seperti yang ditunjukkan pada Gambar 8.19. Berikutnya, daftarkan RC dan pod yang baru.

```
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update mysql-v1 postgresql  --image=post
gres
Created postgresql
Scaling up postgresql from 0 to 3, scaling down mysql-v1 from 3 to 0 (keep 3 pod
s available, don't exceed 4 pods)
Scaling postgresql up to 1
^C
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update mysql-v1 postgresql  --image=post
gres
Found existing update in progress (postgresql), resuming.
Continuing update with existing controller postgresql.
Scaling up postgresql from 1 to 3, scaling down mysql-v1 from 3 to 0 (keep 3 pod
s available, don't exceed 4 pods)
Scaling mysql-v1 down to 2
Scaling postgresql up to 2
Scaling mysql-v1 down to 1
Scaling postgresql up to 3
Scaling mysql-v1 down to 0
Update succeeded. Deleting mysql-v1
replicationcontroller "mysql-v1" rolling updated to "postgresql"
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME         DESIRED   CURRENT   AGE
mysql-v2     0         0         8m
postgresql   3         3         4m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
postgresql-5a0rb   1/1       Running   0          2m
postgresql-7p7a5   1/1       Running   0          1m
postgresql-g8izq   1/1       Running   0          4m
core@ip-10-0-0-50 ~ $
```

Gambar 8.19 Pembaruan Bergulir Menggunakan Pembaruan Yang Sudah Ada

8.4 MEMUTAR BALIK PEMBARUAN BERGULIR DALAM KUBERNETES

Pemutakhiran bergulir dapat dilakukan jika diperlukan. Sebagai contoh, pembaruan bergulir ke RC yang disebut postgresql dimulai menggunakan citra kontainer baru mysql:

```
kubectl rolling-update postgresql mysql --image=mysql
```

Sekarang anggaplah bahwa sementara pembaruan bergulir masih berlangsung, kita menyadari bahwa pembaruan seharusnya tidak dimulai, atau dimulai karena kesalahan, atau perlu dimulai dengan parameter yang berbeda. Dengan menggunakan `^C`, hentikan pembaruan. Kemudian jalankan perintah berikut untuk memutar balik pembaruan:

```
kubectl rolling-update postgresql mysql –rollback
```

Ketika pembaruan bergulir dihentikan, RC postgresql telah diperkecil menjadi satu pod dan RC mysql telah diperbesar menjadi dua pod. Ketika pengembalian dilakukan, pengendali postgresql yang ada akan dikurangi kembali dari satu ke tiga pod dan pengendali mysql RC akan dikurangi dari dua ke nol pod, seperti ditunjukkan pada Gambar 8.20.

```
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update postgresql mysql  --image=mysql
Created mysql
Scaling up mysql from 0 to 3, scaling down postgresql from 3 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling mysql up to 1
Scaling postgresql down to 2
Scaling mysql up to 2
Scaling postgresql down to 1
^C
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update postgresql mysql --rollback
Setting "postgresql" replicas to 3
Continuing update with existing controller postgresql.
Scaling up postgresql from 1 to 3, scaling down mysql from 2 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling postgresql up to 2
```

Gambar 8.20 Mengembalikan Pembaruan Bergulir

Berikutnya, daftar RC dan pod:

```
kubectl get rc kubectl get pods
```

RC mysql tidak terdaftar dan sebaliknya RC postgresql terdaftar, seperti yang ditunjukkan pada Gambar 8.21.

```
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update postgresql mysql  --image=mysql
Created mysql
Scaling up mysql from 0 to 3, scaling down postgresql from 3 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling mysql up to 1
Scaling postgresql down to 2
Scaling mysql up to 2
Scaling postgresql down to 1
^C
core@ip-10-0-0-50 ~ $ ./kubectl rolling-update postgresql mysql --rollback
Setting "postgresql" replicas to 3
Continuing update with existing controller postgresql.
Scaling up postgresql from 1 to 3, scaling down mysql from 2 to 0 (keep 3 pods a
vailable, don't exceed 4 pods)
Scaling postgresql up to 2
Scaling mysql down to 1
Scaling postgresql up to 3
Scaling mysql down to 0
Update succeeded. Deleting mysql
Error from server: replicationcontrollers "mysql" not found
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME         DESIRED   CURRENT   AGE
mysql-v2     0         0         15m
postgresql   3         3         10m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
postgresql-2yq3y   1/1       Running   0          1m
postgresql-g8izq   1/1       Running   0          10m
postgresql-yg6sl   1/1       Running   0          2m
core@ip-10-0-0-50 ~ $
```

Gambar 8.21 Mencantumkan RC Dan Pod Yang Dibatalkan

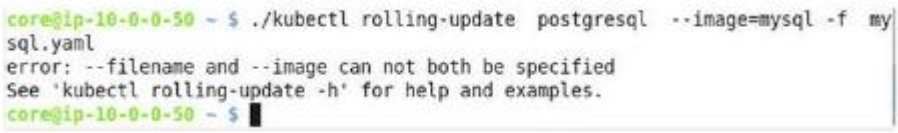
Rollback dari deployment dapat dilakukan bahkan setelah pembaruan bergulir selesai jika revisi sebelumnya tersedia. Pembaruan bergulir dari deployment dibahas di bagian selanjutnya.

Hanya Menggunakan File atau Gambar

Hanya satu dari gambar kontainer atau file definisi yang dapat digunakan, bukan keduanya. Untuk mendemonstrasikan, coba tentukan gambar dan file:

```
kubectl rolling-update mysql --image=mysql -f mysql.yaml
```

Terjadi kesalahan seperti yang ditunjukkan pada Gambar 8.22.



Gambar 8.22 Terjadi Kesalahan Jika Gambar Docker Dan File Definisi Digunakan

Pod Multi-Kontainer

Pembaruan bergulir melakukan pembaruan pada RC, yang terdiri dari replika pod, menggunakan gambar kontainer atau spesifikasi pod baru. Jika `–image` ditentukan dalam perintah kubectl rolling-update untuk melakukan pembaruan, gambar tersebut digunakan untuk memperbarui pod. Namun, bagaimana jika pod tersebut adalah pod multikontainer? Gambar hanya dapat memperbarui salah satu kontainer dalam pod, dan kontainer harus ditentukan menggunakan opsi `–container`.

Pembaruan Bergulir ke Deployment

Deployment yang dibuat menggunakan file definisi memiliki ketentuan untuk menentukan spesifikasi untuk pembaruan bergulir. Strategi default dari deployment adalah rollingUpdate dan juga dapat ditentukan secara eksplisit. Opsi lain untuk jenis strategi adalah Recreate. Kolom berikut (Tabel 8.1) dapat ditentukan untuk deployment pembaruan bergulir, default.

Tabel 8.1 Kolom untuk Pembaruan Bergulir ke Penerapan

Bidang	Keterangan	Contoh
maxTidakTersedia	Jumlah maksimum pod yang mungkin tidak tersedia selama pembaruan. Nilainya bisa berupa angka absolut, seperti 3, atau persentase, misalnya 30%. Nilai default adalah 1. Nilai tidak boleh 0 jika maxSurge adalah 0.	Jika ditetapkan ke 20%, jumlah maksimum pod yang mungkin tidak tersedia tidak boleh melebihi 20%, dan 80% pod harus selalu tersedia. Saat pembaruan dimulai, RC lama akan segera diturunkan ke 80% dan pod baru akan dimulai untuk RC baru. Saat pod baru dimulai, pod RC lama akan dihentikan, sehingga jumlah pod yang tersedia akan

		selalu 80% dari level replikasi yang dikonfigurasi.
maxSurge	Jumlah maksimum pod yang dapat berjalan di atas level yang dikonfigurasi atau diinginkan ditetapkan sebagai angka atau persentase. Nilai default adalah 1. Tidak boleh 0 jika maxUnavailable adalah 0.	Jika ditetapkan ke 10%, RC baru dapat melonjak hingga 110% dari jumlah pod yang dikonfigurasi atau diinginkan segera setelah pembaruan dimulai, tetapi tidak lebih dari 110% dari tingkat replikasi yang dikonfigurasi. Saat pod RC lama dihentikan, lebih banyak pod RC baru yang dimulai, tetapi pada waktu tertentu jumlah total pod tidak boleh melebihi 110%.

Spesifikasi Deployment menyediakan dua kolom (Tabel 8.2) untuk rollback pembaruan bergulir. Tidak satu pun dari kolom ini wajib diisi.

Tabel 8.2 Kolom untuk Rollback Pembaruan Bergulir

Bidang	Deskripsi
rollbackTo	Konfigurasi yang digunakan untuk mengembalikan penyebaran dalam rollback. RollbackConfig menyediakan revisi bidang untuk menentukan revisi yang akan dikembalikan. Jika diatur ke 0, akan dikembalikan ke revisi terakhir.
revisionHistoryLimit	Jumlah set replika lama yang akan dipertahankan untuk memungkinkan rollback.

Berikutnya, kami akan menunjukkan pembaruan berkelanjutan dari sebuah penerapan. Buat file penerapan mysql-deployment.yaml:

```
sudo vi mysql-deployment.yaml
```

Salin daftar berikut ke file definisi:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 5
  template:
    metadata:
      labels:
```

```
        app: mysql
spec:
  containers:
  - name: mysql
    image: mysql:5.5
    ports:
    - containerPort: 80
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxUnavailable: 75%
    maxSurge: 30%
rollbackTo:
  revision: 0
```

Gambar 8.23 menunjukkan berkas definisi dalam editor vi.

Gambar 8.23 Berkas Definisi Untuk Penyebaran

Buat penyebaran:

```
kubectl create -f mysql-deployment.yaml
```

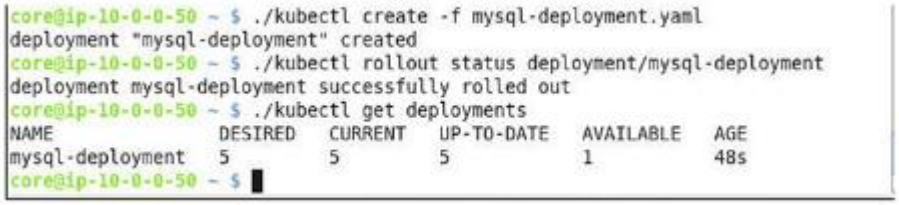
Temukan status peluncuran:

```
kubectl rollout status deployment/mysql-deployment
```

Daftar penyebaran:

```
kubectl get deployments
```

`mysql-deployment` dibuat dan diluncurkan seperti yang ditunjukkan pada Gambar 8.24.



Gambar 8.24 Membuat Dan Meluncurkan Penyebaran

Gambar Docker yang ditentukan dalam berkas definisi adalah `mysql:5.5`. Gambar dapat diperbarui menggunakan perintah:

```
kubectl set image. Sebagai contoh, perbarui gambar ke tag
terbaru:

kubectl set image deployment/mysql-deployment
mysql=mysql:latest
```

Temukan status penyebaran, daftarkan penyebaran, dan jelaskan penyebaran:

```
kubectl describe deployments
```

Seperti yang ditunjukkan dalam output yang ditunjukkan pada Gambar 8.25, `RollingUpdateStrategy 75%` tidak tersedia dan lonjakan maksimum 30%.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql-deployment.yaml
deployment "mysql-deployment" created
core@ip-10-0-0-50 ~ $ ./kubectl rollout status deployment/mysql-deployment
deployment mysql-deployment successfully rolled out
core@ip-10-0-0-50 ~ $ ./kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         5         5            1           48s
core@ip-10-0-0-50 ~ $ ./kubectl set image deployment/mysql-deployment mysql=mysq
l:latest
deployment "mysql-deployment" image updated
core@ip-10-0-0-50 ~ $ ./kubectl rollout status deployment/mysql-deployment
deployment mysql-deployment successfully rolled out
core@ip-10-0-0-50 ~ $ ./kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         5         5            0           1m
core@ip-10-0-0-50 ~ $ ./kubectl describe deployment mysql-deployment
Name:                   mysql-deployment
Namespace:              default
CreationTimestamp:      Sat, 09 Jul 2016 20:22:25 +0000
Labels:                 app=mysql
Selector:               app=mysql
Replicas:               5 updated | 5 total | 0 available | 5 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  75% max unavailable, 30% max surge
OldReplicaSets:         <none>
NewReplicaSet:          mysql-deployment-2839511385 (5/5 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                          Message
  ---------     --------        -----   ----                            --------
-----   --------        ------                          -------
```

Gambar 8.25 Menjelaskan Penyebaran

Seperti yang tercantum dalam Peristiwa, satu set replika diperkecil dan satu set replika lainnya diperbesar. Kedua set replika dicantumkan dengan kubectl get rs, tetapi hanya satu yang memiliki pod, seperti yang ditunjukkan pada Gambar 8.26.

```
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                          Message
  ---------     --------        -----   ----                            --------
-----   --------        ------                          -------
  2m            2m              1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled up replica set my
sql-deployment-227245924 to 5
  2m            2m              2       {deployment-controller }                W
arning          DeploymentRollbackRevisionNotFound      Unable to find last revi
sion.
  59s           59s             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled up replica set my
sql-deployment-2839511385 to 2
  58s           58s             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled down replica set
mysql-deployment-227245924 to 2
  58s           58s             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled up replica set my
sql-deployment-2839511385 to 5
  39s           39s             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled down replica set
mysql-deployment-227245924 to 1
  39s           39s             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled down replica set
mysql-deployment-227245924 to 0


core@ip-10-0-0-50 ~ $ ./kubectl get rs
NAME                          DESIRED   CURRENT   AGE
mysql-deployment-227245924    0         0         3m
mysql-deployment-2839511385   5         5         2m
core@ip-10-0-0-50 ~ $
```

Gambar 8.26 Penskalaan Dan Pencantuman Replicaset

Sebagai alternatif, edit deployment dengan kubectl edit: `kubectl edit deployment/mysql-deployment`. Sebagai contoh, tag gambar mysql dapat ditetapkan ke 5.5 seperti yang ditunjukkan pada Gambar 8.27.

```
  namespace: default
  resourceVersion: "22537"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/mysql-deploy
ment
  uid: 0e80cfc7-4614-11e6-840e-0a975dd0e3e5
spec:
  replicas: 5
  selector:
    matchLabels:
      app: mysql
  strategy:
    rollingUpdate:
      maxSurge: 30%
      maxUnavailable: 75%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mysql
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: mysql
        image: mysql:5.5
        imagePullPolicy: Always
        name: mysql
        ports:
        - containerPort: 3306
          protocol: TCP
```

Gambar 8.27 Mengedit Penyebaran

Ubah tag gambar ke yang terbaru, default saat tidak ada tag yang ditentukan, seperti yang ditunjukkan pada Gambar 8.28.

```
  namespace: default
  resourceVersion: "22537"
  selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/mysql-deploy
ment
  uid: 0e80cfc7-4614-11e6-840e-0a975dd0e3e5
spec:
  replicas: 5
  selector:
    matchLabels:
      app: mysql
  strategy:
    rollingUpdate:
      maxSurge: 30%
      maxUnavailable: 75%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mysql
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: mysql
        image: mysql
        imagePullPolicy: Always
        name: mysql
        ports:
        - containerPort: 3306
          protocol: TCP
:wq
```

Gambar 8.28 Mengatur Tag Gambar Ke Yang Terbaru, Yang Merupakan Default Untuk Gambar Docker Mysql

Simpan file definisi dengan :wq. Pesan deployment edited menunjukkan bahwa deployment telah diedit seperti yang ditunjukkan pada Gambar 8.29. Buat daftar deployment, kumpulan replika, dan pod. Kumpulan replika lama tidak memiliki pod, sedangkan kumpulan replika baru memilikinya.

```
core@ip-10-0-0-50 ~ $ sudo vi mysql-deployment.yaml
core@ip-10-0-0-50 ~ $ ./kubectl edit deployment/mysql-deployment
deployment "mysql-deployment" edited
core@ip-10-0-0-50 ~ $ ./kubectl rollout status deployment/mysql-deployment
deployment mysql-deployment successfully rolled out
core@ip-10-0-0-50 ~ $ ./kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         5         5            5           7m
core@ip-10-0-0-50 ~ $ ./kubectl get rs
NAME                          DESIRED   CURRENT   AGE
mysql-deployment-2212994012   0         0         5m
mysql-deployment-789224202    5         5         8m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
mysql-deployment-789224202-5ws0p    1/1       Running   0          1m
mysql-deployment-789224202-7pib1    1/1       Running   0          1m
mysql-deployment-789224202-ferm1    1/1       Running   0          1m
mysql-deployment-789224202-p69kh    1/1       Running   0          1m
mysql-deployment-789224202-ro1u3    1/1       Running   0          1m
core@ip-10-0-0-50 ~ $
```

Gambar 8.29 Deployment Diedit Dan Diluncurkan

Lakukan beberapa pembaruan bergulir lagi. Misalnya, pembaruan bergulir yang ditunjukkan pada Gambar 8.30 menyetel tag gambar ke 5.6.

```
core@ip-10-0-0-50 ~ $ ./kubectl set image deployment/mysql-deployment mysql=mysq
l:5.6
deployment "mysql-deployment" image updated
core@ip-10-0-0-50 ~ $
```

Gambar 8.30 Menerapkan Pembaruan Bergulir Ke Tag Gambar Docker Mysql

Perintah kubectl set image tidak memverifikasi validitas tag. Misalnya, misalkan gambar mysql dengan tag 5.5.5 yang tidak valid juga digunakan untuk pembaruan bergulir dan penerapan diluncurkan. Beberapa pod dari RC lama dihentikan, tetapi pod baru tidak dimulai. Mencantumkan pod menunjukkan bahwa beberapa pod memiliki Status ImagePullBackOff, atau menampilkan pesan kesalahan lain seperti yang diilustrasikan pada Gambar 8.31.

```
core@ip-10-0-0-50 ~ $ ./kubectl set image deployment/mysql-deployment mysql=mysq
l:555
deployment "mysql-deployment" image updated
core@ip-10-0-0-50 ~ $ ./kubectl rollout status deployments mysql-deployment
deployment mysql-deployment successfully rolled out
core@ip-10-0-0-50 ~ $ ./kubectl get rs
NAME                          DESIRED   CURRENT   AGE
mysql-deployment-2212994012   0         0         9m
mysql-deployment-2296945629   2         2         2m
mysql-deployment-2801114083   5         5         22s
mysql-deployment-3585318469   0         0         2m
mysql-deployment-789224202    0         0         12m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                                READY     STATUS             RESTARTS   AGE
mysql-deployment-2296945629-19o0b   1/1       Running            0          2m
mysql-deployment-2296945629-qan2a   1/1       Running            0          2m
mysql-deployment-2801114083-37h5b   0/1       ImagePullBackOff   0          45s
mysql-deployment-2801114083-k9p7v   0/1       ImagePullBackOff   0          45s
mysql-deployment-2801114083-rg12d   0/1       ErrImagePull       0          45s
mysql-deployment-2801114083-vu09g   0/1       ImagePullBackOff   0          45s
mysql-deployment-2801114083-znlrj   0/1       ImagePullBackOff   0          45s
core@ip-10-0-0-50 ~ $
```

Gambar 8.31 Pembaruan Bergulir Dengan Tag Gambar Yang Tidak Valid

Sebagai contoh lain, rolling-update dan luncurkan penerapan menggunakan gambar mysql:latest. Penerapan juga diluncurkan. Namun seperti yang ditunjukkan dalam deskripsi penerapan, hanya dua pod yang tersedia, seperti yang ditunjukkan pada Gambar 8.32.

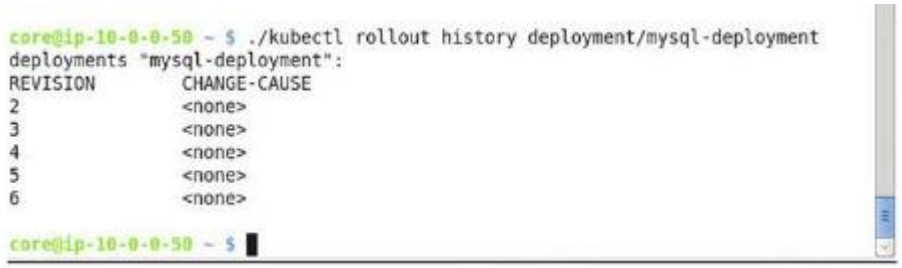
```
core@ip-10-0-0-50 ~ $ ./kubectl describe deployment
Name:                   mysql-deployment
Namespace:              default
CreationTimestamp:      Sat, 09 Jul 2016 20:31:03 +0000
Labels:                 app=mysql
Selector:               app=mysql
Replicas:               5 updated | 5 total | 2 available | 5 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  75% max unavailable, 30% max surge
OldReplicaSets:         mysql-deployment-2296945629 (2/2 replicas created)
NewReplicaSet:          mysql-deployment-2801114083 (5/5 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                          Message
  ---------     --------        -----   ----                            --------
-----   --------        ------                                          -------
  14m           14m             2       {deployment-controller }                W
arning          DeploymentRollbackRevisionNotFound      Unable to find last revi
sion.
  11m           11m             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled up replica set my
sql-deployment-2212994012 to 2
  11m           11m             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled up replica set my
sql-deployment-2212994012 to 5
  11m           11m             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled down replica set
mysql-deployment-789224202 to 1
  11m           11m             1       {deployment-controller }                N
ormal           ScalingReplicaSet                       Scaled down replica set
mysql-deployment-789224202 to 0
```

Gambar 8.32 Hanya Beberapa Replika Yang Tersedia

Jika beberapa penerapan memiliki kesalahan, penerapan dapat dikembalikan ke revisi sebelumnya. Daftar revisi penerapan.

```
kubectl rollout history deployment/mysql-deployment
```

Revisi penerapan tercantum seperti yang ditunjukkan pada Gambar 8.33.

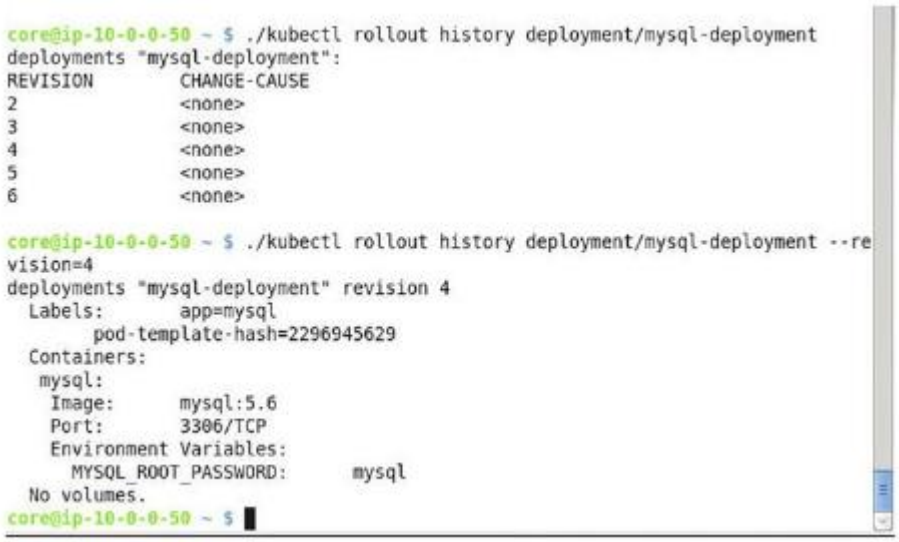


Gambar 8.33 Daftar Revisi Penerapan

Kita perlu menemukan revisi penerapan yang tidak memiliki kesalahan dan kemudian kembali ke revisi tersebut. Rincian revisi dapat ditampilkan. Misalnya, perintah berikut mencantumkan rincian revisi 4:

```
kubectl rollout history deployment/mysql-deployment --
revision=4
```

Rincian revisi 4 tercantum seperti yang ditunjukkan pada Gambar 8.34.



Gambar 8.34 Mencantumkan Detail Revisi 4

Untuk melakukan rollback ke versi sebelumnya, jalankan perintah berikut, dengan asumsi kolom rollbackTo->revision diatur ke 0 (juga default) dalam file definisi deployment:

```
kubectl rollout undo deployment/mysql-deployment
```

Deployment di-rollback seperti yang ditunjukkan pada Gambar 8.35. Cantumkan pod, dan

Anda mungkin melihat beberapa pod masih tidak berjalan, yang menunjukkan bahwa revisi yang di-rollback memiliki kesalahan.

```
core@ip-10-0-0-50 ~ $ ./kubectl rollout undo deployment/mysql-deployment
deployment "mysql-deployment" rolled back
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         7         5            2           17m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                                READY     STATUS             RESTARTS   AGE
mysql-deployment-2296945629-19o0b   1/1       Running            0          7m
mysql-deployment-2296945629-qan2a   1/1       Running            0          7m
mysql-deployment-3585318469-71ahh   0/1       ImagePullBackOff   0          40s
mysql-deployment-3585318469-83lam   0/1       ErrImagePull       0          40s
mysql-deployment-3585318469-nt4hs   0/1       ErrImagePull       0          40s
mysql-deployment-3585318469-vrcyw   0/1       ErrImagePull       0          40s
mysql-deployment-3585318469-y8r0z   0/1       ErrImagePull       0          40s
core@ip-10-0-0-50 ~ $
```

Gambar 8.35 Mengembalikan Penyebaran

Teruslah mengembalikan satu revisi pada satu waktu dan verifikasi apakah revisi tersebut valid atau kembalikan ke revisi tertentu yang diketahui valid, misalnya revisi 4:

```
kubectl rollout undo deployment/mysql-deployment --to-revision=4
```

Sekarang daftarkan pod. Seperti yang ditunjukkan pada Gambar 8.36, semua pod berjalan.

```
core@ip-10-0-0-50 ~ $ ./kubectl rollout undo deployment/mysql-deployment
deployment "mysql-deployment" rolled back
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         7         5            2           19m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                                READY     STATUS             RESTARTS   AGE
mysql-deployment-2296945629-19o0b   1/1       Running            0          9m
mysql-deployment-2296945629-qan2a   1/1       Running            0          9m
mysql-deployment-3585318469-33sus   0/1       ImagePullBackOff   0          16s
mysql-deployment-3585318469-3w1e1   0/1       ErrImagePull       0          16s
mysql-deployment-3585318469-b1dxw   0/1       ErrImagePull       0          16s
mysql-deployment-3585318469-c63wa   0/1       ImagePullBackOff   0          16s
mysql-deployment-3585318469-f280y   0/1       ImagePullBackOff   0          16s
core@ip-10-0-0-50 ~ $ ./kubectl rollout undo deployment/mysql-deployment --to-re
vision=4
deployment "mysql-deployment" rolled back
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
mysql-deployment   5         5         5            5           20m
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
mysql-deployment-2296945629-19o0b   1/1       Running   0          10m
mysql-deployment-2296945629-ohyu0   1/1       Running   0          19s
mysql-deployment-2296945629-q1n74   1/1       Running   0          19s
mysql-deployment-2296945629-qan2a   1/1       Running   0          10m
mysql-deployment-2296945629-rbw0p   1/1       Running   0          19s
core@ip-10-0-0-50 ~ $
```

Gambar 8.36 Mengembalikan Ke Revisi 4

Tidak diperlukan pengembalian lebih lanjut.

Ringkasan

Dalam bab ini kami memperkenalkan pembaruan bergulir, sebuah fitur yang berguna karena memungkinkan Anda memperbarui aplikasi yang sedang berjalan ke citra atau definisi RC yang lebih baru tanpa gangguan dalam layanan. Kami membuat pembaruan bergulir menggunakan berkas definisi RC yang diperbarui dan juga citra kontainer yang diperbarui. Kami juga mendemonstrasikan pembatalan pembaruan. Dalam bab berikutnya, kami akan membahas penjadwalan pod pada node.

BAB 9
PENJADWALAN POD PADA NODE

9.1 EFISIENSI PENJADWALAN POD DI KUBERNETES

Penjadwalan melibatkan pencarian pod yang perlu dijalankan dan menjalankannya (menjadwalkannya) pada node dalam kluster.

Masalah

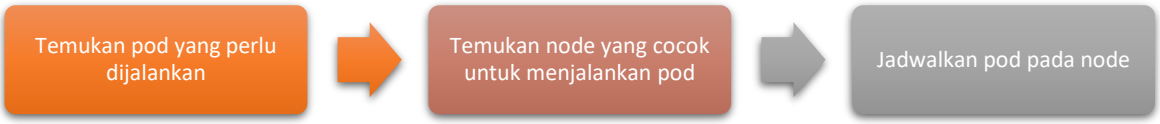
Sering kali kontainer memiliki ketergantungan di antara mereka dan perlu ditempatkan bersama pada node yang sama untuk mengurangi latensi jaringan di antara mereka. Abstraksi pod dapat merangkum beberapa kontainer, yang memecahkan masalah penempatan bersama kontainer dengan ketergantungan di antara mereka. Pola tersebut dapat diperluas lebih jauh ke ketergantungan antara pod yang perlu dijalankan pada mesin (node) yang sama atau berbeda.

Solusi

Dalam publikasi terbaru, Design Patterns for Container-based Distributed Systems, oleh Brendan Burns dan David Oppenheimer (https://www.usenix.org/node/196347), tiga jenis pola desain kontainer dibahas:

1. Pola Manajemen Kontainer Tunggal
2. Pola Aplikasi Node Tunggal, Multi-Kontainer
3. Pola Aplikasi Multi-Node

Semua pola desain ini mengharuskan pod dijadwalkan pada node tertentu dalam sebuah kluster. Kubernetes menyediakan berbagai opsi untuk menjadwalkan pod pada node tertentu dalam sebuah kluster. Urutan yang digunakan untuk menjadwalkan pod pada sebuah node ditunjukkan pada Gambar 9.1.



Gambar 9.1 Urutan Yang Digunakan Dalam Penjadwalan Pod

Ikhtisar

Kubernetes Scheduler adalah komponen atau proses Kubernetes yang berjalan bersama komponen lain seperti API Server. Tujuan Scheduler adalah untuk memantau API untuk pod yang perlu dijadwalkan, menemukan node yang sesuai untuk menjadwalkan pod, dan menjadwalkan pod, satu pod dalam satu waktu. Bab ini membahas topik-topik berikut.

1. Menetapkan kebijakan penjadwalan
2. Menetapkan lingkungan
3. Menggunakan penjadwal default
4. Menjadwalkan pod tanpa pemilih node

5. Menetapkan label node
6. Menjadwalkan pod dengan pemilih node
7. Menetapkan afinitas node
8. Menetapkan requiredDuringSchedulingIgnoredDuringExecution
9. Menetapkan preferredDuringSchedulingIgnoredDuringExecution

Menetapkan Kebijakan Penjadwalan

Penjadwalan ditentukan oleh kebijakan penjadwalan, yang melibatkan predikat dan fungsi prioritas. Penjadwalan melibatkan proses berikut, dimulai dengan semua node yang layak untuk menjadwalkan pod:

1. Memfilter node menggunakan predikat kebijakan pemfilteran. Tujuan penyaringan node adalah untuk mengecualikan node yang tidak memenuhi persyaratan tertentu dari sebuah pod.
2. Node diberi peringkat menggunakan fungsi prioritas.
3. Pod dijadwalkan ke node dengan prioritas tertinggi. Jika beberapa node memiliki prioritas yang sama, salah satu node dipilih secara acak.

Beberapa predikat penting yang menerapkan kebijakan penyaringan dibahas dalam Tabel 9.1.

Tabel 9.1 Predikat Untuk Kebijakan Penyaringan

Predikat	Keterangan
NoDiskConflict	Mengevaluasi apakah ada konflik disk karena volume yang diminta oleh pod. Jenis volume yang didukung adalah AWS EBS, GCE PD, dan Ceph RBD.
NoVolumeZoneConflict	Dengan mempertimbangkan batasan zona, mengevaluasi apakah volume yang diminta pod tersedia di zona tersebut.
PodFitsResources	Memverifikasi bahwa sumber daya yang tersedia (CPU dan memori) pada node sesuai dengan kebutuhan sumber daya pod.
PodFitsHostPorts	Memverifikasi bahwa HostPort yang diminta oleh pod belum diambil.
HostName	Jika spesifikasi pod menentukan nama node, filter semua node lainnya.
MatchNodeSelector	Memfilter node yang tidak memiliki label yang cocok sebagaimana ditetapkan dalam kolom nodeSelector pod dan anotasi pod scheduler.alpha.kubernetes.io/affinity jika ditentukan.
MaxEBSVolumeCount	Memverifikasi bahwa jumlah Volume EBS yang terpasang tidak melebihi batas 39 volume yang tersedia (1 dari 40 yang tersedia dicadangkan untuk volume root).
MaxGCEPDVolumeCount	Memverifikasi bahwa jumlah Volume PD GCE yang terpasang tidak melebihi batas 16 volume yang tersedia.
CheckNodeMemoryPressure	Pod dengan prioritas terendah (BestEffort) tidak dapat

	dijadwalkan pada node dengan kondisi tekanan memori.
CheckNodeDiskPressure	Pod tidak dapat dijadwalkan pada node dengan kondisi tekanan disk.

Setelah simpul yang tidak sesuai telah disaring, simpul yang tersisa diberi peringkat menggunakan fungsi prioritas. Beberapa fungsi prioritas yang menonjol dibahas dalam Tabel 9.2.

Tabel 9.2 Fungsi Prioritas

Priority Function	Keterangan
LeastRequestedPriority	Tujuan dari fungsi prioritas ini adalah untuk menyebarkan konsumsi sumber daya di seluruh node. CPU dan memori diberi bobot yang sama dalam menghitung fraksi sumber daya bebas (fraksi node yang akan bebas jika pod dijadwalkan pada node) menggunakan rumus: (kapasitas – jumlah permintaan semua pod yang sudah ada pada node – permintaan pod yang sedang dijadwalkan) / kapasitas). Node dengan fraksi bebas terbesar dipilih untuk penjadwalan.
BalancedResourceAllocation	Tujuan dari fungsi prioritas ini adalah untuk menyeimbangkan tingkat penggunaan CPU dan memori.
SelectorSpreadPriority	Tujuannya adalah untuk menghindari penjadwalan pod dalam pengontrol replikasi, set replika, atau layanan yang sama pada node atau zona yang sama.
CalculateAntiAffinityPriority	Tujuannya adalah untuk menghindari penjadwalan pod dalam layanan yang sama pada node dengan nilai label yang sama untuk label tertentu.
Prioritas Lokalitas Gambar	Tujuannya adalah untuk menjadwalkan pada node yang telah memiliki beberapa atau semua paket gambar yang terpasang. Node dengan ukuran paket yang lebih besar lebih disukai.
Prioritas Afinitas Node	Mengevaluasi afinitas node menggunakan preferredDuringSchedulingIgnoredDuringExecution dan requiredDuringSchedulingIgnoredDuringExecution.

Peringkat simpul akhir dihitung menggunakan skor fungsi prioritas tertimbang. Setiap simpul diberi skor dalam rentang 1–10 untuk setiap fungsi prioritas yang diterapkan, dan skor akhir dihitung dengan menetapkan bobot untuk setiap fungsi prioritas. Misalnya, dengan tiga fungsi prioritas priorityFunc1Score, priorityFunc2Score, dan priorityFunc3Score, skor akhir dihitung sebagai berikut:

```
RankingScoreNodeA = (weight1 * priorityFunc1Score) + (weight2
* priorityFunc2Score) + (weight3 * priorityFunc3Score)
```

Simpul dengan skor tertinggi dipilih untuk menjadwalkan pod. Kebijakan penjadwalan default sebagaimana ditentukan oleh predikat default dan fungsi prioritas dapat disesuaikan atau diganti menggunakan salah satu prosedur berikut:

1. Gunakan parameter --policy-config-file ke penjadwal. File konfigurasi kebijakan adalah file json, misalnya https://github.com/kubernetes/kubernetes/blob/master/examples/scheduler-policy-config.json.
2. Ubah predikat default dan/atau fungsi prioritas di `plugin/pkg/scheduler/algorithm/predicates/predicates.go` dan/atau `plugin/pkg/scheduler/algorithm/priorities/priorities.go` dan daftarkan kebijakan di `defaultPredicates()` dan/atau `defaultPriorities()` di `plugin/pkg/scheduler/algorithmprovider/defaults/defaults.go`.

Menetapkan Lingkungan

Kita akan menggunakan AWS EC2 Cloud Formation berbasis CoreOS untuk menjalankan kluster Kubernetes satu pengontrol tiga pekerja. Jalankan instans EC2 menggunakan Amazon Linux AMI. Masuk ke instans EC2 melalui SSH:

```
ssh -i docker.pem ec2-user@54.197.206.44
```

Mulai konfigurasi cloud untuk kluster Kubernetes dan daftarkan alamat IP Publik pengontrol di nama DNS Publik. Saat mengonfigurasi kluster, tetapkan versi Kubernetes ke `v1.3.0_coreos.1` di kolom kubernetesVersion di cluster.yaml. Instal biner kubectl. Versi Klien dan Server harus 1.3, seperti yang ditunjukkan pada Gambar 9.2.

Gambar 9.2 Mencantumkan Versi Kubernetes

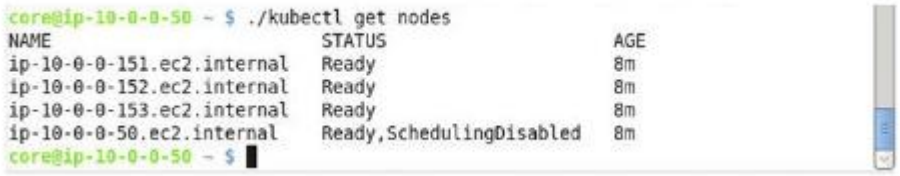
Log masuk SSH ke instans pengontrol:

```
ssh -i "kubernetes-coreos.pem" core@50.19.44.241
```

Mencantumkan node:

```
kubectl get nodes
```

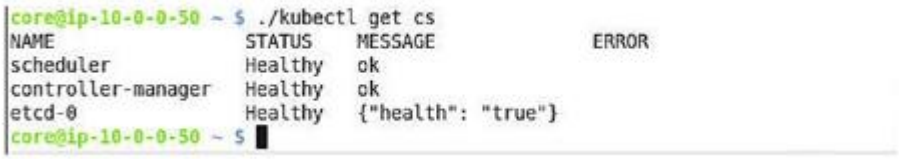
Pengontrol dan tiga node pekerja dicantumkan sebagai berjalan, tetapi node pengontrol tidak dapat dijadwalkan, seperti yang ditunjukkan oleh SchedulingDisabled yang ditunjukkan pada Gambar 9.3.



Gambar 9-.3 Mencantumkan Node; Node Master Tidak Dapat Dijadwalkan

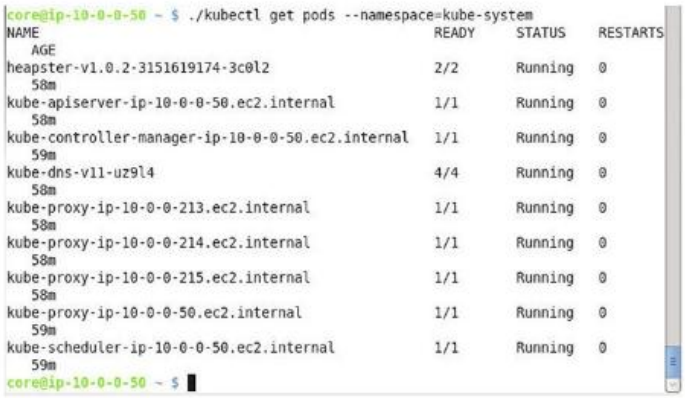
9.2 MENGGUNAKAN PENJADWAL DEFAULT

Penjadwal default kube-scheduler dimulai secara otomatis saat proses Kubernetes (komponen) dimulai. Status komponen harus mencantumkan komponen penjadwal, seperti yang ditunjukkan pada Gambar 9.4.



Gambar 9-4. Mencantumkan Status Komponen Untuk Penjadwal

Pod untuk `kube-scheduler` dimulai di namespace `kube-system`, seperti yang ditunjukkan pada Gambar 9.5.



Gambar 9.5 Daftar Pod Dalam Namespace Kube-System, Termasuk Pod Kube-Scheduler

Perintah `kube-scheduler` dapat digunakan untuk memulai `kube-scheduler` dengan pengaturan khusus. Parameter perintah yang tersedia dapat dicantumkan dengan `kube-scheduler –help` seperti yang ditunjukkan pada Gambar 9.6.

```
core@ip-10-0-0-50 ~ $ ./kube-scheduler -help
Usage of ./kube-scheduler:
      --address=127.0.0.1: The IP address to serve on (set to 0.0.0.0 for all in
terfaces)
      --algorithm-provider="DefaultProvider": The scheduling algorithm provider
to use, one of: DefaultProvider
      --alsologtostderr[=false]: log to standard error as well as files
      --bind-pods-burst=100: Number of bindings per second scheduler is allowed
to make during bursts
      --bind-pods-qps=50: Number of bindings per second scheduler is allowed to
continuously make
      --kubeconfig="": Path to kubeconfig file with authorization and master loc
ation information.
      --log-backtrace-at=:0: when logging hits line file:N, emit a stack trace
      --log-dir="": If non-empty, write log files in this directory
      --log-flush-frequency=5s: Maximum number of seconds between log flushes
      --logtostderr[=true]: log to standard error instead of files
      --master="": The address of the Kubernetes API server (overrides any value
 in kubeconfig)
      --policy-config-file="": File with scheduler policy configuration
      --port=10251: The port that the scheduler's http service runs on
      --profiling[=true]: Enable profiling via web interface host:port/debug/ppr
of/
      --stderrthreshold=2: logs at or above this threshold go to stderr
      --v=0: log level for V logs
      --version=false: Print version information and quit
      --vmodule=: comma-separated list of pattern=N settings for file-filtered l
ogging
core@ip-10-0-0-50 ~ $
```

Gambar 9.6 Penggunaan Perintah Kube-Scheduler

File konfigurasi untuk meluncurkan pod bagi komponen Kubernetes, yang mencakup Server API, Controller Manager, Proxy, dan Scheduler berada di direktori `/etc/kubernetes/manifests` seperti yang ditunjukkan pada Gambar 9.7; `file kube-scheduler.yaml` adalah yang kita perlukan untuk scheduler.

```
core@ip-10-0-0-50 /etc/kubernetes/manifests $ ls -l
total 40
-rw-r--r--. 1 root root  748 Jul 26 16:53 calico-policy-agent.yaml
-rw-r--r--. 1 root root 1466 Jul 26 16:50 kube-apiserver.yaml
-rw-r--r--. 1 root root 1000 Jul 26 16:50 kube-controller-manager.yaml
-rw-r--r--. 1 root root  530 Jul 26 16:50 kube-proxy.yaml
-rw-r--r--. 1 root root  464 Jul 26 16:50 kube-scheduler.yaml
core@ip-10-0-0-50 /etc/kubernetes/manifests $
```

Gambar 9.7 Daftar File Dalam Direktori /Etc/Kubernetes/Manifests

Spesifikasi pod `kube-scheduler` dapat disesuaikan dalam editor vi seperti yang ditunjukkan pada Gambar 9.8.

```
apiVersion: v1
kind: Pod
metadata:
  name: kube-scheduler
  namespace: kube-system
spec:
  hostNetwork: true
  containers:
  - name: kube-scheduler
    image: quay.io/coreos/hyperkube:v1.2.4_coreos.1
    command:
    - /hyperkube
    - scheduler
    - --master=http://127.0.0.1:8080
    - --leader-elect=true
    livenessProbe:
      httpGet:
        host: 127.0.0.1
        path: /healthz
        port: 10251
      initialDelaySeconds: 15
      timeoutSeconds: 1
~
~
~
~
~
~
~
~
~
"kube-scheduler.yaml" 22L, 464C
```

Gambar 9.8 File Kube-Scheduler.Yaml Dalam Editor Vi

Kubelet harus dimulai ulang, seperti yang ditunjukkan pada Gambar 9.9, jika modifikasi pada spesifikasi pod `kube-scheduler` akan berlaku.

```
core@ip-10-0-0-50 ~ $ sudo systemctl restart kubelet
core@ip-10-0-0-50 ~ $ sudo systemctl status kubelet
● kubelet.service
   Loaded: loaded (/etc/systemd/system/kubelet.service; enabled; vendor preset
   Active: active (running) since Wed 2016-07-27 01:39:01 UTC; 15s ago
 Main PID: 21353 (kubelet)
    Tasks: 11
   Memory: 34.8M
      CPU: 4.665s
   CGroup: /system.slice/kubelet.service
           ├─21353 /kubelet --api-servers=http://localhost:8080 --network-plug
           └─21551 journalctl -k -f

Jul 27 01:39:15 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: E0727 01:39:
Jul 27 01:39:15 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: E0727 01:39:
Jul 27 01:39:15 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:15 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: E0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
Jul 27 01:39:16 ip-10-0-0-50.ec2.internal kubelet-wrapper[21353]: W0727 01:39:
lines 1-21/21 (END)
```

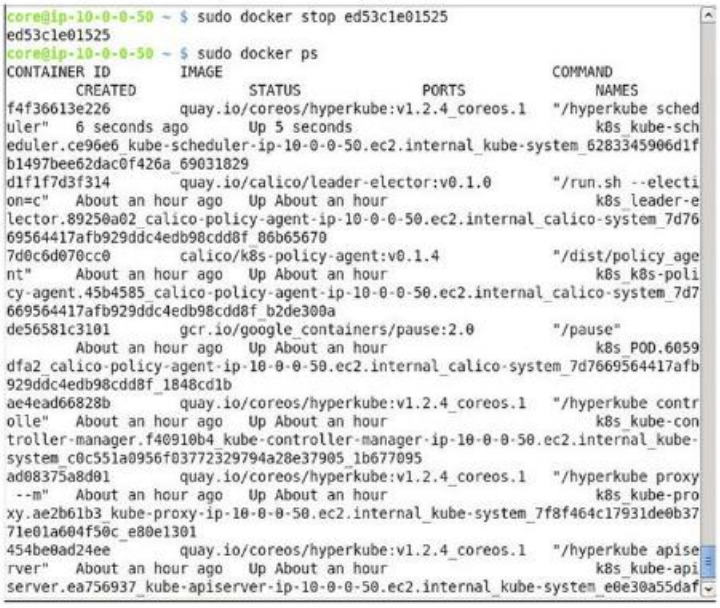
Gambar 9.9 Memulai Ulang Kubelet

Sebuah kontainer dimulai untuk setiap komponen Kubernetes, termasuk penjadwal. Kontainer

dapat dicantumkan dengan perintah docker ps. Kontainer `k8s_kube-schduler` harus dicantumkan seperti yang ditunjukkan pada Gambar 9.10.

Gambar 9.10 Mencantumkan kontainer Docker, termasuk k8s_kube-scheduler

Kontainer penjadwal tidak dapat dihentikan saat kluster Kubernetes sedang berjalan. Jika kontainer penjadwal dihentikan secara eksplisit, kontainer akan dimulai ulang seperti yang ditunjukkan pada Gambar 9.11 oleh kontainer pertama `k8s_kube-schduler` yang dicantumkan dan dimulai 6 detik sebelumnya.

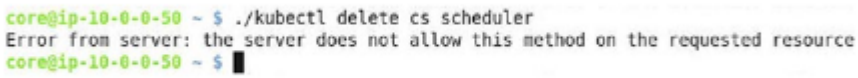


Gambar 9.11 Kontainer k8s_kube-schduler akan dimulai ulang jika dihentikan

Deskripsi pod kube-scheduler, termasuk perintah yang digunakan untuk memulai penjadwal, dapat diperoleh dengan perintah kubectl describe pod seperti yang ditunjukkan pada Gambar 9.12.

Gambar 9.12 Mencantumkan Deskripsi Pod Untuk Kube-Scheduler

Komponen penjadwal tidak dapat dihapus, seperti yang ditunjukkan pada Gambar 9.13.



Gambar 9.13 Komponen Penjadwal Tidak Dihapus

Anotasi `scheduler.alpha.kubernetes.io/name` opsional pada pod dapat digunakan untuk menentukan penjadwal yang akan digunakan. Selanjutnya, kami akan menunjukkan penggunaan anotasi tersebut. Buat file definisi pod bernama `pod1.yaml`:

```
sudo pod1.yaml
```

Pada contoh pertama, kami tidak akan menentukan anotasi `scheduler.alpha.kubernetes.io/name`. Salin daftar berikut ke `pod1.yaml`.

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: pod-without-annotation
  labels:
    name: multischeduler
spec:
  containers:
  - image: "gcr.io/google_containers/pause:2.0"
    name: pod-without-annotation
```

File `pod1.yaml` ditampilkan dalam editor vi pada Gambar 9.14.

Gambar 9.14 Definisi Pod Tanpa Anotasi Penjadwal

Buat pod menggunakan file definisi:

```
./kubectl create -f pod1.yaml
```

Selanjutnya, daftarkan pod:

```
./kubectl get pods -o wide
```

Pod-without-annotation dibuat dan dicantumkan seperti yang ditunjukkan pada Gambar 9.15. Penjadwal default digunakan untuk menjadwalkan pod menggunakan kebijakan penjadwalan default.

Gambar 9.15 Definisi Pod Tanpa Anotasi Penjadwal

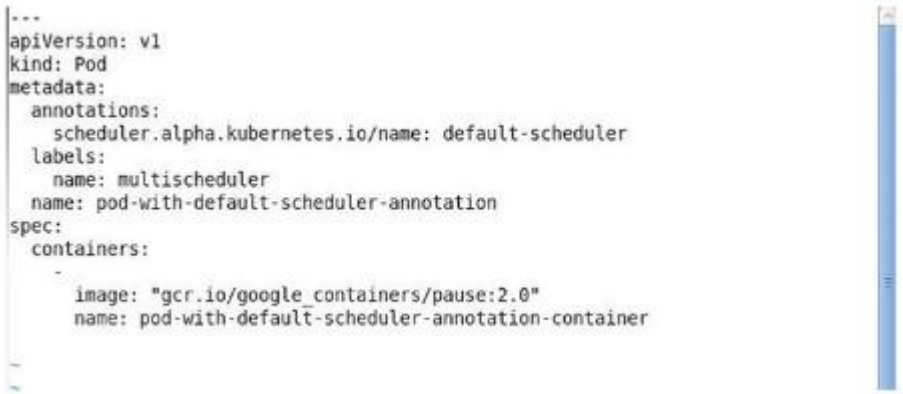
Selanjutnya, kita akan menggunakan anotasi;

`scheduler.alpha.kubernetes.io/name`

dalam file definisi pod. Buat file definisi pod lain, bernama pod2.yaml, dan salin kode berikut ke dalamnya. Anotasi `scheduler.alpha.kubernetes.io/name` ditetapkan ke `default-scheduler` secara eksplisit.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    scheduler.alpha.kubernetes.io/name: default-scheduler
  labels:
    name: multischeduler
    name: default-scheduler # Perhatikan: label ini duplikat,
hanya satu yang akan berlaku
  name: pod-with-default-scheduler-annotation
spec:
  containers:
  - image: "gcr.io/google_containers/pause:2.0"
    name: pod-with-default-scheduler-annotation-container
```

File `pod2.yaml` ditampilkan di editor vi pada Gambar 9.16.



Gambar 9.16 Definisi Pod Dengan Anotasi Penjadwal

Buat pod menggunakan file definisi `pod2.yaml`:

```
./kubectl create -f pod2.yaml
```

Pod `pod-with-default-scheduler-annotation-container` dibuat dan dicantumkan, seperti yang ditunjukkan pada Gambar 9.17.

```
core@ip-10-0-0-50 ~ $ sudo vi pod2.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod2.yaml
pod "pod-with-default-scheduler-annotation" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME                                      READY     STATUS    RESTARTS   AGE
 IP          NODE
pod-with-default-scheduler-annotation     1/1       Running   0          8s
 10.2.54.3   ip-10-0-0-213.ec2.internal
pod-without-annotation                    1/1       Running   0          1m
 10.2.54.2   ip-10-0-0-213.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.17 Membuat Dan Mencantumkan Pod Dengan Anotasi Penjadwal

```
core@ip-10-0-0-50 ~ $ ./kubectl get events
LASTSEEN   FIRSTSEEN   COUNT     NAME                                          KIND
   SUBOBJECT                                                                 TYPE
REASON      SOURCE                                  MESSAGE
59s        59s         1         pod-with-default-scheduler-annotation   Pod
                                                                         Normal
Scheduled   {default-scheduler }                    Successfully assigned pod-wit
h-default-scheduler-annotation to ip-10-0-0-213.ec2.internal
58s        58s         1         pod-with-default-scheduler-annotation   Pod
   spec.containers{pod-with-default-scheduler-annotation-container}   Normal
Pulled      {kubelet ip-10-0-0-213.ec2.internal}   Container image "gcr.io/googl
e_containers/pause:2.0" already present on machine
58s        58s         1         pod-with-default-scheduler-annotation   Pod
   spec.containers{pod-with-default-scheduler-annotation-container}   Normal
Created     {kubelet ip-10-0-0-213.ec2.internal}   Created container with docker
 id 972b50807b73
57s        57s         1         pod-with-default-scheduler-annotation   Pod
   spec.containers{pod-with-default-scheduler-annotation-container}   Normal
Started     {kubelet ip-10-0-0-213.ec2.internal}   Started container with docker
 id 972b50807b73
2m         2m          1         pod-without-annotation                  Pod
                                                                         Normal
Scheduled   {default-scheduler }                    Successfully assigned pod-wit
hout-annotation to ip-10-0-0-213.ec2.internal
2m         2m          1         pod-without-annotation                  Pod
   spec.containers{pod-without-annotation}                              Normal
Pulled      {kubelet ip-10-0-0-213.ec2.internal}   Container image "gcr.io/googl
e_containers/pause:2.0" already present on machine
2m         2m          1         pod-without-annotation                  Pod
   spec.containers{pod-without-annotation}                              Normal
Created     {kubelet ip-10-0-0-213.ec2.internal}   Created container with docker
 id 5d7bb224f843
```

Gambar 9.18 Pod Dijadwalkan Menggunakan Default-Scheduler

Penjadwal default digunakan terlepas dari apakah itu ditetapkan secara eksplisit. Untuk memverifikasi bahwa penjadwal default digunakan, cantumkan Peristiwa. Pod bernama `pod-with-default-scheduler-annotation-container` dicantumkan telah dijadwalkan menggunakan penjadwal default, dan begitu pula pod `pod-without-annotation`, seperti yang ditunjukkan pada Gambar 9.18.

9.3 PENJADWALAN POD TANPA PEMILIH NODE

Kolom nodeSelector dalam spesifikasi pod dapat digunakan untuk memilih node tempat pod dijadwalkan. Kolom nodeSelector menentukan label, yang harus sama dengan label node untuk pod yang akan dijadwalkan pada node tersebut. Jika nodeSelector tidak ditentukan, definisi pod `(pod.yaml)` untuk pod untuk nginx akan mirip dengan berikut ini:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
```

```
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
```

Jalankan pod menggunakan file definisi:

```
kubectl create -f pod.yaml
```

Pod dijadwalkan pada node yang sesuai menggunakan kebijakan penjadwalan default.

Menetapkan Label Node

Selanjutnya, kita akan menggunakan label untuk mencocokkan pod dengan label. Pertama, kita perlu menetapkan label pada node. Nama node tempat label ditetapkan dapat ditemukan dengan kubectl get nodes, seperti yang ditunjukkan pada Gambar 9.3 sebelumnya. Sintaks untuk memberi label pada node adalah sebagai berikut:

```
kubectl label nodes <node-name> <label-key>=<label-value>
```

Beberapa label bawaan juga disediakan, yang juga dapat digunakan di kolom nodeSelector, tetapi hanya satu label yang dapat ditentukan.

```
kubernetes.io/hostname, failure-
domain.beta.kubernetes.io/zone,failure-domain.beta.
kubernetes.io/region,beta.kubernetes.io/instance-type
```

Sebagai contoh, beri label pada node `ip-10-0-0-151.ec2.internal` dengan label `kubernetes.io/image-name=nginx`:

kubectl label nodes `ip-10-0-0-151.ec2.internal kubernetes.io/image-name=nginx`

Demikian pula, beri label pada node `ip-10-0-0-152.ec2.internal`.
kubectl label `nodes ip-10-0-0-152.ec2.internal kubernetes.io/image-name=hello-world`

Node diberi label, seperti yang ditunjukkan pada Gambar 9.19.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-151.ec2.internal kubernete
s.io/image-name=nginx
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-152.ec2.internal kubernete
s.io/image-name=hello-world
node "ip-10-0-0-152.ec2.internal" labeled
core@ip-10-0-0-50 ~ $
```

Gambar 9.19 Memberi Label Pada Node

Buat daftar node, termasuk labelnya, menggunakan argumen perintah `–show-labels` pada perintah `kubectl get nodes`. Label yang ditambahkan dicantumkan sebagai tambahan pada label default, seperti yang ditunjukkan pada Gambar 9.20.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes --show-labels
NAME                       STATUS                     AGE       LABELS
ip-10-0-0-151.ec2.internal   Ready                    11m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-151.ec2.internal,kubernetes.io/image-name=nginx
ip-10-0-0-152.ec2.internal   Ready                    11m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-152.ec2.internal,kubernetes.io/image-name=hello-world
ip-10-0-0-153.ec2.internal   Ready                    11m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-153.ec2.internal
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   11m       kubernetes.io/
hostname=ip-10-0-0-50.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.20 Mencantumkan Node Termasuk Label

Saat menggunakan label untuk mencocokkan pod dengan node, salah satu hasil berikut akan muncul:

1. Pod dijadwalkan pada node berlabel.
2. Pod dijadwalkan pada node tak berlabel jika afinitas node ditentukan.
3. Pod tidak dijadwalkan.

Kita akan membahas masing-masing hal ini di bagian berikut menggunakan node berlabel dan tak berlabel dari bagian ini.

Penjadwalan Pod dengan Pemilih Node

Kolom nodeSelector dalam spesifikasi pod dapat digunakan untuk memilih node secara eksplisit untuk pod. Untuk menetapkan pod ke label, buat file definisi pod pod-nginx.yaml. Salin kode berikut ke file definisi:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
```

```
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    kubernetes.io/image-name: nginx
```

Pod-nginx.yaml yang dihasilkan ditampilkan dalam editor vi pada Gambar 9.21.

Gambar 9.21 Berkas Definisi Pod Pod-Nginx.Yaml

Buat pod menggunakan berkas definisi:

```
kubectl create -f pod-nginx.yaml
```

Demikian pula, buat berkas definisi pod lain pod-helloworld.yaml. Salin daftar berikut ke `pod-helloworld.yaml`:

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-world
  labels:
    env: test
spec:
  containers:
  - name: hello-world
    image: hello-world
    imagePullPolicy: IfNotPresent
  nodeSelector:
    kubernetes.io/image-name: hello-world
```

Buat pod menggunakan file definisi pod:

```
kubectl create -f pod-helloworld.yaml
```

Daftar pod di seluruh klaster:

```
kubectl get pods -o wide
```

Seperti yang ditunjukkan oleh output dari perintah sebelumnya pada Gambar 9.22, kedua pod dibuat dan dimulai. Awalnya pod mungkin tidak berjalan.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-nginx.yaml
pod "nginx" created
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-helloworld.yaml
pod "hello-world" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS              RESTARTS   AGE       IP          NOD
E
hello-world   0/1       CrashLoopBackOff    1          9s        10.2.92.3   ip-
10-0-0-152.ec2.internal
nginx         0/1       ContainerCreating   0          18s       <none>      ip-
10-0-0-151.ec2.internal
```

Gambar 9.22 Membuat Pod Yang Menggunakan Nodeselector

Daftar pod lagi, termasuk node, dan pod harus berjalan atau telah selesai. Kolom Node mencantumkan node tempat pod berjalan, seperti yang ditunjukkan pada Gambar 9.23.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS      RESTARTS   AGE       IP          NODE
hello-world   0/1       Completed   2          22s       10.2.92.3   ip-10-0-0-1
52.ec2.internal
nginx         1/1       Running     0          31s       10.2.17.2   ip-10-0-0-1
51.ec2.internal
```

Gambar 9.23 Mencantumkan Pod Termasuk Node

Dengan menggunakan nama node, dapatkan label untuk masing-masing dari dua node seperti yang ditunjukkan pada Gambar 9.24. Label untuk masing-masing node menyertakan label yang ditentukan dalam nodeSelector untuk pod yang dijadwalkan pada node tersebut.

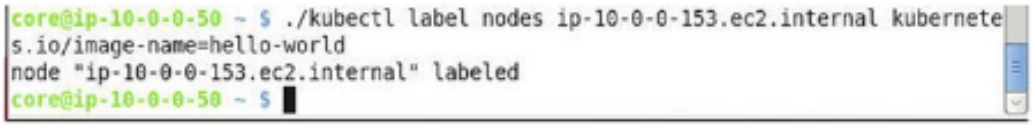
```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes ip-10-0-0-152.ec2.internal --show-labe
ls
NAME                         STATUS    AGE       LABELS
ip-10-0-0-152.ec2.internal   Ready     18m       beta.kubernetes.io/instance-typ
e=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,failure-domain.be
ta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0-0-152.ec2.intern
al,kubernetes.io/image-name=hello-world
core@ip-10-0-0-50 ~ $ ./kubectl get nodes ip-10-0-0-151.ec2.internal --show-labe
ls
NAME                         STATUS    AGE       LABELS
ip-10-0-0-151.ec2.internal   Ready     19m       beta.kubernetes.io/instance-typ
e=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,failure-domain.be
ta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0-0-151.ec2.intern
al,kubernetes.io/image-name=nginx
core@ip-10-0-0-50 ~ $
```

Gambar 9.24 Mencantumkan Label Node

Selanjutnya, kami akan menunjukkan bahwa jika beberapa pod memiliki label yang cocok, salah satu node digunakan. Beri label node ketiga dengan label yang sama dengan salah satu node lainnya:

```
kubectl label nodes ip-10-0-0-153.ec2.internal
kubernetes.io/image-name=hello-world
```

Node ketiga juga diberi label, seperti yang ditunjukkan pada Gambar 9.25.



Gambar 9.25 Memberi Label Pada Node Ketiga

Mencantumkan label node akan menampilkan dua node dengan label umum `kubernetes.io/image-name=hello-world`, seperti yang ditunjukkan pada Gambar 9.26.

Gambar 9.26 Dua Node Dengan Label Yang Sama

Hapus pod hello-world dengan kubectl, karena selanjutnya kita akan membuat pod lagi untuk menemukan node tempat pod dijadwalkan, dengan dua node dengan label yang sama seperti pada kolom nodeSelector. Buat pod hello-world lagi menggunakan berkas definisi yang sama. Cantumkan pod, dan pod tersebut harus ditampilkan pada salah satu dari dua node yang memiliki label `kubernetes.io/image-name=hello-world`, yaitu `ip-10-0-0-152.ec2.internal` dan `ip-10-0-0-153.ec2.internal`. Pod dijadwalkan pada node pertama yang ditemukannya dengan label yang cocok, yaitu `ip-10-0-0-`

152.ec2.internal seperti yang ditunjukkan pada Gambar 9.27.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-helloworld.yaml
pod "hello-world" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS             RESTARTS   AGE       IP          NODE
hello-world   0/1       CrashLoopBackOff   2          33s       10.2.92.3   ip-1
0-0-0-152.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS             RESTARTS   AGE       IP          NODE
hello-world   0/1       CrashLoopBackOff   3          1m        10.2.92.3   ip-1
0-0-0-152.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.27 Pod Dijadwalkan Pada Node Pertama Yang Ditemukannya Dengan Label Yang Cocok

Selanjutnya, kami akan menunjukkan bahwa jika node dengan label yang cocok tidak ditemukan, pod tersebut tidak dijadwalkan sama sekali. Kami perlu menghapus semua label, karena kami akan menggunakan file definisi yang sama untuk pod dan dengan pengaturan kolom nodeSelector yang sama. Hapus label yang ditambahkan sebelumnya ke setiap node:

```
kubectl label nodes ip-10-0-0-151.ec2.internal
kubernetes.io/image-name
kubectl label nodes ip-10-0-0-152.ec2.internal
kubernetes.io/image-name
kubectl label nodes ip-10-0-0-153.ec2.internal
kubernetes.io/image-name
```

Label node dihapus, meskipun output perintah menunjukkan bahwa node telah diberi label, seperti yang ditunjukkan pada Gambar 9.28. Menghapus label node juga dianggap sebagai pelabelan node.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-153.ec2.internal kubernete
s.io/image-name-
node "ip-10-0-0-153.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-152.ec2.internal kubernete
s.io/image-name-
node "ip-10-0-0-152.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-151.ec2.internal kubernete
s.io/image-name-
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $
```

Gambar 9.28 Menghapus Label Node

Cantumkan node, termasuk label, dan label node tidak boleh menyertakan label yang ditambahkan sebelumnya, seperti yang ditunjukkan pada Gambar 9.29.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes --show-labels
NAME                         STATUS                     AGE       LABELS
ip-10-0-0-151.ec2.internal   Ready                      32m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-151.ec2.internal
ip-10-0-0-152.ec2.internal   Ready                      32m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-152.ec2.internal
ip-10-0-0-153.ec2.internal   Ready                      32m       beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-153.ec2.internal
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   32m       kubernetes.io/
hostname=ip-10-0-0-50.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.29 Mencantumkan Label Node Setelah Menghapus Label

Buat dua pod lagi menggunakan file definisi pod yang sama, seperti yang ditunjukkan pada Gambar 9.30.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-nginx.yaml
pod "nginx" created
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-helloworld.yaml
pod "hello-world" created
core@ip-10-0-0-50 ~ $
core@ip-10-0-0-50 ~ $
```

Gambar 9.30 Membuat Pod Menggunakan File Definisi Yang Digunakan Sebelumnya

Cantumkan pod di seluruh klaster. Pod dicantumkan dengan nilai kolom STATUS sebagai Tertunda, seperti yang ditunjukkan pada Gambar 9.31, karena tidak ada node yang memiliki label yang sama seperti yang ditentukan dalam kolom `nodeSelector`.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP        NODE
hello-world   0/1       Pending   0          56s       <none>
nginx         0/1       Pending   0          1m        <none>
core@ip-10-0-0-50 ~ $
```

Gambar 9.31 Pod Dengan Status "Tertunda"

Tambahkan label ke node agar sesuai dengan pengaturan kolom nodeSelector dalam definisi pod, seperti yang ditunjukkan pada Gambar 9.32.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-151.ec2.internal kubernete
s.io/image-name=nginx
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-152.ec2.internal kubernete
s.io/image-name=hello-world
node "ip-10-0-0-152.ec2.internal" labeled
core@ip-10-0-0-50 ~ $
```

Gambar 9.32 Memberi Label Pada Node Agar Sesuai Dengan Label Nodeselector

Kemudian cantumkan pod; pod tidak boleh dalam status Tertunda, telah selesai atau berjalan seperti yang ditunjukkan pada Gambar 9.33. Pod dijadwalkan saat node yang sesuai ditemukan.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS             RESTARTS   AGE       IP          NODE
hello-world   0/1       CrashLoopBackOff   2          2m        10.2.92.3   ip-1
0-0-0-152.ec2.internal
nginx         1/1       Running            0          2m        10.2.17.2   ip-1
0-0-0-151.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.33 Sebelumnya, Pendingpod Dijadwalkan Saat Node Dengan Label Yang Cocok Ditemukan

Jika label node dimodifikasi saat runtime, misalnya jika label dari node dihapus, status pod yang Berjalan tidak berubah menjadi Tertunda dan terus berjalan jika berjalan meskipun node tempat pod berjalan tidak memiliki label yang cocok. Sebagai contoh, hapus label dari node tempat pod nginx berjalan, dan pod terus berjalan seperti yang ditunjukkan pada Gambar 9.34.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-151.ec2.internal kubernete
s.io/image-name-
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS             RESTARTS   AGE       IP          NODE
hello-world   0/1       CrashLoopBackOff   4          4m        10.2.92.3   ip-1
0-0-0-152.ec2.internal
nginx         1/1       Running            0          4m        10.2.17.2   ip-1
0-0-0-151.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.34 Pod Yang Berjalan Terus Berjalan Pada Node Meskipun Label Yang Cocok Dari Node Dihapus

9.4 Menetapkan Afinitas Node

Dimulai dengan versi 1.2, Kubernetes menawarkan versi alfa dari mekanisme baru untuk memilih node, yang disebut afinitas node. Versi alfa afinitas node didasarkan pada label, tetapi dukungan untuk jenis afinitas node lainnya direncanakan akan ditambahkan, seperti penjadwalan pod pada node berdasarkan pod lain yang berjalan pada node tersebut. Saat ini, dua jenis afinitas node didukung seperti yang dibahas dalam Tabel 9.3.

Tabel 9.3 Jenis-jenis Afinitas Node

Afinitas Node	Keterangan
diperlukanSelamaPenjadwalan DiabaikanSelamaEksekusi	Menentukan kondisi afinitas node yang harus dipenuhi. Mirip dengan nodeSelector tetapi deklaratif. IgnoredDuringExecution menyiratkan bahwa persyaratan afinitas node diabaikan setelah pod berjalan. Misalnya, jika label pada node diubah untuk membuat pod yang sedang berjalan tidak dapat dijadwalkan pada node tersebut, pod tersebut akan terus berjalan pada node tersebut. Jika nodeSelector dan nodeAffinity ditetapkan dan nodeAffinity adalah requiredDuringSchedulingIgnoredDuringExecution, keduanya harus dipenuhi agar pod dapat dijadwalkan pada node.
preferedDuringScheduling DiabaikanSelamaEksekusi	Afinitas node yang coba diimplementasikan oleh penjadwal tetapi tidak dijamin. Pod dapat dijadwalkan pada node berlabel tertentu atau tidak berdasarkan pada label yang cocok. Pod bahkan dapat dijadwalkan pada node yang tidak berlabel. Jika nodeAffinity diatur ke preferredDuringSchedulingIgnoredDuringExecution dan tidak ada node yang memenuhi pengaturan, node lain dijadwalkan pada. Jika nodeSelector dan nodeAffinity diatur dan nodeAffinity adalah preferredDuringSchedulingIgnoredDuringExecution, hanya nodeSelector yang harus dipenuhi, karena yang lain hanya merupakan petunjuk untuk preferensi.

Afinitas node dalam versi alfa ditentukan menggunakan anotasi, tetapi ini akan diganti dengan kolom. Contoh pengaturan nodeAffinity requiredDuringSchedulingIgnoredDuringExecution menggunakan anotasi adalah sebagai berikut:

```
annotations:
  scheduler.alpha.kubernetes.io/affinity: >
    {
      "nodeAffinity": {
        "requiredDuringSchedulingIgnoredDuringExecution": {
          "nodeSelectorTerms": [
            {
              "matchExpressions": [
                {
                  "key": "kubernetes.io/image-name",
```

```
                "operator": "In",
                "values": ["image1", "image2"]
              }
            ]
          }
        ]
      }
    }
  another-annotation-key: another-annotation-value
```

Pengaturan another-annotation-key: another-annotation-value menyiratkan bahwa dari node yang ditemukan sesuai dengan kondisi nodeAffinity, node dengan label another-annotation-key: another-annotation-value harus lebih disukai, yang lagi-lagi merupakan petunjuk untuk preferensi yang mungkin atau mungkin tidak diimplementasikan. another-annotation-key: another-annotation-value ditemukan diimplementasikan dengan requiredDuringScheduling IgnoredDuringExecution dan tidak dengan preferredDuringScheduling IgnoredDuringExecution. Selain operator In, operator lain yang didukung adalah NotIn, Exists, DoesNotExist, Gt dan Lt. Selanjutnya, kita akan membahas setiap afinitas node dengan sebuah contoh.

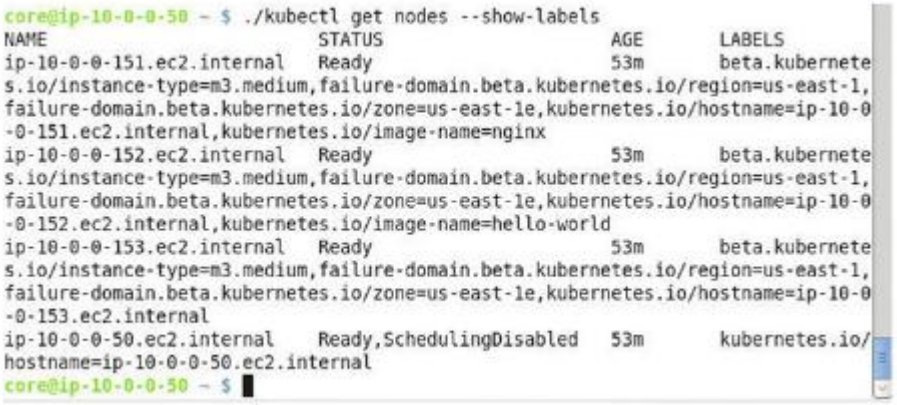
Menyetel requiredDuringSchedulingIgnoredDuringExecution

Buat file definisi pod `pod-node-affinity.yaml` untuk pod bernama with-labels dan tetapkan nodeAffinity ke requiredDuringSchedulingIgnoredDuringExecution dengan ekspresi yang cocok untuk nodeSelectorTerms menjadi label `kubernetes.io/image-name` dengan nilai sebagai salah satu dari `nginx2` atau `hello-world2`. `another-annotation-key: another-annotation-value` adalah `kubernetes.io/image-name: nginx`. Gambar kontainer adalah nginx.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
```

```
                    "values": ["nginx2", "hello-world2"]
                  }
                ]
              }
            ]
          }
        }
      }
  labels:
    kubernetes.io/image-name: nginx # Label ini mungkin tidak
sesuai dengan maksud affinity
spec:
  containers:
  - name: with-labels
    image: nginx
```

Cantumkan node, termasuk label. Label yang dihasilkan pada node tidak boleh menyertakan label yang diperlukan `nginx2` atau `hello-world2`, seperti yang ditunjukkan pada Gambar 9.35.



Gambar 9.35 Tidak Ada Node Yang Memiliki Label Yang Cocok

File `pod-node-affinity.yaml` ditampilkan dalam editor vi pada Gambar 9.36.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx2", "hello-world2"]
                  }
                ]
              }
            ]
          }
        }
      }
#     kubernetes.io/image-name: nginx
spec:
  containers:
  - name: with-labels
    image: nginx
~
```

Gambar 9.36 File Definisi Pod-Node-Affinity.Yaml

Buat pod dari file definisi:

```
kubectl create -f pod-node-affinity.yaml
```

Pod with-labels dibuat seperti yang ditunjukkan pada Gambar 9.37.

```
core@ip-10-0-0-50 ~ $ sudo vi pod-node-affinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-node-affinity.yaml
pod "with-labels" created
```

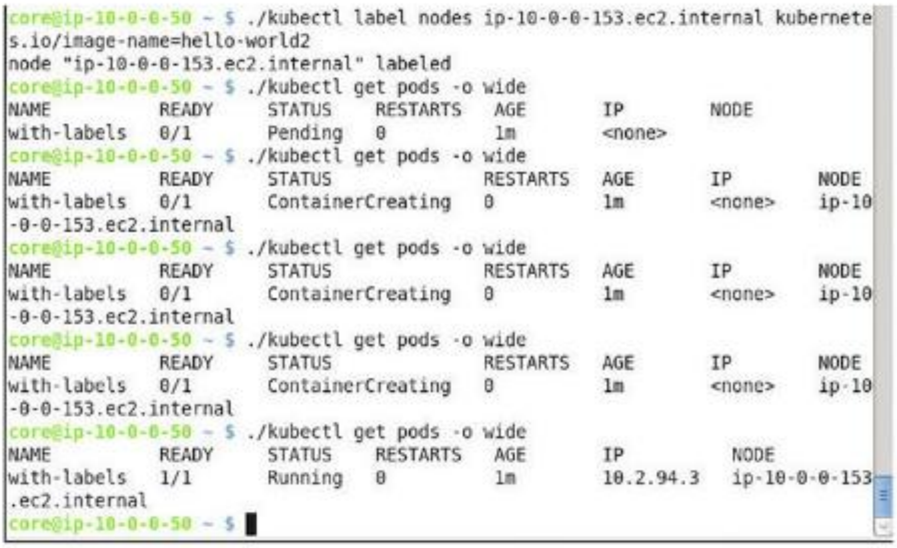
Gambar 9.37 Membuat Pod With-Labels

Cantumkan pod di seluruh kluster. STATUS pod adalah Pending karena tidak ada node yang memiliki label `nginx2` atau `hello-world2`, seperti yang ditunjukkan pada Gambar 9.38.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP        NODE
with-labels   0/1       Pending   0          27s       <none>
core@ip-10-0-0-50 ~ $
```

Gambar 9.38 Mencantumkan Pod Dengan Status Tertunda

Selanjutnya, tambahkan salah satu label yang diperlukan ke salah satu node, misalnya label `hello-world2` ke node `ip-10-0-0-153.ec2.internal`. STATUS pod with-labels berubah dari Tertunda menjadi Berjalan, seperti yang ditunjukkan pada Gambar 9.39.



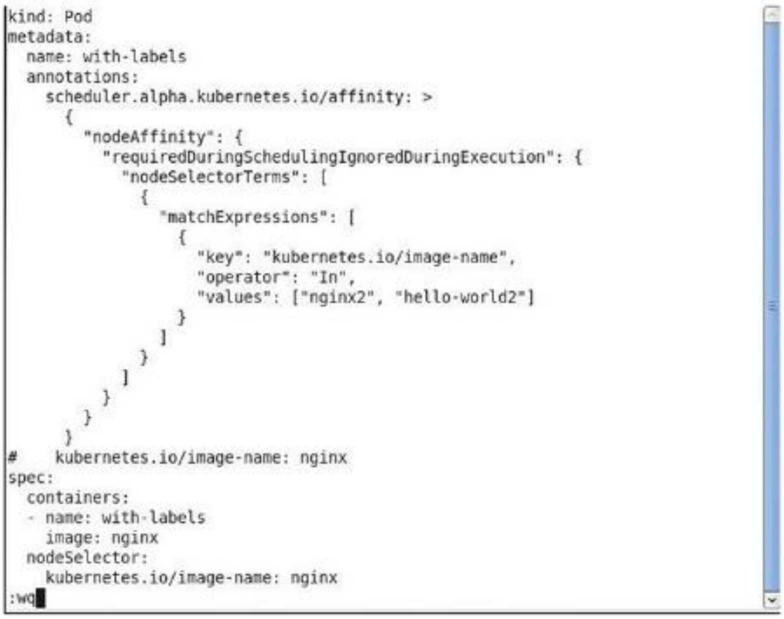
```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-153.ec2.internal kubernete
s.io/image-name=hello-world2
node "ip-10-0-0-153.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP        NODE
with-labels   0/1       Pending   0          1m        <none>
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS              RESTARTS   AGE       IP        NODE
with-labels   0/1       ContainerCreating   0          1m        <none>    ip-10
-0-0-153.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS              RESTARTS   AGE       IP        NODE
with-labels   0/1       ContainerCreating   0          1m        <none>    ip-10
-0-0-153.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS              RESTARTS   AGE       IP        NODE
with-labels   0/1       ContainerCreating   0          1m        <none>    ip-10
-0-0-153.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          1m        10.2.94.3   ip-10-0-0-153
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.39 Status Pod Berubah Dari Tertunda Menjadi Berjalan

Selanjutnya, kami akan menunjukkan bahwa jika nodeAffinity dan nodeSelector ditetapkan dengan nodeAffinity yang ditetapkan ke requiredDuringSchedulingIgnoredDuringExecution, kedua kondisi tersebut harus dipenuhi. Tambahkan label nodeSelector ke `pod-node-affinity.yaml`:

```
nodeSelector:
kubernetes.io/image-name: nginx
```

Pod-node-affinity.yaml yang dimodifikasi ditunjukkan dalam editor vi pada Gambar 9.40.



```
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx2", "hello-world2"]
                  }
                ]
              }
            ]
          }
        }
      }
#     kubernetes.io/image-name: nginx
spec:
  containers:
  - name: with-labels
    image: nginx
  nodeSelector:
    kubernetes.io/image-name: nginx
:wq
```

Gambar 9.40 Menambahkan Nodeselector Sebagai Tambahan Pada Nodeaffinity Yang Ditetapkan Ke Requireddurringschedulingignoreddurringexecution

Kami telah menambahkan label node `kubernetes.io/image-name dengan nilai hello-world2`, tetapi tidak ada node yang memiliki label `kubernetes.io/image-name: nginx`. Saat pod dibuat, pod tersebut dibuat tetapi tidak dijadwalkan, seperti yang ditunjukkan oleh status Pending pada Gambar 9.41.

```
core@ip-10-0-0-50 ~ $ sudo vi pod-node-affinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-node-affinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP        NODE
with-labels   0/1       Pending   0          9s        <none>
core@ip-10-0-0-50 ~ $
```

Gambar 9.41 Pod Dibuat Tetapi Tidak Dijadwalkan

Ubah kolom nodeSelector untuk menentukan label yang ada sebagai tambahan pada label yang diperlukan dari afinitas node. Tambahkan label `kubernetes.io/host-name: ip-10-0-0-151.ec2.internal` seperti yang ditunjukkan pada editor vi pada Gambar 9.42.

```
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx2", "hello-world2"]
                  }
                ]
              }
            ]
          }
        }
      }
#     kubernetes.io/image-name: nginx
spec:
  containers:
  - name: with-labels
    image: nginx
  nodeSelector:
    kubernetes.io/hostname: ip-10-0-0-153.ec2.internal
:wq
```

Gambar 9.42 Menentukan Label Nodeselector Yang Ada

Hapus pod with-labels. Buat pod dengan file definisi pod yang diperbarui. Anda akan melihat bahwa pod dijadwalkan dan berjalan pada host terjadwal seperti yang ditunjukkan pada Gambar 9-43 dengan kondisi nodeSelector dan node affinity terpenuhi.

```
core@ip-10-0-0-50 ~ $ sudo vi pod-node-affinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-node-affinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          12s       10.2.94.3   ip-10-0-0-153
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.43 Baik Nodeselector Maupun Node Affinity Terpenuhi

Selanjutnya, kami akan menunjukkan bahwa jika beberapa nilai label seperti yang ditentukan dalam kolom matchExpressions cocok, node pertama dengan ekspresi yang cocok digunakan. Tambahkan atau timpa label untuk menambahkan `kubernetes.io/image-name:nginx` ke salah satu node dan `kubernetes.io/image-name:hello-world` ke dua dari tiga node, seperti yang ditunjukkan pada Gambar 9.44.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes --overwrite ip-10-0-0-151.ec2.intern
al kubernetes.io/image-name=nginx
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes --overwrite ip-10-0-0-153.ec2.intern
al kubernetes.io/image-name=hello-world
node "ip-10-0-0-153.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl get nodes --show-labels
NAME                         STATUS                     AGE       LABELS
ip-10-0-0-151.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-151.ec2.internal,kubernetes.io/image-name=nginx
ip-10-0-0-152.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-152.ec2.internal,kubernetes.io/image-name=hello-world
ip-10-0-0-153.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-153.ec2.internal,kubernetes.io/image-name=hello-world
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   1h        kubernetes.io/
hostname=ip-10-0-0-50.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.44 Menambahkan Label Ke Node

Ubah `pod-node-affinity.yaml` untuk menambahkan nginx dan hello-world agar ekspresi cocok seperti yang ditunjukkan pada Gambar 9.45.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx", "hello-world"]
                  }
                ]
              }
            ]
          }
        }
      }
#     kubernetes.io/image-name: nginx2
spec:
  containers:
  - name: with-labels
    image: nginx
#  nodeSelector:
#    kubernetes.io/hostname: ip-10-0-0-153.ec2.internal
~
:wq
```

Gambar 9.45 Menetapkan Nilai Label Matchexpressions

Hapus pod dengan label dan buat pod lagi seperti yang ditunjukkan pada Gambar 9.46. Pod dijadwalkan pada node dengan label kubernetes.io/image-name: nginx.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-node-affinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          16s       10.2.17.2   ip-10-0-0-151
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.46 Menjadwalkan Pod Pada Node Pencocokan Pertama

Selanjutnya, kami akan menunjukkan bahwa node berlabel another-annotation-key dengan nilai another-annotation-value lebih disukai jika afinitas node adalah requiredDuringSchedulingIgnoredDuringExecution. Tambahkan atau timpa label node sehingga ada node dengan masing-masing nilai label nginx2 dan hello-world2 untuk kunci kubernetes.io/image-name seperti yang ditunjukkan pada Gambar 9.47.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes --overwrite ip-10-0-0-153.ec2.intern
al kubernetes.io/image-name=hello-world2
node "ip-10-0-0-153.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ kubectl label nodes --overwrite ip-10-0-0-151.ec2.internal
 kubernetes.io/image-name=nginx2
-bash: kubectl: command not found
core@ip-10-0-0-50 ~ $ ./kubectl label nodes --overwrite ip-10-0-0-151.ec2.intern
al kubernetes.io/image-name=nginx2
node "ip-10-0-0-151.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl get nodes --show-labels
NAME                         STATUS                     AGE       LABELS
ip-10-0-0-151.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-151.ec2.internal,kubernetes.io/image-name=nginx2
ip-10-0-0-152.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-152.ec2.internal,kubernetes.io/image-name=hello-world
ip-10-0-0-153.ec2.internal   Ready                      1h        beta.kubernete
s.io/instance-type=m3.medium,failure-domain.beta.kubernetes.io/region=us-east-1,
failure-domain.beta.kubernetes.io/zone=us-east-1e,kubernetes.io/hostname=ip-10-0
-0-153.ec2.internal,kubernetes.io/image-name=hello-world2
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   1h        kubernetes.io/
hostname=ip-10-0-0-50.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.47 Menambahkan Nilai Label Nginx2 Dan Hello-World2 Untuk Kunci Kubernetes.Io/Image-Name

Dalam file `pod-node-affinity.yaml`, tetapkan `another-annotation-key kubernetes.io/image-name ke nginx2` dan beri komentar pada kolom nodeSelector seperti yang ditunjukkan pada Gambar 9.48.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "requiredDuringSchedulingIgnoredDuringExecution": {
            "nodeSelectorTerms": [
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx2", "hello-world2"]
                  }
                ]
              }
            ]
          }
        }
    kubernetes.io/image-name: nginx2
spec:
  containers:
  - name: with-labels
    image: nginx
#  nodeSelector:
#    kubernetes.io/hostname: ip-10-0-0-153.ec2.internal
~
:wq
```

Gambar 9.48 Menetapkan Another-Annotation-Key Dan Menghapus Nodeselector

Hapus pod with-labels dan buat pod lagi. Pod dijadwalkan pada node dengan label kubernetes.io/image-name: nginx2, seperti yang ditunjukkan oleh NODE dalam daftar pod pada Gambar 9.49.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete pod with-labels
pod "with-labels" deleted
core@ip-10-0-0-50 ~ $ sudo vi pod-node-affinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-node-affinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          7s        10.2.17.2   ip-10-0-0-151
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.49 Pod Dijadwalkan Pada Node Bernilai Another-Annotation-Key

9.5 PENGATURAN PREFERREDDURINGSCHEDULINGIGNOREDDURINGEXECUTION

Pada bagian ini, kita akan menggunakan afinitas node preferredDuringSchedulingIgnoredDuringExecution, yang hanya merupakan petunjuk bagi penjadwal dan tidak dijamin. Seperangkat nilai node yang sedikit berbeda digunakan untuk contoh, seperti yang ditunjukkan pada Gambar 9.50.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-222.ec2.internal    Ready                      22m
ip-10-0-0-223.ec2.internal    Ready                      22m
ip-10-0-0-224.ec2.internal    Ready                      22m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   22m
core@ip-10-0-0-50 ~ $
```

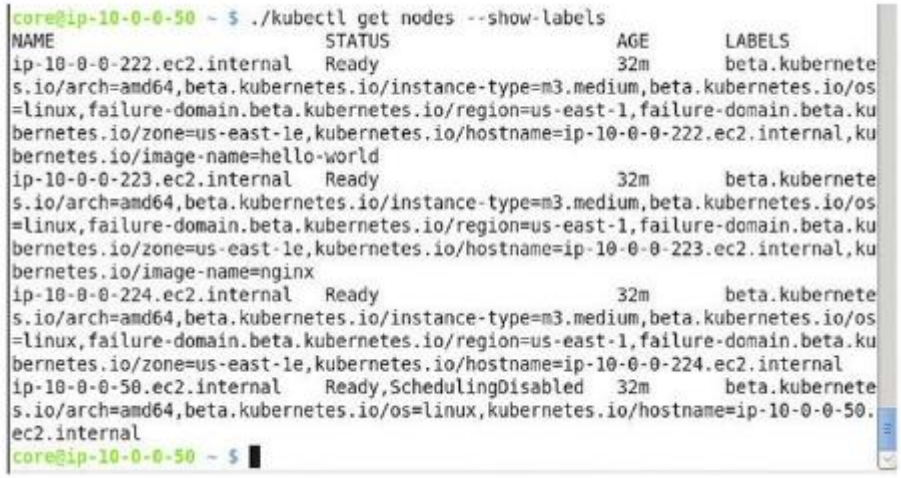
Gambar 9.50 Mencantumkan Node Yang Digunakan Untuk Contoh Afinitas Node Preferredduringschedulingignoredduringexecution

Tetapkan kunci label kubernetes.io/image-name ke nginx pada salah satu node dan hello-world pada node lain seperti yang ditunjukkan pada Gambar 9.51. Node ketiga dibiarkan tidak berlabel.

```
core@ip-10-0-0-50 ~ $ ./kubectl label nodes --overwrite ip-10-0-0-222.ec2.intern
al kubernetes.io/image-name=hello-world
node "ip-10-0-0-222.ec2.internal" labeled
core@ip-10-0-0-50 ~ $ ./kubectl label nodes ip-10-0-0-223.ec2.internal kubernete
s.io/image-name=nginx
node "ip-10-0-0-223.ec2.internal" labeled
core@ip-10-0-0-50 ~ $
```

Gambar 9.51 Menetapkan Label Node

Daftar label untuk setiap node seperti yang ditunjukkan pada Gambar 9.52.



Gambar 9.52 Mencantumkan Label Node

Seperti yang dibahas sebelumnya, NodeAffinity adalah fungsi prioritas; dan fungsi prioritas memiliki bobot yang dialokasikan untuknya dalam pemeringkatan node. Buat file definisi pod podNodeAffinity.yaml dan alokasikan bobot 75 untuk pod menggunakan afinitas node preferredDuringSchedulingIgnoredDuringExecution. Tetapkan ekspresi agar sesuai dengan kunci label `kubernetes.io/image-name menjadi nginx atau hello-world`.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference": {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx", "hello-world"]
                  }
                ]
              }
            }
          ]
        }
      }
```

```
  labels:
    kubernetes.io/image-name: hello-world
spec:
  containers:
  - name: with-labels
    image: nginx
```

Berkas definisi pod `podNodeAffinity.yaml` ditampilkan dalam editor vi pada Gambar 9.53.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference":
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx", "hello-world"]
                  }
                ]
              }
            }
          ]
        }
      }
#    kubernetes.io/image-name: hello-world
spec:
  containers:
  - name: with-labels
    image: nginx

:wq
```

Gambar 9.53 Berkas Definisi Pod Podnodeaffinity.Yaml

Buat pod with-labels menggunakan berkas definisi pod. Daftarkan pod di seluruh klaster. Pod with-labels dijadwalkan pada node dengan label kubernetes.io/image-name: nginx, seperti yang ditunjukkan pada Gambar 9.54. Kebijakan penjadwalan tidak hanya membentuk fungsi prioritas, dan afinitas node bukanlah satu-satunya fungsi prioritas; dan dengan afinitas node yang lunak, pod dapat dialokasikan ke node acak atau alokasi dapat didasarkan pada hasil perhitungan skor fungsi prioritas.

```
core@ip-10-0-0-50 ~ $ sudo vi podNodeAffinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f podNodeAffinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          26s       10.2.45.2   ip-10-0-0-223
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.54 Penjadwalan Pod Menggunakan Afinitas Node

Preferredduringschedulingignoredduringexecution

Tambahkan `another-annotation-key: another-annotation-value` sebagai `kubernetes.io/image-name: hello-world` seperti yang ditunjukkan pada Gambar 9.55.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference":
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["nginx", "hello-world"]
                  }
                ]
              }
            }
          ]
        }
      }
    kubernetes.io/image-name: hello-world
spec:
  containers:
  - name: with-labels
    image: nginx

:wq
```

Gambar 9.55 Menambahkan Another-Annotation-Key: Another-Annotation-Value

Hapus pod with-label dan buat pod lagi seperti yang ditunjukkan pada Gambar 9.56. Pod dijadwalkan lagi pada node dengan label kubernetes.io/image-name: nginx.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f podNodeAffinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          8s        10.2.45.2   ip-10-0-0-223
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.56 Menjadwalkan Pod Dengan Afinitas Node

Afinitas node preferredDuringSchedulingIgnoredDuringExecution hanyalah petunjuk. Untuk menunjukkannya, tetapkan semua opsi untuk kunci label kubernetes.io/image-name ke hello-world, baik dalam ekspresi In maupun dalam anotasi another seperti yang ditunjukkan pada Gambar 9.57.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference":
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["hello-world", "hello-world"]
                  }
                ]
              }
            }
          ]
        }
      }
    kubernetes.io/image-name: hello-world
spec:
  containers:
  - name: with-labels
    image: nginx

:wq
```

Gambar 9.57 Menetapkan Semua Nilai Label Ke Nginx

Hapus dan buat pod lagi. Pod dijadwalkan pada node dengan kubernetes. Kunci label io/image-name ditetapkan ke hello-world, seperti yang ditunjukkan pada Gambar 9.58. Sekali lagi penjadwal tidak menjamin alokasi pod ke node dengan label yang ditentukan saat afinitas node adalah preferredDuringSchedulingIgnoredDuringExecution. Dengan pengaturan yang sama, pod dapat dialokasikan ke node yang berbeda.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f podNodeAffinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS              RESTARTS   AGE       IP        NODE
with-labels   0/1       ContainerCreating   0          8s        <none>    ip-10
-0-0-222.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP           NODE
with-labels   1/1       Running   0          24s       10.2.100.3   ip-10-0-0-22
2.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.58 Penjadwalan Pod Dengan Afinitas Node Preferredduringschedulingignoredduringexecution Tidak Menjamin Penjadwalan Pod Pada Node Tertentu

Sebagai contoh lain, tentukan semua nilai kunci kubernetes.io/image-name ke nilai yang tidak

digunakan dalam label node, seperti yang ditunjukkan pada Gambar 9.59.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference":
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["helloworld", "nginx2"]
                  }
                ]
              }
            }
          ]
        }
      }
    kubernetes.io/image-name: helloworld
spec:
  containers:
  - name: with-labels
    image: nginx

:wq
```

Gambar 9.59 Menetapkan Semua Nilai Kunci Kubernetes.Io/Image-Name Ke Nilai Yang Tidak Ada

Hapus pod dengan label dan buat pod lagi. Pod masih dijadwalkan meskipun tidak ada node yang memiliki label yang cocok, seperti yang ditunjukkan pada Gambar 9-60. Sebagai perbandingan, ketika kami menggunakan afinitas node requiredDuringSchedulingIgnoredDuringExecution tanpa ada node yang memiliki label yang cocok, pod ditempatkan dalam status Tertunda hingga label yang cocok ditambahkan. Sekarang pod dijadwalkan karena pengaturan preferredDuringSchedulingIgnoredDuringExecution tidak mengikat dan hanya berupa petunjuk.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f podNodeAffinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          6s        10.2.45.2   ip-10-0-0-223
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.60 Pod Dijadwalkan Meskipun Tidak Ada Node Dengan Label Yang Cocok Yang Ditemukan

Kolom nodeSelector jika ditentukan dengan afinitas node preferredDuringSchedulingIgnoredDuringExecution masih dijamin. Tambahkan kolom nodeSelector dengan label kubernetes.io/image-name: nginx seperti yang ditunjukkan pada Gambar 9.61. Semua ekspresi yang cocok lainnya ditetapkan ke kubernetes.io/image-name: hello-world.

```
metadata:
  name: with-labels
  annotations:
    scheduler.alpha.kubernetes.io/affinity: >
      {
        "nodeAffinity": {
          "preferredDuringSchedulingIgnoredDuringExecution": [
            {
              "weight": 75,
              "preference":
              {
                "matchExpressions": [
                  {
                    "key": "kubernetes.io/image-name",
                    "operator": "In",
                    "values": ["hello-world", "hello-world"]
                  }
                ]
              }
            }
          ]
        }
      }
    kubernetes.io/image-name: hello-world
spec:
  containers:
  - name: with-labels
    image: nginx
  nodeSelector:
    kubernetes.io/image-name: nginx

:wq
```

Gambar 9.61 Menetapkan Nodeselector Sebagai Tambahan Pada Afinitas Node Preferredduringschedulingignoredduringexecution

Hapus dan buat pod with-labels lagi. Pod dijadwalkan pada node dengan label kubernetes.io/image-name: nginx karena ekspresi nodeSelector adalah kubernetes.io/image-name: nginx seperti yang ditunjukkan pada Gambar 9.62.

```
core@ip-10-0-0-50 ~ $ sudo vi podNodeAffinity.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f podNodeAffinity.yaml
pod "with-labels" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME          READY     STATUS    RESTARTS   AGE       IP          NODE
with-labels   1/1       Running   0          21s       10.2.45.2   ip-10-0-0-223
.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 9.62 Pod Dijadwalkan Pada Node Dengan Label Yang Cocok Dengan Ekspresi Nodeselector

Ringkasan

Pada bab ini, pertama-tama kita membahas kebijakan penjadwalan default yang digunakan oleh Kubernetes. Kemudian, kita menggunakan penjadwal default dan juga pemilih node untuk menjadwalkan pod pada node. Kita juga membahas penjadwalan pod menggunakan afinitas node. Pada bab berikutnya, kita akan membahas konfigurasi sumber daya komputasi.

BAB 10
MENGONFIGURASI SUMBER DAYA KOMPUTASI

10.1 PENGELOLAAN SUMBER DAYA FLEKSIBEL DI KUBERNETES

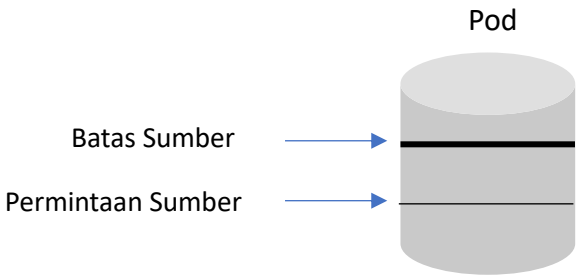
Model sumber daya Kubernetes sederhana, teratur, dapat diperluas, dan tepat. Pengelola kluster kontainer Kubernetes menyediakan dua jenis sumber daya: sumber daya komputasi dan sumber daya API. Sumber daya komputasi yang didukung (disebut "sumber daya" dalam bab ini) adalah CPU dan RAM (atau memori). Dukungan untuk sumber daya komputasi lainnya, seperti lebar pita jaringan, operasi jaringan, ruang penyimpanan, operasi penyimpanan, dan waktu penyimpanan dapat ditambahkan kemudian.

Masalah

Kapasitas node Kubernetes dalam hal sumber daya yang dapat dialokasikan (CPU dan memori) bersifat tetap dan harus dibagi di antara berbagai pod yang berjalan pada node tersebut. Pod juga memiliki beberapa persyaratan tetap untuk sumber daya (CPU dan memori) dengan beberapa fleksibilitas dalam konsumsi sumber daya. Masalah dalam penggunaan sumber daya adalah bagaimana mengalokasikan sumber daya ke berbagai pod dan juga menambahkan beberapa fleksibilitas agar pod dapat menggunakan lebih dari sumber daya minimum yang diminta jika tersedia.

Solusi

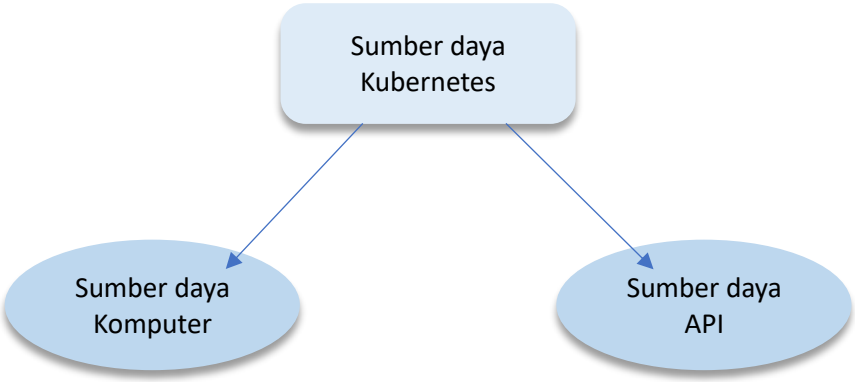
Kubernetes menyediakan pola desain penggunaan sumber daya yang fleksibel berdasarkan permintaan dan batasan seperti yang ditunjukkan pada Gambar 10-1. Permintaan adalah sumber daya minimum (CPU dan memori) yang diminta kontainer dalam pod agar dapat dijadwalkan dan dijalankan pada node. Batasan adalah sumber daya maksimum (CPU dan memori) yang dapat dialokasikan ke kontainer.



Gambar 10.1 Permintaan dan batasan sumber daya Kubernetes

Ikhtisar

Dua jenis sumber daya, sumber daya komputasi dan sumber daya API, ditunjukkan pada Gambar 10.2. Sumber daya komputasi adalah kuantitas terukur yang dapat diminta oleh kontainer dalam pod, dialokasikan ke kontainer, dan dikonsumsi oleh kontainer. Sumber daya API adalah objek Kubernetes seperti pod dan layanan, yang ditulis ke dan diambil dari server API.



Gambar 10.2. Jenis sumber daya Kubernetes

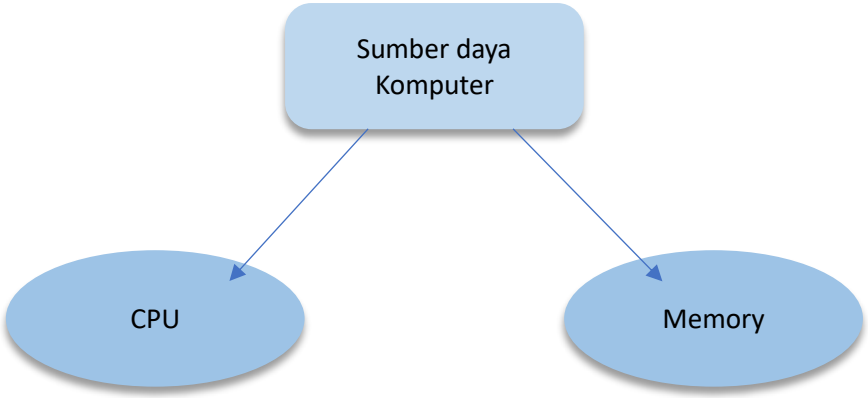
Kami hanya akan membahas sumber daya komputasi dalam bab ini. Secara default, sumber daya komputasi yang tersedia untuk kontainer atau pod hanya dibatasi oleh kapasitas node. Sementara sumber daya dikonsumsi oleh kontainer dalam pod (pod dapat memiliki satu atau lebih kontainer), sumber daya juga tersirat dikonsumsi oleh pod. Sumber daya yang diminta, dialokasikan, dan dikonsumsi oleh pod adalah total sumber daya yang diminta, dialokasikan, dan dikonsumsi oleh kontainer dalam pod. Kapasitas node terdiri dari sumber daya yang tersedia untuk node dalam hal CPU, memori, dan jumlah maksimum pod yang dapat dijadwalkan pada node.

Total semua sumber daya yang dialokasikan untuk setiap sumber daya (CPU, memori) untuk kontainer yang berjalan pada node tidak dapat melebihi kapasitas node untuk sumber daya tersebut. Penjadwal Kubernetes memastikan bahwa sumber daya yang cukup tersedia pada node sebelum menjadwalkan pod pada node. Bahkan setelah menjadwalkan node, Penjadwal memastikan bahwa total sumber daya yang dialokasikan pada node tidak melebihi kapasitas node, yang tidak dapat dilakukan berdasarkan kapasitas node. Scheduler hanya memantau kontainer yang dimulai oleh kubelet dan bukan kontainer yang dimulai oleh mesin Docker. Bab ini membahas topik-topik berikut:

Jenis sumber daya komputasi
Permintaan dan batasan sumber daya
Kualitas layanan
Menetapkan lingkungan
Menemukan kapasitas node
Membuat pod dengan sumber daya yang ditentukan
Melebihi batas sumber daya
Memesan sumber daya node

10.2 JENIS SUMBER DAYA KOMPUTASI

Kubernetes menyediakan dua jenis sumber daya komputasi, CPU dan memori, seperti yang ditunjukkan pada Gambar 10.3.



Gambar 10.3. Jenis sumber daya komputasi

Sumber daya komputasi juga disebut sebagai "sumber daya" dan setiap jenis sumber daya adalah kuantitas yang dapat diukur secara jelas di tingkat kontainer. Secara default, sumber daya komputasi pod tidak terbatas, hanya dibatasi oleh kapasitas node. Kontainer pod secara opsional dapat menentukan permintaan sumber daya dan tingkat batas untuk setiap jenis sumber daya, seperti yang dibahas di bagian berikutnya. Menentukan nilai eksplisit untuk permintaan dan batas sumber daya direkomendasikan karena alasan berikut:

- Melakukan hal tersebut memudahkan Penjadwal untuk menetapkan pod pada node.
- Menyelesaikan penjadwal memudahkan Penjadwal untuk menangani kelebihan sumber daya dan perebutan sumber daya.
- Kapasitas node yang tersedia dapat membuat beberapa pod tidak dapat dijadwalkan pada node jika kontainer pod memerlukan kapasitas lebih besar daripada kapasitas node.
- Menentukan permintaan dan batas sumber daya memungkinkan desain penjadwalan pod yang lebih baik.
- Ruang nama terpisah dapat dibuat untuk beban kerja pengembangan dan produksi dengan permintaan dan batas sumber daya terpisah, konsumsi sumber daya berbeda untuk berbagai jenis beban kerja. Pemanfaatan sumber daya kluster yang lebih efisien dapat dilakukan dengan menentukan permintaan dan batasan sumber daya secara eksplisit. Meskipun melampaui kapasitas node merupakan salah satu masalah, node dapat kurang dimanfaatkan jika pod yang hanya menggunakan sebagian kecil kapasitas node dijadwalkan pada node tersebut, dengan kapasitas node yang tersisa tidak sesuai untuk menjadwalkan pod lain maupun digunakan oleh pod yang dijadwalkan.

CPU adalah siklus prosesor dan diukur dalam satuan inti CPU (atau hanya "CPU"); satu inti CPU dapat berupa satu AWS vCPU, satu inti GCP, satu Azure vCore, atau satu hyperthread pada prosesor Intel bare-metal dengan hyperthreading. Nilai 1 di bidang cpu adalah 1000 millicpu. Nilai tersebut dapat berupa kelipatan 1 seperti 2, 3, atau 4. Nilai CPU fraksional juga dapat

ditentukan dan diterjemahkan ke x1000 millicpu. Misalnya, nilai .001 adalah 1 millicpu, yang merupakan nilai terendah yang dapat ditentukan; presisi yang lebih baik tidak memungkinkan.

Satuan bidang memori adalah byte dan dapat ditetapkan sebagai bilangan bulat biasa seperti 134217728 atau 135e6, bilangan bulat titik tetap dengan sufiks SI seperti 135M, atau kelipatan biner byte yang setara dengan 128Mi.

Permintaan dan Batasan Sumber Daya

Selain batasan yang diberlakukan oleh kapasitas node, kontainer dapat meminta sejumlah sumber daya tertentu dan juga memberlakukan batasan pada jumlah maksimum sumber daya yang dapat digunakan kontainer. Ini disebut permintaan dan batasan kontainer. Permintaan kontainer adalah jumlah sumber daya yang dijamin kontainer; penjadwal tidak akan menetapkan pod pada node jika node tidak dapat menyediakan total permintaan kontainer untuk setiap jenis sumber daya. Batas kontainer adalah jumlah maksimum sumber daya yang diizinkan sistem untuk digunakan kontainer. Sementara total permintaan sumber daya yang dialokasikan tidak dapat melebihi batas kapasitas node untuk sumber daya tersebut, total batasan sumber daya dapat melebihi batas kapasitas node—dengan asumsi bahwa setiap kontainer pada node tidak akan menggunakan batas sumber daya maksimum secara bersamaan.

Ketika total batasan sumber daya dari semua pod yang berjalan pada node melebihi kapasitas node, node tersebut dikatakan terlalu banyak digunakan. Setiap kontainer dapat melebihi sumber daya yang dijamin yang dialokasikan kepadanya melalui permintaan sumber daya, hingga batas sumber daya selama total konsumsi sumber daya pada node tidak melebihi kapasitas node. Namun jika karena perebutan sumber daya, total konsumsi sumber daya oleh kontainer pada suatu node melebihi kapasitas node, atau cenderung melebihi kapasitas node, beberapa pod mungkin harus dihentikan; dan jika restartPolicy diatur ke Selalu, pod dapat dimulai ulang.

Jaminan sumber daya dapat dikompresi atau tidak dapat dikompresi. Jaminan sumber daya CPU dapat dikompresi dan jaminan sumber daya memori tidak dapat dikompresi. Jaminan sumber daya CPU yang dapat dikompresi menyiratkan bahwa pod atau lebih khusus kontainer dibatasi jika melebihi batas CPU-nya. Sebuah kontainer dapat dibatasi kembali ke level CPU yang dijamin jika kelebihan memori yang dialokasikan untuknya diminta oleh proses lain seperti pod yang baru dimulai atau tugas sistem atau daemon. Jika CPU tambahan tersedia setelah semua pod pada node telah dialokasikan, CPU minimum yang diminta (dijamin) dan tugas sistem dan daemon mendapatkan CPU yang mereka butuhkan, CPU tambahan didistribusikan di antara pod dalam proporsi permintaan CPU minimum mereka (CPU yang dijamin). Misalnya, jika sebuah node memiliki tiga pod dengan satu dialokasikan CPU terjamin sebesar 150m, yang kedua CPU terjamin sebesar 300m, dan yang ketiga CPU terjamin sebesar 450m, CPU tambahan didistribusikan dalam proporsi yang sama 1:2:3 hingga batas setiap kontainer.

Sumber daya CPU adalah sumber daya elastis yang dialokasikan dalam rentang jaminan permintaan minimum dan batas sumber daya. Jaminan sumber daya memori bersifat

elastis hanya dalam satu arah; kontainer atau pod dapat menggunakan lebih banyak memori daripada minimum yang diminta (terjamin) hingga batasnya, tetapi jika kontainer menggunakan lebih dari memori tingkat permintaan, pod dapat dihentikan jika pod lain yang menggunakan kurang dari tingkat minimum yang dijamin mulai menggunakan lebih banyak memori atau jika tugas sistem atau daemon meminta lebih banyak memori. Kontainer yang menggunakan kurang dari dan hingga tingkat permintaan memori terjamin tidak akan pernah dihentikan kecuali beberapa tugas sistem atau daemon telah meminta lebih banyak memori.

Dan kontainer yang menggunakan lebih banyak memori daripada batas dihentikan terlepas dari ketersediaan memori berlebih. Saat merujuk pada kapasitas node, yang tersirat adalah node yang dapat dialokasikan, karena beberapa sumber daya harus dicadangkan untuk komponen sistem dan komponen Kubernetes. Permintaan dan batasan sumber daya menentukan rentang 0 <= permintaan <= Node yang Dapat Dialokasikan dan permintaan <= batas. Spesifikasi Pod menyediakan kolom yang ditunjukkan pada Tabel 10.1 untuk permintaan dan batasan sumber daya.

Tabel 10.1. Bidang Spesifikasi Pod untuk Sumber Daya Komputasi

Bidang Spesifikasi Pod	Deskripsi
`spec.container[].resources.requests.cpu`	Sumber daya CPU yang diminta oleh kontainer. Kontainer dijamin mendapatkan CPU yang diminta. Penjadwal menjadwalkan pod berdasarkan CPU yang diminta dan CPU yang tersedia pada node. Defaultnya adalah spec.container[].resources.limits. cpu jika tidak ditentukan.
`spec.container[].resources.requests.memory`	Sumber daya memori yang diminta oleh kontainer. Kontainer dijamin mendapatkan memori yang diminta. Scheduler menjadwalkan pod berdasarkan memori yang diminta dan memori yang tersedia pada node. Defaultnya adalah spec.container[].resources. limits.memory jika tidak ditentukan.
`spec.container[].resources.limits.cpu`	Batas atas CPU yang dapat digunakan kontainer. Penjadwal tidak mempertimbangkan batas CPU. Nilai spec.container[].resources.limits. cpu harus lebih besar atau sama

	dengan spec. container[].resources.requests.cpu. Secara default kapasitas node yang dapat dialokasikan.
spec.container[].resources.limits.memory	Batas atas memori yang dapat digunakan kontainer. Penjadwal tidak mempertimbangkan batas memori. Nilai spec.container[].resources. limits.memory harus lebih besar dari atau sama dengan spec.container[].resources.requests. memory. Secara default sesuai kapasitas node yang dapat dialokasikan.

Menentukan kolom sumber daya bersifat opsional; jika tidak ditetapkan, nilainya dapat ditetapkan ke 0 atau nilai default dan penerapannya bervariasi sesuai konfigurasi kluster. Berikut ini adalah beberapa contoh pengaturan kolom untuk CPU dan memori:

```
containers:
- name: db
  image: mysql
  resources:
   requests:
     memory: "64Mi"
     cpu: ".1"
   limits:
     memory: "128Mi"
     cpu: ".5"

containers:
  - name: db
    image: mysql
    resources:


    requests:
     memory: "64Mi"
     cpu: "100m"
    limits:
     memory: "64Mi"
     cpu: "500m"
```

```
containers:
- name: db
  image: mysql
  resources:
     requests:
       memory: "1Gi"
       cpu: "250m"
 limits:
      memory: "2Gi"
      cpu: "250m"
```

Permintaan dan batasan diterapkan pada perintah Docker run saat memulai kontainer seperti yang ditunjukkan pada Tabel 10.2.

Tabel 10.2. Opsi Perintah Docker run yang Setara untuk Bidang Spesifikasi Pod

Bidang Spesifikasi	Opsi Perintah Docker run	Deskripsi
`spec.container[].resources.requests.cpu`	`--cpu-shares`	CPU berbagi CPU
`spec.container[].resources.limits.cpu`	`--cpu-quota`	Menetapkan kuota CPU CFS (Completely Fair Scheduler)
`spec.container[].resources.limits.memori`	`--memory flag`	Batas memori

10.3 KUALITAS LAYANAN

Kualitas Layanan (QoS) Kubernetes adalah level untuk ketersediaan sumber daya. Pod atau kontainer dalam pod yang membutuhkan level sumber daya minimum dapat meminta sumber daya terjamin dengan kolom spec.container[].resources.requests.cpu dan spec.container[].resources.requests.memory. Pod yang tidak membutuhkan sumber daya terjamin dapat mengabaikan penentuan level permintaan. Tiga kelas QoS disediakan untuk kontainer untuk setiap jenis sumber daya. Kelas QoS didasarkan pada permintaan dan batasan dan seperti yang ditunjukkan pada Tabel 10-3 dalam urutan prioritas menurun.

Tabel 10.3. Kelas QoS

Kelas QoS	Keterangan
Terjamin	Batasan dan permintaan opsional (tidak sama dengan 0) ditetapkan untuk semua sumber daya di semua kontainer dan semuanya sama. Permintaan secara default ke batasan jika tidak ditetapkan. Ini

	adalah pod dengan prioritas tertinggi dan tidak dihentikan (karena memori) atau dibatasi (karena CPU) kecuali tugas sistem atau daemon meminta sumber daya dan pod dengan prioritas lebih rendah tidak tersedia.
Bisa meledak	Permintaan dan batasan opsional (tidak sama dengan 0) ditetapkan untuk satu atau beberapa sumber daya di satu atau beberapa kontainer dan keduanya tidak sama. Pod ini memiliki prioritas menengah dan memiliki beberapa tingkat jaminan sumber daya. Jika CPU diperlukan oleh pod atau sistem dengan prioritas lebih tinggi dan tidak ada pod Best-Effort yang berjalan, CPU pod dapat dibatasi. Demikian pula, jika memori diperlukan oleh pod atau sistem dengan prioritas lebih tinggi dan tidak ada pod Best-Effort yang berjalan, pod dapat dihentikan.
Upaya Terbaik	Permintaan dan batasan tidak ditetapkan untuk sumber daya apa pun untuk kontainer mana pun. Pod ini adalah pod dengan prioritas terendah dan dapat dihentikan jika sumber daya memori dibutuhkan oleh pod lain dengan prioritas lebih tinggi atau tugas sistem atau daemon membutuhkan memori. CPU dapat dibatasi jika dibutuhkan oleh pod dan sistem lain.

Kebijakan QoS mengasumsikan bahwa swap dinonaktifkan.

Menetapkan Lingkungan

Buat klaster Kubernetes sebagai AWS CloudFormation dengan CoreOS Linux. Pertama, buat instans AWS EC2 dari Amazon Linux AMI. Masuk ke instans EC2 melalui SSH.

```
ssh -i "docker.pem" ec2-user@174.129.50.31
```

Luncurkan CloudFormation untuk klaster Kubernetes dengan satu node pengontrol dan tiga node pekerja.

Instal biner kubectl dan daftarkan node:

```
./kubectl get nodes
```

Node dalam klaster Kubernetes tercantum, seperti yang ditunjukkan pada Gambar 10-4.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                         STATUS                     AGE
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   2m
ip-10-0-0-63.ec2.internal    Ready                      2m
ip-10-0-0-64.ec2.internal    Ready                      2m
ip-10-0-0-65.ec2.internal    Ready                      2m
core@ip-10-0-0-50 ~ $
```

Gambar 10.4. Kluster node Kubernetes

Menemukan Kapasitas Node

Kapasitas node dapat ditemukan dengan mendeskripsikan node tersebut. Misalnya:

```
kubectl describe node ip-10-0-0-50.ec2.internal
```

```
Addresses:      10.0.0.64,10.0.0.64,54.243.23.193
Capacity:
 cpu:           1
 memory:        3857824Ki
 pods:          110
Allocatable:
 cpu:           1
 memory:        3857824Ki
 pods:          110
```

Gambar 10.5. Kapasitas node, total dan yang dapat dialokasikan

Kolom Capacity mencantumkan kapasitas node dalam hal CPU, memori, dan jumlah pod. Kolom Allocatable mencantumkan CPU, memori, dan jumlah pod yang dapat dialokasikan seperti yang ditunjukkan pada Gambar 10-5.

Permintaan dan Batas CPU dan Memori termasuk sumber daya yang dialokasikan juga tercantum tetapi awalnya semuanya harus 0 jika tidak ada pod yang berjalan pada node tersebut, seperti yang ditunjukkan pada Gambar 10-6.

```
Non-terminated Pods:          (5 in total)
  Namespace                   Name                                          C
PU Requests     CPU Limits    Memory Requests Memory Limits
  ---------     ----------    --------------- -------------
  calico-system               calico-policy-agent-ip-10-0-0-50.ec2.internal 0
 (0%)           0 (0%)        0 (0%)          0 (0%)
  kube-system                 kube-apiserver-ip-10-0-0-50.ec2.internal      0
 (0%)           0 (0%)        0 (0%)          0 (0%)
  kube-system                 kube-controller-manager-ip-10-0-0-50.ec2.intern
al               0 (0%)        0 (0%)          0 (0%)          0 (0%)
  kube-system                 kube-proxy-ip-10-0-0-50.ec2.internal          0
 (0%)           0 (0%)        0 (0%)          0 (0%)
  kube-system                 kube-scheduler-ip-10-0-0-50.ec2.internal      0
 (0%)           0 (0%)        0 (0%)          0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted. More info: http:/
/releases.k8s.io/HEAD/docs/user-guide/compute-resources.md)
  CPU Requests  CPU Limits    Memory Requests Memory Limits
  ------------  ----------    --------------- -------------
  0 (0%)        0 (0%)        0 (0%)          0 (0%)
No events.

core@ip-10-0-0-50 ~ $
```

Gambar 10.6. Permintaan dan batas CPU dan memori

```
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted. More info: http://
releases.k8s.io/HEAD/docs/user-guide/compute-resources.md)
  CPU Requests  CPU Limits    Memory Requests Memory Limits
  ------------  ----------    --------------- -------------
  0 (0%)        0 (0%)        0 (0%)          0 (0%)
Events:
  FirstSeen     LastSeen      Count   From                                  S
ubobjectPath    Type          Reason                  Message
  ---------     --------      -----   ----                                  -
------------    --------      ------                  -------
  11m           11m           1       {kubelet ip-10-0-0-50.ec2.internal}   N
ormal          Starting                Starting kubelet.
  11m           11m           1       {kubelet ip-10-0-0-50.ec2.internal}   N
ormal          NodeNotSchedulable      Node ip-10-0-0-50.ec2.internal status is
 now: NodeNotSchedulable
  10m           10m           1       {kube-proxy ip-10-0-0-50.ec2.internal} N
ormal          Starting                Starting kube-proxy.
```

Gambar 10.7 Node pengontrol tidak dapat dijadwalkan

Untuk node pengontrol, deskripsi node harus selalu mencantumkan sumber daya yang dialokasikan sebagai 0 karena node tersebut tidak dapat dijadwalkan, seperti yang ditunjukkan oleh NodeNotSchedulable di kolom Type pada Gambar 10.7.

10.4 MEMBUAT POD DENGAN SUMBER DAYA YANG DITENTUKAN

Di bagian ini, kita akan membuat contoh pod dengan permintaan dan batasan sumber daya yang ditentukan untuk kontainer. Buat file definisi mysql.yaml menggunakan citra Docker mysql untuk pengontrol replikasi. Tentukan permintaan dan batasan sumber daya kontainer. Jenis sumber daya yang sama hanya dapat ditentukan satu kali dalam daftar.

--

```
apiVersion: v1
kind: ReplicationController
metadata:
   name: mysql-v1
   labels:
   app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
template:
 metadata:
  labels:
   app: mysql-app
   deployment: v1
spec:
 containers:
-
env:
 -
    name: MYSQL_ROOT_PASSWORD
    value: mysql
image: mysql
name: mysql

ports:
-
  containerPort: 3306
resources:
  requests:
   memory: "64Mi"
   cpu: "250m"
limits:
  memory: "128Mi"
  cpu: "500m"
```

Berkas definisi mysql.yaml ditunjukkan dalam editor vi pada Gambar 10.8.

```
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              value: mysql
          image: mysql
          name: mysql
          ports:
            -
              containerPort: 3306
          resources:
            requests:
              memory: "64Mi"
              cpu: "250m"
            limits:
              memory: "128Mi"
              cpu: "500m"
:wq
```

Gambar 10.8. Berkas definisi pengontrol replikasi mysql.yaml

Buat pengontrol replikasi menggunakan berkas definisi:

```
./kubectl create -f mysql.yaml
```

Daftar pod di seluruh kluster:

```
./kubectl get pods
```

Awalnya pod mungkin tidak berjalan atau Siap. Daftarkan pod setelah satu menit, dan semua pod seharusnya berjalan. Setiap pod dijadwalkan pada node yang berbeda, seperti yang ditunjukkan pada Gambar 10.9.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
replicationcontroller "mysql-v1" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-v1-80tj3   1/1       Running   0          13s       10.2.56.3   ip-10-0-0
-63.ec2.internal
mysql-v1-kn53w   1/1       Running   0          13s       10.2.83.2   ip-10-0-0
-64.ec2.internal
mysql-v1-pfd3r   1/1       Running   0          13s       10.2.39.3   ip-10-0-0
-65.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 10.9. Setiap pod dijadwalkan pada node yang berbeda

Jelaskan sebuah node untuk menemukan konsumsi sumber daya pada node tersebut, seperti yang ditunjukkan pada Gambar 10-10. Hanya satu pod yang berjalan pada node tersebut. Permintaan dan Batas CPU dan Memori untuk setiap pod dalam namespace default dicantumkan. Permintaan CPU pod MySQL sebesar 250m dan Batas CPU sebesar 500m dan Permintaan Memori sebesar 64Mi dan Batas Memori sebesar 128 Mi dicantumkan. Permintaan dan Batas CPU dan Memori yang dialokasikan juga dicantumkan.

Permintaan CPU dan Memori yang dialokasikan kurang dari batas, yang merupakan level yang diinginkan.

```
Non-terminated Pods:          (3 in total)
  Namespace                   Name                                    C
PU Requests     CPU Limits    Memory Requests Memory Limits
  ---------                   ----                                    -
-----------     ----------    --------------- -------------
  default                     mysql-v1-80tj3                          2
50m (25%)       500m (50%)    64Mi (1%)       128Mi (3%)
  kube-system                 kube-dns-v11-uzc57                      3
10m (31%)       310m (31%)    170Mi (4%)      920Mi (24%)
  kube-system                 kube-proxy-ip-10-0-0-63.ec2.internal    0
 (0%)           0 (0%)        0 (0%)          0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted. More info: http:/
/releases.k8s.io/HEAD/docs/user-guide/compute-resources.md)
  CPU Requests  CPU Limits    Memory Requests Memory Limits
  ------------  ----------    --------------- -------------
  560m (56%)    810m (81%)    234Mi (6%)      1048Mi (27%)
No events.

core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME             READY     STATUS    RESTARTS   AGE
mysql-v1-80tj3   1/1       Running   2          1m
mysql-v1-kn53w   1/1       Running   2          1m
mysql-v1-pfd3r   1/1       Running   2          1m
core@ip-10-0-0-50 ~ $
```

Gambar 10.10. Permintaan dan batas CPU dan memori pada node yang dapat dijadwalkan

Jelaskan pod yang dimulai dengan pengaturan yang telah dibahas sebelumnya, dan Batasan dan Permintaan untuk jenis sumber daya harus dicantumkan seperti yang ditunjukkan pada Gambar 10.11 (nama pod dapat berasal dari proses yang berbeda dengan pengaturan yang sama).

```
core@ip-10-0-0-50 ~ $ ./kubectl describe pod mysql-v1-f3j7k
Name:           mysql-v1-f3j7k
Namespace:      default
Node:           ip-10-0-0-63.ec2.internal/10.0.0.63
Start Time:     Mon, 11 Jul 2016 16:35:02 +0000
Labels:         app=mysql-app
                deployment=v1
Status:         Running
IP:             10.2.56.3
Controllers:    ReplicationController/mysql-v1
Containers:
  mysql:
    Container ID:       docker://5c5df522ab85de6eb02ddb1d153d7743878a21444065915
3e16136eff9ce3240
    Image:              mysql
    Image ID:           docker://sha256:1195b21c3a45d9bf93aae497f2538f89a09aaded
18d6648753aa3ce76670f41d
    Port:               3306/TCP
    Limits:
      cpu:      500m
      memory:   128Mi
    Requests:
      cpu:              250m
      memory:           64Mi
    State:              Running
```

Gambar 10.11. Permintaan dan batasan CPU dan memori Pod

Batas sumber daya harus ditetapkan lebih tinggi daripada permintaan. Sebagai contoh, tetapkan batas agar lebih rendah daripada permintaan, seperti yang ditunjukkan pada Gambar 10.12.

```
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              value: mysql
          image: mysql
          name: mysql
          ports:
            -
              containerPort: 3306
          resources:
            requests:
              memory: "64Mi"
              cpu: "250m"
            limits:
              memory: "60Mi"
              cpu: "200m"
~
:wq
```

Gambar 10.12. Permintaan CPU dan memori Pod ditetapkan lebih tinggi dari batas

Saat pod dibuat, muncul kesalahan yang menunjukkan bahwa batas CPU dan memori harus lebih tinggi dari permintaan, seperti yang ditunjukkan pada Gambar 10.13.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete rc mysql-v1
replicationcontroller "mysql-v1" deleted
core@ip-10-0-0-50 ~ $ sudo vi mysql.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
The ReplicationController "mysql-v1" is invalid.

* spec.template.spec.containers[0].resources.limits[cpu]: Invalid value: "200m":
 must be greater than or equal to request
* spec.template.spec.containers[0].resources.limits[memory]: Invalid value: "60M
i": must be greater than or equal to request
core@ip-10-0-0-50 ~ $
```

Gambar 10.13. Kesalahan yang menunjukkan nilai tidak valid untuk batas sumber daya

CPU dapat ditetapkan sebagai pecahan (misalnya 0,3) alih-alih nilai absolut, seperti yang ditunjukkan pada Gambar 10.14. Nilai CPU sebesar 0,3 adalah 300m. Permintaan sama dengan batasan dalam contoh.

```
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              value: mysql
          image: mysql
          name: mysql
          ports:
            -
              containerPort: 3306
          resources:
            requests:
              memory: "385782Ki"
              cpu: "0.3"
            limits:
              memory: "3857824Ki"
              cpu: "0.3"

:wq
```

Gambar 10.14. CPU ditetapkan sebagai pecahan

Buat pengontrol replikasi dan daftarkan pod. Ketiga replika dijadwalkan pada tiga node yang berbeda. Skala RC menjadi enam pod. Keenam pod dijadwalkan pada tiga node dengan dua pod pada setiap node, seperti yang ditunjukkan pada Gambar 10.15.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale --replicas=6 rc/mysql-v1
replicationcontroller "mysql-v1" scaled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-v1-c8aed   1/1       Running   0          23m       10.2.56.3   ip-10-0-0
-63.ec2.internal
mysql-v1-ckywi   1/1       Running   0          7m        10.2.56.4   ip-10-0-0
-63.ec2.internal
mysql-v1-gl2r4   1/1       Running   0          23m       10.2.39.3   ip-10-0-0
-65.ec2.internal
mysql-v1-kgp2t   1/1       Running   0          7m        10.2.83.2   ip-10-0-0
-64.ec2.internal
mysql-v1-q9nep   1/1       Running   0          2m        10.2.39.4   ip-10-0-0
-65.ec2.internal
mysql-v1-qins5   1/1       Running   0          23m       10.2.83.3   ip-10-0-0
-64.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 10.15. Dua pod dijadwalkan pada setiap node

Jumlah replika pod memiliki batas, karena kapasitas sumber daya per node dapat mulai didekati dengan lebih banyak replika. Dengan pengaturan sebelumnya, tujuh replika dijadwalkan, seperti yang ditunjukkan pada Gambar 10.16. Salah satu node memiliki tiga pod.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale --replicas=7 rc/mysql-v1
replicationcontroller "mysql-v1" scaled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-v1-b3ucj   1/1       Running   0          55s       10.2.83.4   ip-10-0-0
-64.ec2.internal
mysql-v1-c8aed   1/1       Running   0          26m       10.2.56.3   ip-10-0-0
-63.ec2.internal
mysql-v1-ckywi   1/1       Running   0          10m       10.2.56.4   ip-10-0-0
-63.ec2.internal
mysql-v1-gl2r4   1/1       Running   0          26m       10.2.39.3   ip-10-0-0
-65.ec2.internal
mysql-v1-kgp2t   1/1       Running   0          10m       10.2.83.2   ip-10-0-0
-64.ec2.internal
mysql-v1-q9nep   1/1       Running   0          5m        10.2.39.4   ip-10-0-0
-65.ec2.internal
mysql-v1-qins5   1/1       Running   0          26m       10.2.83.3   ip-10-0-0
-64.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 10.16. Kapasitas kluster Kubernetes memiliki batas, yang memungkinkan tujuh pod dijadwalkan pada tiga node dalam contoh ini

Jelaskan node dengan tiga pod, dan Anda akan melihat bahwa konsumsi sumber daya berada pada 90% untuk CPU dan memori, seperti yang ditunjukkan pada Gambar 10.17. Tidak ada lagi pod yang dapat dijadwalkan pada pod tersebut.

```
Non-terminated Pods:          (4 in total)
  Namespace                   Name                                        C
PU Requests     CPU Limits    Memory Requests Memory Limits
  ---------                   ----                                        -
-----------     ----------    --------------- -------------
  default                     mysql-v1-b3ucj                              3
00m (30%)       300m (30%)    385782Ki (9%)   3857824Ki (100%)
  default                     mysql-v1-kgp2t                              3
00m (30%)       300m (30%)    385782Ki (9%)   3857824Ki (100%)
  default                     mysql-v1-qins5                              3
00m (30%)       300m (30%)    385782Ki (9%)   3857824Ki (100%)
  kube-system                 kube-proxy-ip-10-0-0-64.ec2.internal        0
 (0%)           0 (0%)        0 (0%)          0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted. More info: http:/
/releases.k8s.io/HEAD/docs/user-guide/compute-resources.md)
  CPU Requests  CPU Limits    Memory Requests Memory Limits
  ------------  ----------    --------------- -------------
  900m (90%)    900m (90%)    1157346Ki (29%) 11573472Ki (300%)
No events.

core@ip-10-0-0-50 ~ $
```

Gambar 10.17. Konsumsi sumber daya berada pada 90%

Batas memori minimum yang diizinkan adalah 4 MB.

10.5 BATAS JUMLAH POD

Jumlah pod yang dapat dijadwalkan pada sebuah node dibatasi oleh kapasitas node yang dapat dialokasikan, yang mencakup batas 110 pada jumlah pod. Untuk mendemonstrasikannya, skalakan RC menjadi 400 pod. RC diskalakan menjadi 400 replika dan tidak ada kesalahan yang dihasilkan, seperti yang ditunjukkan pada Gambar 10.18.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale --replicas=400 rc/mysql-v1
replicationcontroller "mysql-v1" scaled
core@ip-10-0-0-50 ~ $
```

Gambar 10.18. Penskalaan menjadi 400 node

Namun, 400 pod tidak dapat berjalan pada tiga node; ini dibatasi tidak hanya oleh batas jumlah pod per node tetapi juga oleh CPU dan memori yang dapat dialokasikan. Pod yang tidak dapat berjalan dimasukkan ke dalam status Tertunda dan tidak ada node yang dialokasikan untuknya, seperti yang ditunjukkan pada Gambar 10.19.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-v1-02ptf    0/1       Pending   0          30s       <none>
mysql-v1-076yo    0/1       Pending   0          35s       <none>
mysql-v1-087wk    0/1       Pending   0          40s       <none>
mysql-v1-08dwx    0/1       Pending   0          30s       <none>
mysql-v1-0a5s5    0/1       Pending   0          37s       <none>
mysql-v1-0dngp    0/1       Pending   0          46s       <none>
mysql-v1-0ilqr    0/1       Pending   0          33s       <none>
mysql-v1-0l0w4    0/1       Pending   0          32s       <none>
mysql-v1-0rbl5    0/1       Pending   0          44s       <none>
mysql-v1-0rgui    0/1       Pending   0          33s       <none>
mysql-v1-0w8ht    0/1       Pending   0          36s       <none>
mysql-v1-16sf9    0/1       Pending   0          44s       <none>
mysql-v1-183yv    0/1       Pending   0          34s       <none>
mysql-v1-1dfzl    0/1       Pending   0          30s       <none>
mysql-v1-1elp2    0/1       Pending   0          44s       <none>
mysql-v1-1ffkn    0/1       Pending   0          30s       <none>
mysql-v1-1mh9d    0/1       Pending   0          38s       <none>
mysql-v1-1ofg5    0/1       Pending   0          47s       <none>
mysql-v1-1ojz6    0/1       Pending   0          31s       <none>
mysql-v1-1pl9f    0/1       Pending   0          34s       <none>
mysql-v1-1pswb    0/1       Pending   0          34s       <none>
mysql-v1-1uqyb    0/1       Pending   0          47s       <none>
mysql-v1-1x88g    0/1       Pending   0          45s       <none>
mysql-v1-22gkg    0/1       Pending   0          39s       <none>
mysql-v1-23gx5    0/1       Pending   0          34s       <none>
mysql-v1-27ol8    0/1       Pending   0          34s       <none>
mysql-v1-2839x    0/1       Pending   0          34s       <none>
mysql-v1-2acfx    0/1       Pending   0          38s       <none>
mysql-v1-2i4cl    0/1       Pending   0          41s       <none>
mysql-v1-2ixqd    0/1       Pending   0          32s       <none>
```

Gambar 10.19. Pod dimasukkan ke dalam status Tertunda

Skalakan RC kembali menjadi lima replika. Pod Tertunda tidak langsung dihapus. Namun, lima pod yang berjalan dicantumkan, seperti yang ditunjukkan pada Gambar 10-20.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale --replicas=5 rc/mysql-v1
replicationcontroller "mysql-v1" scaled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME              READY     STATUS        RESTARTS   AGE       IP          NODE
mysql-v1-087wk    0/1       Pending       0          3m        <none>
mysql-v1-2i4cl    0/1       Pending       0          3m        <none>
mysql-v1-2n792    0/1       Pending       0          3m        <none>
mysql-v1-2s3rd    0/1       Pending       0          3m        <none>
mysql-v1-2tnkn    0/1       Pending       0          3m        <none>
mysql-v1-38f0p    0/1       Pending       0          3m        <none>
mysql-v1-3aiy6    0/1       Pending       0          3m        <none>
mysql-v1-3lxxb    0/1       Pending       0          3m        <none>
mysql-v1-5ldcn    0/1       Pending       0          3m        <none>
mysql-v1-60nlf    0/1       Pending       0          3m        <none>
mysql-v1-630i9    0/1       Pending       0          3m        <none>
mysql-v1-72psi    0/1       Pending       0          3m        <none>
mysql-v1-7hgki    0/1       Pending       0          3m        <none>
mysql-v1-8go2u    0/1       Pending       0          3m        <none>
mysql-v1-8k0o2    0/1       Pending       0          3m        <none>
mysql-v1-8umyf    0/1       Pending       0          3m        <none>
mysql-v1-bb5fs    0/1       Pending       0          3m        <none>
mysql-v1-bz5s7    0/1       Pending       0          3m        <none>
mysql-v1-c8aed    1/1       Running       0          19m       10.2.56.3   ip-10
-0-0-63.ec2.internal
mysql-v1-cad66    0/1       Pending       0          3m        <none>
mysql-v1-cc5aw    0/1       Pending       0          3m        <none>
mysql-v1-cfszj    0/1       Pending       0          3m        <none>
mysql-v1-ckywi    1/1       Running       0          3m        10.2.56.4   ip-10
-0-0-63.ec2.internal
mysql-v1-cqcf5    0/1       Pending       0          3m        <none>
```

Gambar 10.20. Pod yang dimasukkan ke dalam status pending

Skalakan RC ke 10 replika. Hanya tujuh pod yang Berjalan dan yang lainnya Tertunda, seperti yang ditunjukkan pada Gambar 10.21.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale --replicas=10 rc/mysql-v1
replicationcontroller "mysql-v1" scaled
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME             READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-v1-3n0tv   0/1       Pending   0          10s       <none>
mysql-v1-c8aed   1/1       Running   0          21m       10.2.56.3   ip-10-0-0
-63.ec2.internal
mysql-v1-ckywi   1/1       Running   0          5m        10.2.56.4   ip-10-0-0
-63.ec2.internal
mysql-v1-cmkod   0/1       Pending   0          10s       <none>
mysql-v1-gl2r4   1/1       Running   0          21m       10.2.39.3   ip-10-0-0
-65.ec2.internal
mysql-v1-kgp2t   1/1       Running   0          5m        10.2.83.2   ip-10-0-0
-64.ec2.internal
mysql-v1-q9nep   1/1       Running   0          10s       10.2.39.4   ip-10-0-0
-65.ec2.internal
mysql-v1-qins5   1/1       Running   0          21m       10.2.83.3   ip-10-0-0
-64.ec2.internal
mysql-v1-u51zs   1/1       Running   0          10s       10.2.83.4   ip-10-0-0
-64.ec2.internal
mysql-v1-wek22   0/1       Pending   0          10s       <none>
core@ip-10-0-0-50 ~ $
```

Gambar 10.21. Penskalaan ke 10 replika hanya memiliki tujuh dari 10 pod yang Berjalan

Batasan Sumber Daya yang Terlalu Banyak Dikomit

Seperti yang ditunjukkan dalam contoh sebelumnya, maksimal tiga pod dapat dijadwalkan pada sebuah node dengan permintaan yang dialokasikan menghabiskan 90% dari CPU yang dapat dialokasikan. Batasan memori berada pada 300%, yang membuat batasan tersebut menjadi terlalu banyak dikomit. Jika semua pod meminta memori maksimum yang dapat dialokasikan secara bersamaan, konsumsi sumber daya akan melebihi 100% dan beberapa pod akan berhenti. Bahkan dengan satu pod pada sebuah node, batas memori akan terlalu banyak dikomit sebesar 109%, meskipun tidak terlalu banyak, seperti yang ditunjukkan pada Gambar 10-22.

```
Non-terminated Pods:          (3 in total)
  Namespace                   Name                                    C
PU Requests     CPU Limits    Memory Requests Memory Limits
  ---------                   ----                                    -
-----------     ----------    --------------- -------------
  default                     mysql-v1-gl2r4                          3
00m (30%)       300m (30%)    385782Ki (9%)   3857824Ki (100%)
  kube-system                 heapster-v1.0.2-3151619174-5dqrl        1
50m (15%)       150m (15%)    366Mi (9%)      366Mi (9%)
  kube-system                 kube-proxy-ip-10-0-0-65.ec2.internal    0
 (0%)           0 (0%)        0 (0%)          0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted. More info: http:/
/releases.k8s.io/HEAD/docs/user-guide/compute-resources.md)
  CPU Requests  CPU Limits    Memory Requests Memory Limits
  ------------  ----------    --------------- -------------
  450m (45%)    450m (45%)    760566Ki (19%)  4232608Ki (109%)
```

Gambar 10.22. Batasan memori yang terlalu banyak dikomit

Menyimpan Sumber Daya Node

Pod bukan satu-satunya objek atau proses yang menghabiskan sumber daya pada sebuah node. Mungkin cocok untuk mencadangkan beberapa sumber daya untuk proses non-pod seperti proses sistem. Sumber daya dapat dicadangkan dengan menjalankan pod placeholder. Buat file definisi pod pod-reserve-resource.yaml. Jalankan citra Docker gcr.io/google_containers/pause dan tentukan batas sumber daya untuk sumber daya yang akan dicadangkan seperti 200m untuk CPU dan 200Mi untuk memori.

```
apiVersion: v1
kind: Pod
metadata:
     name: reserve-resource
spec:
   containers:
   - name: reserve-resource
     image: gcr.io/google_containers/pause:0.8.0
     resources:
      limits:
        cpu: “0.1”
        memory: 200Mi
```

Berkas definisi pod ditunjukkan dalam editor vi pada Gambar 10.23.

```
apiVersion: v1
kind: Pod
metadata:
  name: reserve-resource
spec:
  containers:
  - name: reserve-resource
    image: gcr.io/google_containers/pause:0.8.0
    resources:
      limits:
        cpu: ".1"
        memory: "200Mi"
~
```

Gambar 10.23. Definisi pod untuk memesan beberapa sumber daya

Pertama, buat pod placeholder seperti yang ditunjukkan pada Gambar 10.24. Kemudian buat MySQL RC.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete rc mysql-v1
replicationcontroller "mysql-v1" deleted
core@ip-10-0-0-50 ~ $ ls -l
total 55212
-rwxr-xr-x 1 root root 56515944 Jul  1 20:06 kubectl
-rw-r--r-- 1 root root      685 Jul 11 18:13 mysql.yaml
-rw-r--r-- 1 root root      226 Jul 11 18:18 pod-reserve-resource.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod-reserve-resource.yaml
pod "reserve-resource" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
reserve-resource   1/1       Running   0          10s
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
replicationcontroller "mysql-v1" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
mysql-v1-c8aed     1/1       Running   0          7s
mysql-v1-gl2r4     1/1       Running   0          7s
mysql-v1-qins5     1/1       Running   0          7s
reserve-resource   1/1       Running   0          39s
core@ip-10-0-0-50 ~ $
```

Gambar 10.24. Membuat pod reservasi sumber daya dan RC untuk MySQL

Jelaskan pod reservasi sumber daya, dan Anda akan melihat bahwa pod tersebut memesan sumber daya yang ditentukan seperti yang ditunjukkan pada Gambar 10.25.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe pod reserve-resource
Name:           reserve-resource
Namespace:      default
Node:           ip-10-0-0-64.ec2.internal/10.0.0.64
Start Time:     Mon, 11 Jul 2016 18:22:09 +0000
Labels:         <none>
Status:         Running
IP:             10.2.83.2
Controllers:    <none>
Containers:
  reserve-resource:
    Container ID:       docker://4bb88ca8468d834db1a97fe9b94ee3341ca95967553abb
4a06406178f6cbb93d
    Image:              gcr.io/google_containers/pause:0.8.0
    Image ID:           docker://sha256:bf595365a5588ec1bae3e9dc9efde13672f7b75
61b6cc4514a82f07be9d01ca6
    Port:
    Limits:
      cpu:      100m
      memory:   200Mi
    Requests:
      cpu:                      100m
      memory:                   200Mi
    State:                      Running
      Started:                  Mon, 11 Jul 2016 18:22:10 +0000
    Ready:                      True
    Restart Count:              0
    Environment Variables:      <none>
Conditions:
  Type          Status
  Ready         True
Volumes:
```

Gambar 10.25. Deskripsi pod untuk pod reservasi sumber daya

Ringkasan

Dalam bab ini, kita membahas konfigurasi dan penggunaan sumber daya komputasi. Dua sumber daya komputasi yang dapat dikonfigurasi adalah CPU dan memori. Dua nilai konfigurasi dapat ditentukan untuk masing-masing sumber daya ini, nilai yang diminta dan nilai yang dibatasi. Kemudian, kita membuat pod dengan permintaan dan batasan sumber daya komputasi yang dikonfigurasi. Kita juga membahas tentang overcommitting sumber daya dan memesan sumber daya pada sebuah node. Dalam bab berikutnya, kita akan membahas penggunaan configmaps.

BAB 11
MENGGUNAKAN CONFIGMAP

11.1 MANAJEMEN KONFIGURASI DENGAN CONFIGMAP DI KUBERNETES

Masalah

Pertimbangkan kasus penggunaan bahwa beberapa variabel lingkungan seperti nama pengguna dan kata sandi untuk basis data akan digunakan dalam beberapa file definisi pengontrol replikasi atau pod. Nilai nama pengguna dan kata sandi perlu ditentukan dalam setiap file definisi. Dan jika nama pengguna dan kata sandi berubah, semua file definisi juga perlu diperbarui, yang bisa sangat membosankan. Sebagai alternatif, nilai variabel dapat diberikan ke kubectl saat perintah dijalankan, yang melibatkan penentuan tanda baris perintah setiap kali perintah dijalankan.

Solusi

Pola manajemen ConfigMap adalah peta properti konfigurasi yang dapat digunakan dalam file definisi untuk pod, pengontrol replikasi, dan objek Kubernetes lainnya untuk mengonfigurasi variabel lingkungan, argumen perintah, dan file konfigurasi seperti pasangan kunci-nilai dalam volume, untuk mencantumkan beberapa contoh penggunaan. Satu ConfigMap dapat mengemas beberapa properti konfigurasi sebagai pasangan kunci/nilai. Dengan membuat ConfigMap, Anda menentukan properti konfigurasi dalam satu peta konfigurasi, yang dapat diperbarui sesuai kebutuhan tanpa harus memperbarui setiap file definisi tempat ConfigMap digunakan. Memisahkan kontainer dari data konfigurasi menyediakan portabilitas aplikasi yang berjalan di kontainer.

Gambaran Umum

Skema file definisi ConfigMap menyediakan bidang-bidang berikut (Tabel 11.1).

Tabel 11.1. Bidang-bidang ConfigMap

Bidang	Deskripsi
Jenis	Jenis sumber daya. Harus ditetapkan ke ConfigMap.
versi api	Versi skema
metadata	Metadata seperti nama, label, namespace, dan anotasi.
Data	Data konfigurasi sebagai pasangan kunci/nilai.

Dalam bab ini kita akan membahas ConfigMap dan beberapa penggunaan umum dari ConfigMap. Bab ini membahas topik-topik berikut:

- Perintah kubectl create configmap
- Menetapkan lingkungan
- Membuat ConfigMap dari direktori
- Membuat ConfigMap dari file

- ❖ Membuat ConfigMap dari nilai literal
- ❖ Menggunakan ConfigMap dalam volume

11.2 PERINTAH KUBECTL CREATE CONFIGMAP

Perintah kubectl create configmap digunakan untuk membuat ConfigMap dari file, direktori, atau nilai literal dan memiliki sintaksis berikut:

```
kubectl create configmap NAMA [--from-file=[kunci=]sumber] [--
from-literal=kunci1=nilai1] [--dry-run]
```

Saat membuat ConfigMap dari file, nama file membentuk kunci dalam ConfigMap dan konten file membentuk nilai. Saat membuat ConfigMap dari direktori, pasangan kunci/nilai ConfigMap dibuat dari setiap file dalam direktori dengan nama file sebagai kunci dan konten file sebagai nilai.

Hanya file biasa dalam direktori yang digunakan untuk membuat entri ConfigMap, dan konten direktori lainnya seperti subdirektori dan symlink dihilangkan. Argumen perintah untuk membuat ConfigMap dari direktori atau file adalah sama, --from-file.

Di bagian berikut, kita akan mengatur lingkungan dan membuat ConfigMap dari direktori, file, dan nilai literal dan juga menggunakan ConfigMap dalam pod sebagai variabel lingkungan, argumen perintah, atau file konfigurasi dalam volume.

Menetapkan Lingkungan

Buat kluster Kubernetes menggunakan AWS CloudFormation. Masuk SSH ke instans pengontrol, instal biner kubectl, dan daftarkan node, seperti yang dibahas di Bab 2. Jumlah node dalam kluster adalah variabel, defaultnya adalah satu node pekerja yang dapat dijadwalkan dan satu node pengontrol. Perintah kubectl get nodes mencantumkan enam node pekerja dan satu node pengontrol.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                         STATUS                     AGE
ip-10-0-0-50.ec2.internal    Ready,SchedulingDisabled   3m
ip-10-0-0-87.ec2.internal    Ready                      3m
ip-10-0-0-88.ec2.internal    Ready                      3m
ip-10-0-1-29.ec2.internal    Ready                      3m
ip-10-0-1-30.ec2.internal    Ready                      3m
ip-10-0-2-122.ec2.internal   Ready                      3m
ip-10-0-2-123.ec2.internal   Ready                      3m
core@ip-10-0-0-50 ~ $
```

Gambar 11.1. Node kluster Kubernetes

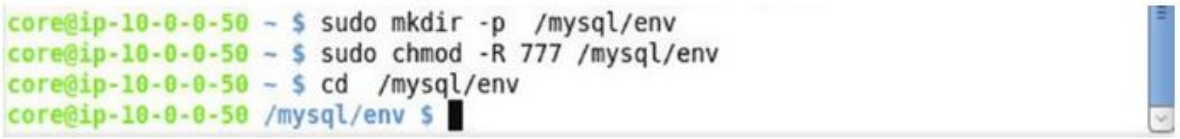
Membuat ConfigMap dari Direktori

Pada bagian ini kita akan membuat ConfigMap dari file-file dalam sebuah direktori.

Pertama, buat direktori /mysql/env dan tetapkan izin direktori ke global (777):

```
sudo mkdir /mysql/env
sudo chmod -R 777 /mysql/env
```

Direktori /mysql/env dibuat seperti yang ditunjukkan pada Gambar 11.2. CD (ganti direktori) ke direktori /mysql/env.



Gambar 11.2. Membuat direktori /mysql/env

Buat lima file, masing-masing dengan nama file yang akan membentuk kunci untuk properti konfigurasi di ConfigMap seperti yang tercantum dalam Tabel 11.2.

Tabel 11-2. Kolom ConfigMap

Nama berkas	Konten berkas
mysql.root.password	mysql
mysql.user	mysql
mysql.password	mysql
mysql.allow.empty.password	tidak
mysql.database	mysql

Gunakan editor vi untuk membuat setiap file; misalnya:

```
sudo vi mysql.root.password
```

Tentukan nilai yang akan digunakan sebagai kata sandi root dan simpan file dengan :wq seperti yang ditunjukkan pada Gambar 11.3.

Gambar 11.3. File mysql.root.password

Demikian pula, nilai yang disimpan dalam mysql.allow.empty.password akan menjadi no seperti yang ditunjukkan pada Gambar 11.4.

```
no

~
~
:wq
```

Gambar 11.4. Berkas mysql.allow.empty.password

Berkas-berkas tersebut harus dibuat di direktori /mysql/env, seperti yang ditunjukkan pada Gambar 11.5.

```
core@ip-10-0-0-50 /mysql/env $ ls -l
total 32
-rw-r--r--. 1 root root 4 Jul 16 18:52 mysql.allow.empty.password
-rw-r--r--. 1 root root 6 Jul 16 18:51 mysql.password
-rw-r--r--. 1 root root 6 Jul 16 18:50 mysql.root.password
-rw-r--r--. 1 root root 6 Jul 16 18:51 mysql.user
core@ip-10-0-0-50 /mysql/env $
```

Gambar 11.5. Berkas untuk membuat ConfigMap

Buat ConfigMap yang disebut `mysql-config dari direktori /mysql/env.`

```
./kubectl create configmap mysql-config --from-file=/mysql/env
```

ConfigMap `mysql-config` dibuat seperti yang ditunjukkan pada Gambar 11-6. Jelaskan ConfigMap:

```
./kubectl describe configmaps mysql-config
```

```
core@ip-10-0-0-50 ~ $  ./kubectl create configmap mysql-config --from-file=/mysq
l/env
configmap "mysql-config" created
core@ip-10-0-0-50 ~ $ ./kubectl describe configmaps mysql-config
Name:           mysql-config
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data
====
mysql.allow.empty.password:     4 bytes
mysql.database:                 6 bytes
mysql.password:                 6 bytes
mysql.root.password:            6 bytes
mysql.user:                     6 bytes


core@ip-10-0-0-50 ~ $
```

Gambar 11.6. Membuat ConfigMap dari sebuah direktori

Data konfigurasi yang disimpan dalam ConfigMap, yang pada dasarnya terdiri dari

pasangan kunci/nilai yang dibuat dari berkas-berkas di direktori, tercantum seperti yang ditunjukkan pada Gambar 11.6.

Anda dapat mencantumkan definisi ConfigMap YAML dengan perintah berikut:

```
./kubectl get configmaps mysql-config -o yaml
```

Berkas definisi mysql-config tercantum seperti yang ditunjukkan pada Gambar 11-7.

Gambar 11.7. Berkas definisi ConfigMap

Selanjutnya, gunakan ConfigMap dalam pengontrol replikasi; untuk melakukannya, buat file definisi mysql.yaml: sudo vi mysql.yaml

Gunakan peta konfigurasi mysql-config untuk mendapatkan nilai variabel lingkungan untuk citra Docker basis data MySQL mysql.

```
--
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: mysql-app
  name: mysql
spec:
  replicas: 3
  selector:
app: mysql-app
  template:
```

```
    metadata:
      labels:
        app: mysql-app
    spec:
      containers:
        -
          env:
          -
name: MYSQL_ROOT_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: mysql.root.password
                  name: mysql-config
              -
              name: MYSQL_DATABASE
              valueFrom:
                configMapKeyRef:
                  key: mysql.database
                  name: mysql-config
              -
              name: MYSQL_USER
              valueFrom:
                configMapKeyRef:
                  key: mysql.user
                  name: mysql-config
              -
              name: MYSQL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: mysql.user
                  name: mysql-config
              -
              name: MYSQL_ALLOW_EMPTY_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: mysql.allow.empty.password
                  name: mysql-config
          image: mysql
          name: mysql
          ports:
            -
              containerPort: 3306
```

MySQL.yaml ditunjukkan dalam editor vi pada Gambar 11.8.

```
-
  env:
    -
     name: MYSQL_ROOT_PASSWORD
     valueFrom:
       configMapKeyRef:
         key: mysql.root.password
         name: mysql-config
    -
     name: MYSQL_DATABASE
     valueFrom:
       configMapKeyRef:
         key: mysql.database
         name: mysql-config
    -
     name: MYSQL_USER
     valueFrom:
       configMapKeyRef:
         key: mysql.user
         name: mysql-config
    -
     name: MYSQL_PASSWORD
     valueFrom:
       configMapKeyRef:
         key: mysql.password
         name: mysql-config
    -
     name: MYSQL_ALLOW_EMPTY_PASSWORD
     valueFrom:
       configMapKeyRef:
         key: mysql.allow.empty.password
         name: mysql-config
  image: mysql
_
```

Gambar 11.8. Menggunakan referensi kunci ConfigMap dalam file definisi RC

Kunci untuk ConfigMap tidak boleh berupa nama sembarangan, tetapi harus mengikuti regexp tertentu. Untuk mendemonstrasikannya, gunakan kunci yang sama dengan nama variabel lingkungan seperti yang ditunjukkan pada Gambar 11.9.

```
    env:
      -
        name: MYSQL_ROOT_PASSWORD
        valueFrom:
          configMapKeyRef:
            key: MYSQL_ROOT_PASSWORD
            name: mysql-config
      -
        name: MYSQL_DATABASE
        valueFrom:
          configMapKeyRef:
            key: MYSQL_DATABASE
            name: mysql-config
      -
        name: MYSQL_USER
        valueFrom:
          configMapKeyRef:
            key: MYSQL_USER
            name: mysql-config
      -
        name: MYSQL_PASSWORD
        valueFrom:
          configMapKeyRef:
            key: MYSQL_PASSWORD
            name: mysql-config
      -
        name: MYSQL_ALLOW_EMPTY_PASSWORD
        valueFrom:
          configMapKeyRef:
            key: MYSQL_ALLOW_EMPTY_PASSWORD
            name: mysql-config
    image: mysql
:wq
```

Gambar 11.9. Referensi kunci ConfigMap ditetapkan ke nilai yang sama dengan nama variabel lingkungan

Terjadi kesalahan, seperti yang ditunjukkan pada Gambar 11.10.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
The ReplicationController "mysql" is invalid.

* spec.template.spec.containers[0].env[0].valueFrom.configMapKeyRef.key: Invalid
 value: "MYSQL_ROOT_PASSWORD": must have at most 253 characters and match regex
\.?[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*
* spec.template.spec.containers[0].env[1].valueFrom.configMapKeyRef.key: Invalid
 value: "MYSQL_DATABASE": must have at most 253 characters and match regex \.?[a
-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*
* spec.template.spec.containers[0].env[2].valueFrom.configMapKeyRef.key: Invalid
 value: "MYSQL_USER": must have at most 253 characters and match regex \.?[a-z0-
9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*
* spec.template.spec.containers[0].env[3].valueFrom.configMapKeyRef.key: Invalid
 value: "MYSQL_PASSWORD": must have at most 253 characters and match regex \.?[a
-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*
* spec.template.spec.containers[0].env[4].valueFrom.configMapKeyRef.key: Invalid
 value: "MYSQL_ALLOW_EMPTY_PASSWORD": must have at most 253 characters and match
 regex \.?[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*
core@ip-10-0-0-50 ~ $
```

Gambar 11.10. Kesalahan yang menunjukkan nilai tidak valid untuk referensi kunci ConfigMap

Hapus RC mysql jika sudah ada dan buat pengontrol replikasi dari file definisi dengan ConfigMapKeyRef yang valid seperti pada Gambar 11.8.

```
./kubectl create -f mysql.yaml
```

Daftar RC dan pod:

```
./kubectl get rc
./kubectl get pods
```

RC dan pod dibuat seperti yang ditunjukkan pada Gambar 11.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl delete rc mysql
replicationcontroller "mysql" deleted
core@ip-10-0-0-50 ~ $ sudo vi mysql.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
replicationcontroller "mysql" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME      DESIRED   CURRENT   AGE
mysql     3         3         8s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
mysql-1cwns   1/1       Running   0          8s
mysql-5eegy   1/1       Running   0          8s
mysql-unuye   1/1       Running   0          8s
core@ip-10-0-0-50 ~ $
```

Gambar 11.11. Membuat pengontrol replikasi dengan file definisi yang valid

11.3 MEMBUAT CONFIGMAP DARI FILE

Selanjutnya, kita akan membuat ConfigMap hanya menggunakan beberapa file di direktori /mysql/env. Hanya variabel lingkungan `MYSQL_ROOT_PASSWORD` yang wajib diisi. Sebagai contoh, buat ConfigMap yang disebut mysql-config-2 dari file mysql.allow.empty.password dan mysql.root.password.

```
/kubectl create configmap mysql-config-2 --from-
file=/mysql/env/mysql.root.password --from-
file=/mysql/env/mysql.allow.empty.password
```

ConfigMap mysql-config-2 dibuat seperti yang ditunjukkan pada Gambar 11-12. Berikutnya, jelaskan ConfigMap. Dua pasangan kunci/nilai dicantumkan. ConfigMap juga dapat dicantumkan sebagai YAML.

```
core@ip-10-0-0-50 ~ $ ./kubectl create configmap mysql-config-2 --from-file=/my
sql/env/mysql.root.password --from-file=/mysql/env/mysql.allow.empty.password
configmap "mysql-config-2" created
core@ip-10-0-0-50 ~ $ ./kubectl describe configmaps mysql-config-2
Name:           mysql-config-2
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data
====
mysql.allow.empty.password:     3 bytes
mysql.root.password:            6 bytes


core@ip-10-0-0-50 ~ $ ./kubectl get configmaps mysql-config-2 -o yaml
apiVersion: v1
data:
  mysql.allow.empty.password: |
    no
  mysql.root.password: |
    mysql
kind: ConfigMap
metadata:
  creationTimestamp: 2016-07-16T19:22:39Z
  name: mysql-config-2
  namespace: default
  resourceVersion: "16040"
  selfLink: /api/v1/namespaces/default/configmaps/mysql-config-2
  uid: a9245c47-4b8a-11e6-ac0d-1241999f191f
core@ip-10-0-0-50 ~ $
```

Gambar 11.12. Membuat ConfigMap dari file

Berikutnya, gunakan ConfigMap dalam file definisi pengontrol replikasi mysql.yaml.

```
apiVersion: v1
kind: ReplicationController
```

```
metadata:
 labels:
   app: mysql-app
 name: mysql
spec:
 replicas: 3
 selector:
   app: mysql-app
 template:
    metadata:
      labels:
        app: mysql-app
       spec:
        containers:
       -
       env:
       -
       name: MYSQL_ROOT_PASSWORD
        valueFrom:
         configMapKeyRef:
           key: mysql.root.password
           name: mysql-config-2
       -
       name: MYSQL_ALLOW_EMPTY_PASSWORD
       valueFrom:
        configMapKeyRef:
          key: mysql.allow.empty.password
          name: mysql-config-2
       image: mysql
       name: mysql
       ports:
        -
          containerPort: 3306
```

Berkas definisi mysql.yaml ditunjukkan dalam editor vi pada Gambar 11-13.

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: mysql-app
  name: mysql
spec:
  replicas: 3
  selector:
    app: mysql-app
  template:
    metadata:
      labels:
        app: mysql-app
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: mysql.root.password
                  name: mysql-config-2
            -
              name: MYSQL_ALLOW_EMPTY_PASSWORD
              valueFrom:
                configMapKeyRef:
                  key: mysql.allow.empty.password
                  name: mysql-config-2
          image: mysql
:wq
```

Gambar 11.13. Menggunakan ConfigMaps

Buat pengontrol replikasi dari berkas definisi:

```
./kubectl create -f mysql.yaml
```

Daftar RC dan pod:

```
./kubectl get rc
./kubectl get pods
```

RC dan pod dibuat dan dicantumkan seperti yang ditunjukkan pada Gambar 11.14.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml
replicationcontroller "mysql" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc
NAME      DESIRED   CURRENT   AGE
mysql     3         3         7s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
mysql-96a3s   1/1       Running   0          7s
mysql-b19pk   1/1       Running   0          7s
mysql-mveyp   1/1       Running   0          7s
core@ip-10-0-0-50 ~ $
```

Gambar 11.14. Membuat pengontrol replikasi dan mencantumkan pod

Jelaskan sebuah pod, dan ConfigMap mysql-config-2 dengan data harus dicantumkan seperti yang ditunjukkan pada Gambar 11.15.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe pod mysql-96a3s
Name:           mysql-96a3s
Namespace:      default
Node:           ip-10-0-0-88.ec2.internal/10.0.0.88
Start Time:     Sat, 16 Jul 2016 19:33:38 +0000
Labels:         app=mysql-app
Status:         Running
IP:             10.2.40.2
Controllers:    ReplicationController/mysql
Containers:
  mysql:
    Container ID:       docker://7084958b889c213ad55c58afc5013e290cee5933215e10
6deae90573789f1979
    Image:              mysql
    Image ID:           docker://sha256:1195b21c3a45d9bf93aae497f2538f89a09aade
d18d6648753aa3ce76670f41d
    Port:               3306/TCP
    State:              Running
      Started:          Sat, 16 Jul 2016 19:33:39 +0000
    Ready:              True
    Restart Count:      0
    Environment Variables:
      MYSQL_ROOT_PASSWORD:              <set to the key 'mysql.root.password' o
f config map 'mysql-config-2'>
      MYSQL_ALLOW_EMPTY_PASSWORD:       <set to the key 'mysql.allow.empty.pass
word' of config map 'mysql-config-2'>
Conditions:
  Type          Status
  Ready         True
Volumes:
  default-token-ni84s:
    Type:       Secret (a volume populated by a Secret)
    SecretName: default-token-ni84s
```

Gambar 11.15. Deskripsi pod mencakup nilai variabel lingkungan yang menggunakan ConfigMap

11.4 MEMBUAT CONFIGMAP DARI NILAI LITERAL

Di bagian ini, kita akan membuat dan menggunakan ConfigMap menggunakan pasangan kunci/nilai literal yang ditentukan pada baris perintah dengan opsi --from-literal. Sebagai contoh, buat ConfigMap yang disebut hello-config dengan dua pasangan kunci/nilai,

message1=hello dan message2=kubernetes.

```
kubectl create configmap hello-config --from-literal=message1=hello -
--from- literal=message2=kubernetes
```

ConfigMap hello-config dengan dua pasangan kunci/nilai dibuat seperti yang ditunjukkan pada Gambar 11.16. Jelaskan configmap untuk mencantumkan kunci/nilai.

```
core@ip-10-0-0-50 ~ $ ./kubectl create configmap hello-config --from-literal=me
ssage1=hello --from-literal=message2=kubernetes
configmap "hello-config" created
core@ip-10-0-0-50 ~ $ ./kubectl describe configmaps hello-config
Name:           hello-config
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data
====
message1:       5 bytes
message2:       10 bytes


core@ip-10-0-0-50 ~ $ ./kubectl get configmaps hello-config  -o yaml
apiVersion: v1
data:
  message1: hello
  message2: kubernetes
kind: ConfigMap
metadata:
  creationTimestamp: 2016-07-16T19:41:52Z
  name: hello-config
  namespace: default
  resourceVersion: "19233"
  selfLink: /api/v1/namespaces/default/configmaps/hello-config
  uid: 58645d5b-4b8d-11e6-ac0d-1241999f191f
core@ip-10-0-0-50 ~ $
```

Gambar 11.16. Membuat ConfigMap dari nilai literal

Buat file definisi pod hello-world.yaml untuk menggunakan ConfigMap hello-world. Pod didasarkan pada citra Docker Ubuntu dan menjalankan perintah /bin/echo dengan dua properti konfigurasi di ConfigMap sebagai argumen.

```
--
apiVersion: v1
kind: Pod
metadata:
 labels:
  app: helloApp
 name: hello-world
spec:
```

```
containers:
  -
  args:
    - " $(MESSAGE1)"
    - " $(MESSAGE2)"
  command:
    - /bin/echo
    env:
    -
    name: MESSAGE1
    valueFrom:
     configMapKeyRef:
       key: message1
       name: hello-config
    -
    name: MESSAGE2
    valueFrom:
     configMapKeyRef:
       key: message2
       name: hello-config
    image: ubuntu
    name: hello
```

File definisi pod hello.yaml ditampilkan di editor vi pada Gambar 11-17.

Gambar 11.17. File definisi pod yang menggunakan ConfigMap

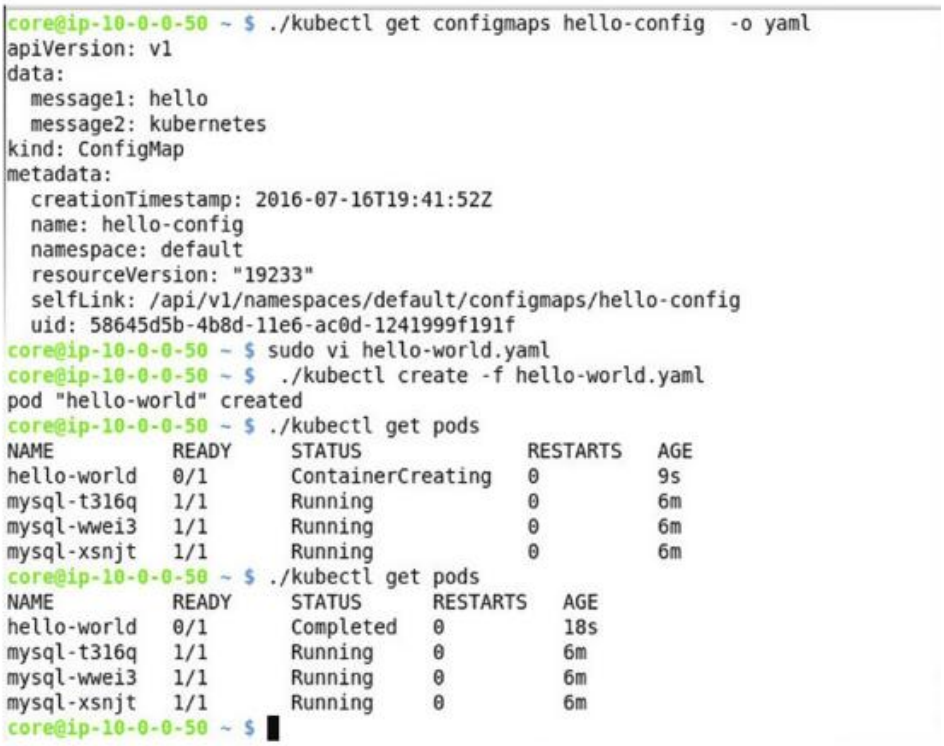
Daftar ConfigMap hello-world. ConfigMap memiliki dua pasangan kunci/nilai data. Buat pod dari file definisi:

```
./kubectl create –f hello-world.yaml
```

Daftar pod:

```
./kubectl get pods
```

Pod hello-world terdaftar sebagai Selesai seperti yang ditunjukkan pada Gambar 11.18.



Gambar 11.18. Membuat ConfigMap yang menggunakan pod

Daftar log yang dihasilkan dari pod:

```
./kubectl logs hello-world
```

Pesan yang dihasilkan dari dua pasangan kunci/nilai harus ditampilkan seperti yang ditunjukkan pada Gambar 11-19.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME          READY     STATUS      RESTARTS   AGE
hello-world   0/1       Completed   0          18s
mysql-t316q   1/1       Running     0          6m
mysql-wwei3   1/1       Running     0          6m
mysql-xsnjt   1/1       Running     0          6m
core@ip-10-0-0-50 ~ $ ./kubectl logs hello-world
 hello  kubernetes
core@ip-10-0-0-50 ~ $
```

Gambar 11.19. Log pod menyertakan pesan yang dihasilkan menggunakan ConfigMap

11.5 MENGGUNAKAN CONFIGMAP DALAM VOLUME

Di bagian ini, kita akan membuat ConfigMap untuk menyimpan pasangan kunci-nilai sertifikat dan menggunakan ConfigMap dalam volume. Buat file definisi cert.yaml untuk ConfigMap yang akan digunakan untuk menentukan sertifikat.

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: nginx-cert
data:
   cert.pem:
   |------BEGIN CERTIFICATE----
     abc
     -----END CERTIFICATE----
   privkey.pem:
   |------BEGIN PRIVATE KEY----
     abc
     -----END PRIVATE KEY-----
```

Berkas definisi ConfigMap ditampilkan di editor vi pada Gambar 11.20.

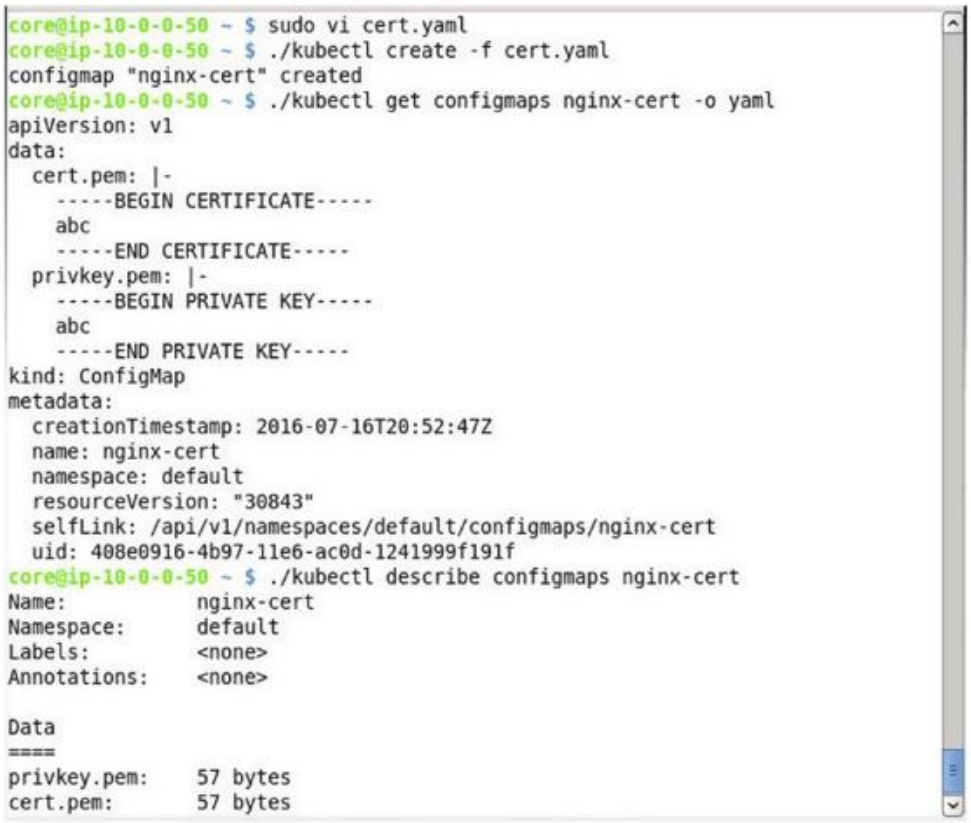
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-cert
data:
  cert.pem: |-
    -----BEGIN CERTIFICATE-----
    abc
    -----END CERTIFICATE-----
  privkey.pem: |-
    -----BEGIN PRIVATE KEY-----
    abc
    -----END PRIVATE KEY-----
```

Gambar 11.20. ConfigMap untuk menyimpan pasangan kunci-nilai sertifikat

Buat ConfigMap dari berkas definisi seperti yang ditunjukkan pada Gambar 11.21.

```
./kubectl create -f cert.yaml
```

Jelaskan ConfigMap untuk mencantumkan dua pasangan kunci/nilai seperti yang ditunjukkan pada Gambar 11.21.



Gambar 11.21. Membuat dan mencantumkan ConfigMap yang menyimpan pasangan kunci/nilai

Selanjutnya, gunakan ConfigMap dalam pod. Buat volume bertipe ConfigMap dari nginx-cert ConfigMap. Pasang volume dalam pod di beberapa direktori cert tempat sertifikat dapat diambil, seperti /etc/config/.

```
--
apiVersion: v1
kind: Pod
metadata:
 name: configmap-volume
spec:
 containers:
   -
```

```
image: nginx
name: nginx
volumeMounts:
 -
   mountPath: /etc/config/cert
   name: config-volume
   readOnly: true
volumes:
 -
configMap:
 name: nginx-cert
name: config-volume
```

File definisi pod ditampilkan di editor vi pada Gambar 11.22.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: configmap-volume
spec:
  containers:
    -
     image: nginx
     name: nginx
     volumeMounts:
        -
         mountPath: /etc/config/cert
         name: config-volume
  volumes:
    -
     configMap:
        name: nginx-cert
     name: config-volume
~
```

Gambar 11.22. Pod yang menggunakan ConfigMap dalam volume mount

Buat pod dari file definisi dan daftarkan pod seperti yang ditunjukkan pada Gambar 11.23.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f pod.yaml
pod "configmap-volume" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME               READY     STATUS    RESTARTS   AGE
configmap-volume   1/1       Running   0          12s
core@ip-10-0-0-50 ~ $
```

Gambar 11.23. Membuat dan mencantumkan pod

Jelaskan pod untuk mencantumkan volume bertipe ConfigMap seperti yang ditunjukkan pada Gambar 11.24.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
configmap-volume  1/1       Running   0          12s
core@ip-10-0-0-50 ~ $ ./kubectl describe pod configmap-volume
Name:           configmap-volume
Namespace:      default
Node:           ip-10-0-0-88.ec2.internal/10.0.0.88
Start Time:     Sat, 16 Jul 2016 20:56:47 +0000
Labels:         <none>
Status:         Running
IP:             10.2.40.2
Controllers:    <none>
Containers:
  nginx:
    Container ID:               docker://aa7d39f31c6d8946837cb546bd05772d0b5a63
019afb8e65b8eb719fe5b5dc70
    Image:                      nginx
    Image ID:                   docker://sha256:0d409d33b27e47423b049f7f863faa0
8655a8c901749c2b25b93ca67d01a470d
    Port:
    State:                      Running
      Started:                  Sat, 16 Jul 2016 20:56:48 +0000
    Ready:                      True
    Restart Count:              0
    Environment Variables:      <none>
Conditions:
  Type          Status
  Ready         True
Volumes:
  config-volume:
    Type:       ConfigMap (a volume populated by a ConfigMap)
    Name:       nginx-cert
```

Gambar 11.24. Deskripsi pod mencantumkan volume bertipe ConfigMap

Ringkasan

Dalam bab ini, kami memperkenalkan ConfigMap, yang merupakan peta properti konfigurasi yang dapat digunakan dalam definisi objek Kubernetes seperti pod, pengontrol replikasi, dan juga untuk menetapkan variabel lingkungan dan argumen perintah. Selanjutnya, kami membahas pembuatan ConfigMap dari direktori, file, dan nilai literal, dan akhirnya menggunakan ConfigMap. Dalam bab berikutnya, kami akan membahas pengaturan kuota sumber daya.

BAB 12
MENGGUNAKAN KUOTA SUMBER DAYA

12.1 MANAJEMEN KUOTA SUMBER DAYA DI KUBERNETES

Pada Bab 10, kami memperkenalkan model konsumsi sumber daya berdasarkan permintaan dan batasan, yang digunakan untuk mengalokasikan sumber daya (CPU dan memori) ke kontainer pod.

Masalah

Meskipun kami membahas pengalokasian sumber daya ke kontainer pod, kami tidak mempertimbangkan beberapa faktor lain. Persyaratan sumber daya bervariasi dari satu tim pengembangan ke tim pengembangan lainnya. Jika satu tim pengembangan menggunakan semua atau sebagian besar sumber daya pada node, tim lain tidak akan dapat menjalankan aplikasi apa pun pada node yang sama.

Kedua, persyaratan sumber daya bervariasi di berbagai fase pengembangan aplikasi. Pengembangan aplikasi akan memiliki penggunaan sumber daya yang berbeda dari pengujian aplikasi dan pekerjaan aplikasi dalam produksi. Pola alokasi sumber daya yang dibahas dalam Bab 10 tidak memberikan solusi untuk semua faktor ini.

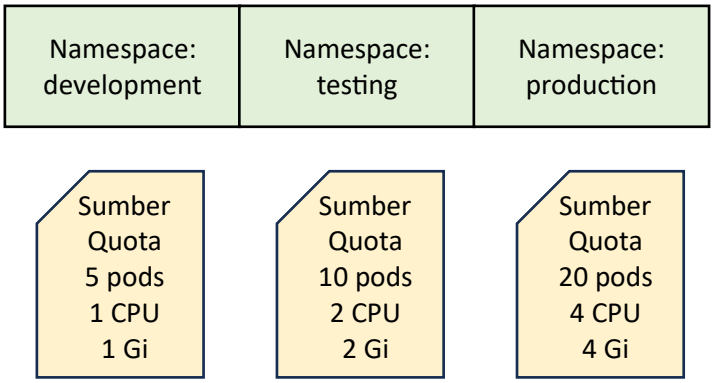
Solusi

Kubernetes menyediakan pola desain manajemen untuk kuota elastis. Kuota elastis tidak sepenuhnya elastis, dan batas atas tetap yang fleksibel sampai batas tertentu berdasarkan cakupan (dibahas nanti dalam bab ini) diberlakukan. Kuota sumber daya adalah spesifikasi untuk membatasi penggunaan sumber daya tertentu dalam namespace tertentu. Kuota tidak berlaku untuk objek tertentu, seperti pod atau pengontrol replikasi, tetapi untuk penggunaan agregat dalam namespace. Tujuannya adalah untuk menyediakan sumber daya yang adil bagi tim yang berbeda, dengan setiap tim diberi namespace dengan kuota.

Aplikasi kuota lainnya adalah membuat namespace yang berbeda untuk produksi, pengembangan, dan pengujian; fase pengembangan aplikasi yang berbeda memiliki persyaratan sumber daya yang berbeda. Pembuatan atau pembaruan sumber daya tidak boleh melebihi batasan kuota, jika tidak, sumber daya tidak akan dibuat atau diperbarui, dan pesan kesalahan akan muncul. Kuota dapat ditetapkan pada sumber daya komputasi (CPU dan memori), yang dibahas dalam bab 10, dan jumlah objek (seperti pod, pengontrol replikasi, layanan, penyeimbang beban, dan ConfigMap, untuk menyebutkan beberapa).

Saat kuota ditetapkan untuk sumber daya komputasi, permintaan atau batasan harus ditentukan untuk sumber daya tersebut. Kuota diaktifkan secara default. Total kapasitas kluster, yang dapat bervariasi jika node ditambahkan atau dihapus, bukanlah faktor pembatas saat menetapkan kuota. Jumlah total kuota namespace dapat melebihi kapasitas klaster, dan perebutan sumber daya akan diselesaikan berdasarkan siapa yang datang pertama dilayani pertama. Perebutan sumber daya diselesaikan sebelum sumber daya dibuat dan tidak

memengaruhi sumber daya yang telah dibuat. Setelah sumber daya dibuat, perubahan apa pun pada pengaturan kuota tidak memengaruhi sumber daya tersebut.



Gambar 12.1 Kuota Sumber Daya Yang Berbeda Untuk Namespace Yang Berbeda

Kuota secara opsional dapat dikaitkan dengan cakupan, yang selanjutnya membatasi jenis sumber daya yang akan diterapkan kuota. Cakupan yang tersedia adalah Terminating, NotTerminating, BestEffort, dan NotBestEffort. Cakupan Terminating adalah untuk pod yang berakhir, dan cakupan NotTerminating adalah untuk pod yang tidak berakhir. Cakupan BestEffort adalah untuk pod yang memiliki kualitas layanan upaya terbaik, dan cakupan NotBestEffort adalah untuk pod yang tidak memiliki kualitas layanan upaya terbaik. Bidang spesifikasi kuota sumber daya dibahas dalam Tabel 12.1.

Tabel 12.1 Bidang Spesifikasi Kuota Sumber Daya

Bidang	Deskripsi
jenis	Harus ditetapkan ke ResourceQuota.
versiapi	Versi skema.
metadata	Metadata seperti nama, label, dan anotasi.
spesifikasi	Spesifikasi ResourceQuota menyediakan dua kolom: hard, yang menetapkan batasan hard untuk setiap sumber daya yang ditentukan, dan scopes, yang menetapkan cakupan. Kuota mengukur penggunaan sumber daya hanya jika cocok dengan persimpangan cakupan. Spesifikasi menetapkan pengaturan yang diinginkan untuk batasan hard.
status	Status adalah penggunaan sumber daya yang sebenarnya dan ditetapkan dengan hard dan used. Status hard adalah batasan hard yang diberlakukan, dan used adalah total penggunaan sumber daya yang sebenarnya di namespace. Nilai status adalah nilai yang benar-benar diterapkan, berbeda dengan pengaturan yang diinginkan dalam spesifikasi.

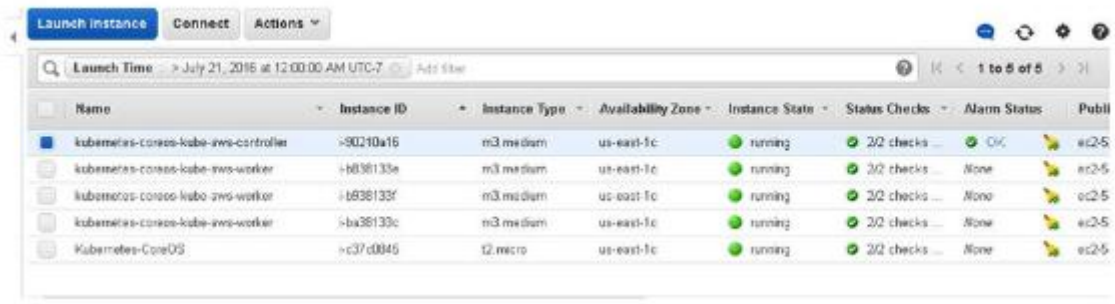
Tinjauan Umum

Dalam bab ini, kita akan membahas penggunaan kuota sumber daya dengan aplikasi Kubernetes. Bab ini membahas topik-topik berikut.

- Menetapkan lingkungan
- Menentukan kuota sumber daya komputasi
- Melebihi kuota sumber daya komputasi
- Menentukan kuota objek
- Melebihi kuota sumber daya objek
- Menentukan kuota upaya terbaik
- Menggunakan kuota
- Melebihi kuota objek
- Melebihi kuota ConfigMaps

Mengatur Lingkungan

Buat klaster Kubernetes dengan AWS CloudFormation dengan satu simpul pengontrol dan tiga simpul pekerja seperti yang ditunjukkan pada Gambar 12.2.



Gambar 12.2 Cloudformation Untuk Klaster Kubernetes Di Coreos

Login SSH ke instans pengontrol, instal biner kubectl, dan daftarkan simpul:

```
./kubectl get nodes
```

Instans pengontrol dan simpul pekerja harus dicantumkan seperti yang ditunjukkan pada Gambar 12.3.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-180.ec2.internal    Ready                      1m
ip-10-0-0-181.ec2.internal    Ready                      1m
ip-10-0-0-182.ec2.internal    Ready                      1m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   5m
core@ip-10-0-0-50 ~ $
```

Gambar 12.3 Mencantumkan Simpul Klaster Kubernetes

12.2 MENENTUKAN KUOTA SUMBER DAYA KOMPUTASI

Kuota sumber daya komputasi membatasi total sumber daya komputasi yang digunakan oleh pod dalam namespace. Tabel 12.2 mencantumkan sumber daya komputasi yang didukung.

Tabel 12.2 Sumber Daya Komputasi yang Didukung

Sumber Daya Komputasi	Deskripsi
cpu	Total semua permintaan CPU di semua pod dalam status non-terminal tidak boleh melebihi pengaturan ini. Kontainer harus menentukan nilai permintaan->CPU jika kuota CPU ditetapkan, atau pembuatan pod dapat gagal.
limits.cpu	Total semua batasan CPU di semua pod dalam status non-terminal tidak boleh melebihi pengaturan ini. Kontainer harus menentukan nilai limits->CPU jika kuota limits.cpu ditetapkan, atau pembuatan pod dapat gagal.
limits.memory	Jumlah total semua batas memori di semua pod dalam status non-terminal tidak boleh melebihi pengaturan ini. Kontainer harus menentukan nilai limits->memory jika kuota limits.memory ditetapkan, atau pembuatan pod dapat gagal.
memory	Total semua permintaan memori di semua pod dalam status non-terminal tidak boleh melebihi pengaturan ini. Kontainer harus menentukan nilai permintaan->memori jika kuota memori ditetapkan. Jika tidak, pembuatan pod bisa gagal.
requests.cpu	Sama seperti CPU.
requests.memory	Sama seperti memori.

Buat file definisi ResourceQuota compute-resource-quotas.yaml. Di kolom spec, tetapkan batasan ketat pada jumlah pod, total permintaan CPU, total permintaan memori, batasan CPU, dan batasan memori. Tetapkan NotBestEffort sebagai cakupan dalam daftar cakupan.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resource-quotas
spec:
  hard:
    pods: "10"
    requests.cpu: "1"
    requests.memory: 2Gi
    limits.cpu: "2"
    limits.memory: 4Gi
  scopes:
```

-
  NotBestEffort

Berkas definisi ditampilkan di editor vi pada Gambar 12.4.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resource-quotas
spec:
  hard:
    pods: "10"
    requests.cpu: "1"
    requests.memory: 2Gi
    limits.cpu: "2"
    limits.memory: 4Gi
  scopes:
    -
     NotBestEffort
~
```

Gambar 12.4 Berkas Definisi Resourcequota

Buat ResourceQuota di namespace default:

```
./kubectl create -f compute-resource-quotas.yaml --
namespace=default
```

ResourceQuota dibuat seperti yang ditunjukkan pada Gambar 12.5.

```
core@ip-10-0-0-50 ~ $ sudo vi compute-resource-quotas.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f compute-resource-quotas.yaml --namespa
ce=default
resourcequota "compute-resource-quotas" created
core@ip-10-0-0-50 ~ $
```

Gambar 12.5 Membuat ResourceQuota

Daftar kuota:

```
./kubectl get quota –namespace=default
```

Kuota compute-resource-quotas harus dicantumkan seperti yang ditunjukkan pada Gambar 12.6.

```
core@ip-10-0-0-50 ~ $ ./kubectl get quota --namespace=default
NAME                   AGE
compute-resource-quotas   24s
core@ip-10-0-0-50 ~ $
```

Gambar 12.6 Mencantumkan Kuota Di Namespace Default

Jelaskan kuota compute-resource-quotas:

```
./kubectl describe quota compute-resource-quotas --
namespace=default
```

Deskripsi kuota mencakup sumber daya yang digunakan dan batasan yang ketat. Karena kita belum membuat sumber daya apa pun, nilai kolom Used semuanya adalah 0, seperti yang ditunjukkan pada Gambar 12.7.

```
core@ip-10-0-0-50 ~ $ ./kubectl get quota --namespace=default
NAME                   AGE
compute-resource-quotas   24s
core@ip-10-0-0-50 ~ $ ./kubectl describe quota compute-resource-quotas --namespa
ce=default
Name:           compute-resource-quotas
Namespace:      default
Scopes:         NotBestEffort
 * Matches all pods that do not have best effort quality of service.
Resource        Used    Hard
--------        ----    ----
limits.cpu      0       2
limits.memory   0       4Gi
pods            0       10
requests.cpu    0       1
requests.memory 0       2Gi


core@ip-10-0-0-50 ~ $
```

Gambar 12.7 Menjelaskan Compute-Resource-Quotas

12.3 MELEBIHI KUOTA SUMBER DAYA KOMPUTASI

Selanjutnya, kita akan menggunakan kuota sumber daya untuk membatasi penggunaan sumber daya komputasi di namespace default. Buat file definisi RC mysql.yaml:

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql-rc
  labels:
    app: mysql-app
spec:
  replicas: 3
```

```
  selector:
    app: mysql-app
  template:
    metadata:
      labels:
        app: mysql-app
spec:
   containers:
     -
       env:
         -
        name: MYSQL_ROOT_PASSWORD
        value: mysql
    image: mysql
    name: mysql
    ports:
    -
        containerPort: 3306
   resources:
     requests:
        memory: “640Mi”
        cpu: ‘500m”
     limits:
        memory: “1280Mi”
        cpu: “2”
```

Buat pengontrol replikasi dengan 10 replika:

```
./kubectl scale rc mysql-rc --replicas=10
```

Berikutnya, jelaskan compute-resource-quotas. Kolom Used mencantumkan sumber daya aktual yang digunakan. Tidak ada sumber daya yang digunakan yang melebihi batas keras, seperti yang ditunjukkan pada Gambar 12.8.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe quota compute-resource-quotas --namespa
ce=default
Name:           compute-resource-quotas
Namespace:      default
Scopes:         NotBestEffort
 * Matches all pods that do not have best effort quality of service.
Resource        Used    Hard
--------        ----    ----
limits.cpu      2       2
limits.memory   1280Mi  4Gi
pods            10      10
requests.cpu    500m    1
requests.memory 640Mi   2Gi


core@ip-10-0-0-50 ~ $
```

Gambar 12.8 Sumber Daya Yang Digunakan Tidak Melebihi Batas Keras

Untuk menunjukkan bahwa batas keras tidak dapat dilampaui, skala RC menjadi 20 replika:

```
./kubectl scale rc mysql-rc --replicas=20
```

Sekarang jelaskan compute-resources-quota. Kolom Used masih memiliki 10 di baris pods, seperti yang ditunjukkan pada Gambar 12.9.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale rc mysql-rc --replicas=20
replicationcontroller "mysql-rc" scaled
core@ip-10-0-0-50 ~ $ ./kubectl describe quota compute-resource-quotas --namespa
ce=default
Name:           compute-resource-quotas
Namespace:      default
Scopes:         NotBestEffort
 * Matches all pods that do not have best effort quality of service.
Resource        Used    Hard
--------        ----    ----
limits.cpu      2       2
limits.memory   1280Mi  4Gi
pods            10      10
requests.cpu    500m    1
requests.memory 640Mi   2Gi


core@ip-10-0-0-50 ~ $
```

Gambar 12.9 Pod Tidak Melebihi Batas Keras Meskipun Diskalakan Untuk Melakukannya

Jelaskan RC, dan Anda akan melihat bahwa Replika tercantum sebagai 10 saat ini / 20 yang diinginkan, seperti yang ditunjukkan pada Gambar 12.10.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale rc mysql-rc --replicas=20
replicationcontroller "mysql-rc" scaled
core@ip-10-0-0-50 ~ $ ./kubectl describe rc mysql-rc
Name:           mysql-rc
Namespace:      default
Image(s):       mysql
Selector:       app=mysql-app,deployment=v1
Labels:         app=mysql-app
Replicas:       10 current / 20 desired
Pods Status:    10 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                  Message
  ---------     --------        -----   ----                            --------
-----   --------        ------                  -------
  23m           23m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-1nyks
  23m           23m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-h2q9h
  23m           23m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-rgxzs
  22m           22m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-y6cvy
  22m           22m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-5j4gp
  22m           22m             1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-xhf84
  22m           22m             1       {replication-controller }               N
```

Gambar 12.10 Menjelaskan Pengontrol Replikasi: 10 Replika Saat Ini, Bukan 20 Yang Diinginkan

Cantumkan pod di seluruh kluster, dan Anda mungkin melihat beberapa pod dihentikan atau dimulai ulang jika beberapa sumber daya komputasi lainnya terlampaui, seperti yang ditunjukkan pada Gambar 12.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME              READY     STATUS             RESTARTS   AGE       IP          N
ODE
mysql-rc-1nyks    1/1       Running            4          5m        10.2.26.2   i
p-10-0-0-180.ec2.internal
mysql-rc-5911t    0/1       CrashLoopBackOff   3          4m        10.2.47.4   i
p-10-0-0-182.ec2.internal
mysql-rc-5j4gp    0/1       CrashLoopBackOff   3          4m        10.2.47.3   i
p-10-0-0-182.ec2.internal
mysql-rc-h2q9h    0/1       CrashLoopBackOff   4          5m        10.2.98.3   i
p-10-0-0-181.ec2.internal
mysql-rc-k07te    1/1       Running            3          4m        10.2.26.6   i
p-10-0-0-180.ec2.internal
mysql-rc-rgxzs    1/1       Running            4          5m        10.2.26.3   i
p-10-0-0-180.ec2.internal
mysql-rc-uf3za    1/1       Running            3          4m        10.2.26.5   i
p-10-0-0-180.ec2.internal
mysql-rc-xhf84    1/1       Running            3          4m        10.2.26.4   i
p-10-0-0-180.ec2.internal
mysql-rc-xj2oj    1/1       Running            4          4m        10.2.98.5   i
p-10-0-0-181.ec2.internal
mysql-rc-y6cvy    1/1       Running            4          4m        10.2.98.4   i
p-10-0-0-181.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 12.11 Pod Dihentikan Atau Dimulai Ulang Jika Beberapa Sumber Daya Terlampaui

12.4 MENENTUKAN KUOTA OBJEK

Di bagian ini, kami akan menetapkan kuota objek dan menunjukkan apa yang terjadi saat kuota objek terlampaui: objek sumber daya tidak dibuat. Buat file definisi ResourceQuota object-quotas.yaml. Tetapkan batasan pasti untuk jumlah ConfigMap, pengontrol replikasi, dan layanan:

```
apiVersion: v1
kind: ResourceQuota
metadata:
   name: object-quotas
spec:
   hard:
     configmaps: "5"
     replicationcontrollers: "1"
     services: "2"
```

Gambar 12.12. Berkas definisi ResourceQuota untuk kuota objek

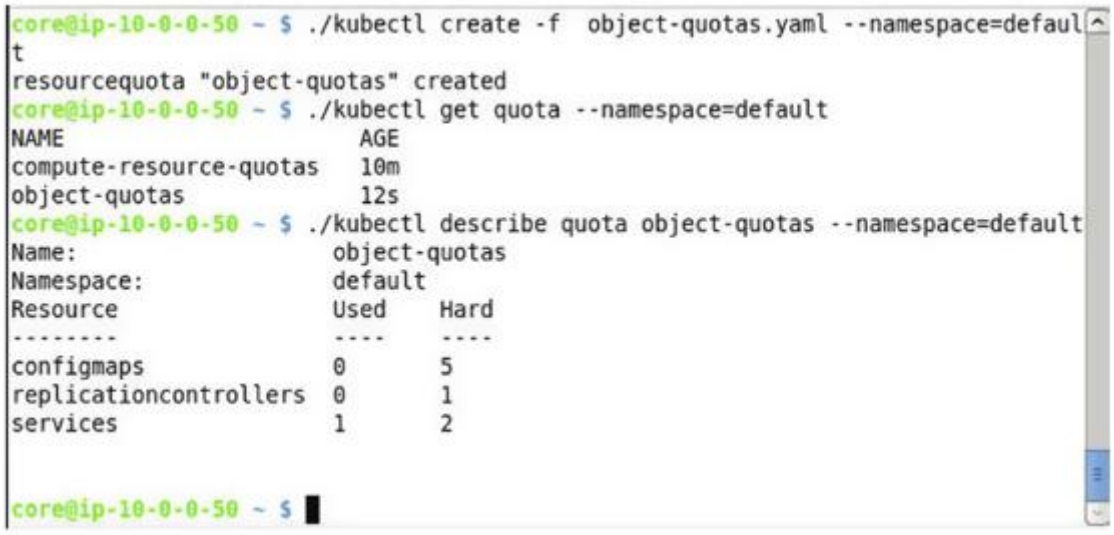
Buat ResourceQuota dari berkas definisi di namespace default seperti yang ditunjukkan pada Gambar 12.13.

```
./kubectl create -f object-quotas.yaml --namespace=default
```

Kemudian buat daftar dan jelaskan kuota:

```
./kubectl get quota --namespace=default
./kubectl describe quota object-quotas --namespace=default
```

Kuota sumber daya dibuat dan dicantumkan seperti yang ditunjukkan pada Gambar 12.13. Deskripsi kuota mencakup Sumber daya yang digunakan dan Batasan ketat.



Gambar 1213 Membuat, Mencantumkan, Dan Menjelaskan Kuota Sumber Daya Untuk Kuota Objek

Melebihi Kuota Objek

Pada bagian ini kami akan menunjukkan bahwa kuota objek tidak dapat dilampaui; sebaliknya, objek sumber daya yang akan melampaui batas yang ditetapkan tidak dibuat. Pertama, buat file definisi RC mysql-rc.yaml.

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: mysql-rc
  labels:
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
       containers:
      -
       name: mysql
       image: mysql
    env:
```

```
      -
         name: MYSQL_ROOT_PASSWORD
         value: mysql
    image: mysql
    name: mysql
    ports:
      -
        containerPort: 3306
    resources:
      requests:
        memory: "64Mi"
        cpu: "0.1"
      limits:
        memory: "128Mi"
        cpu: "0.2"
```

Berkas definisi ditampilkan di editor vi pada Gambar 12.14.

```
    app: mysql-app
spec:
  replicas: 3
  selector:
    app: mysql-app
    deployment: v1
  template:
    metadata:
      labels:
        app: mysql-app
        deployment: v1
    spec:
      containers:
        -
         env:
            -
             name: MYSQL_ROOT_PASSWORD
             value: mysql
         image: mysql
         name: mysql
         ports:
            -
             containerPort: 3306
         resources:
            requests:
              memory: "64Mi"
              cpu: ".1"
            limits:
              memory: "128Mi"
              cpu: ".2"

:wq
```

Gambar 12.14 Berkas Definisi Replicationcontroller

Buat RC dan daftarkan RC dan pod seperti yang ditunjukkan pada Gambar 12.15.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-rc-b6j4o    1/1       Running   0          29s       10.2.26.2   ip-10-0-0-
180.ec2.internal
mysql-rc-t09up    1/1       Running   0          28s       10.2.47.3   ip-10-0-0-
182.ec2.internal
mysql-rc-zxdun    1/1       Running   0          29s       10.2.98.3   ip-10-0-0-
181.ec2.internal
core@ip-10-0-0-50 ~ $
```

Gambar 12.15 Membuat dan mencantumkan RC

Mencantumkan pod di seluruh klaster menunjukkan bahwa setiap pod dijadwalkan pada node yang berbeda, seperti yang ditunjukkan pada Gambar 12.16.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME              READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-rc-b6j4o    1/1       Running   0          29s       10.2.26.2   ip-10-0-0-
180.ec2.internal
mysql-rc-t09up    1/1       Running   0          28s       10.2.47.3   ip-10-0-0-
182.ec2.internal
mysql-rc-zxdun    1/1       Running   0          29s       10.2.98.3   ip-10-0-0-
181.ec2.internal
core@ip-10-0-0-50 ~ $
```

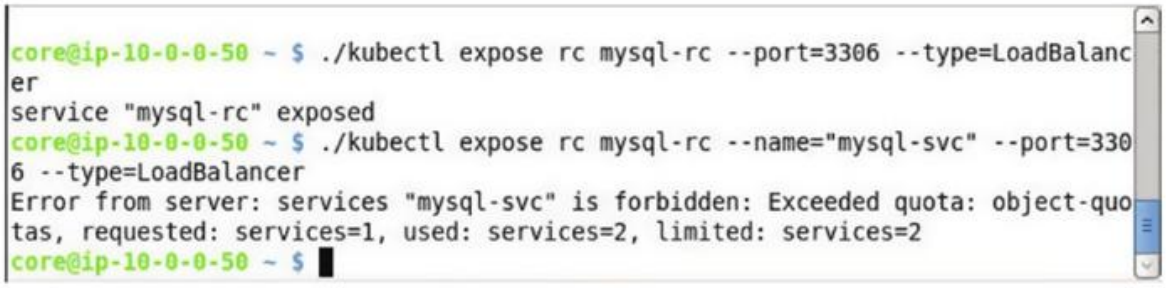
Gambar 12.16 Masing-Masing Dari Tiga Pod Dijadwalkan Pada Node Yang Berbeda, Dengan Mempertimbangkan Konsumsi Sumber Daya

Selanjutnya, buat pengontrol replikasi lain dari file definisi RC lain, mirip dengan yang pertama. RC kedua tidak dibuat, dan pesan kesalahan menunjukkan bahwa kuota objek telah terlampaui, seperti yang ditunjukkan pada Gambar 12.17.

```
core@ip-10-0-0-50 ~ $ sudo vi mysql2.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql2.yaml
Error from server: error when creating "mysql2.yaml": replicationcontrollers "my
sql-rc2" is forbidden: Exceeded quota: object-quotas, requested: replicationcont
rollers=1, used: replicationcontrollers=1, limited: replicationcontrollers=1
core@ip-10-0-0-50 ~ $
```

Gambar 12.17 Pesan Kesalahan Menunjukkan Bahwa Kuota Objek Telah Terlampaui Untuk Pengontrol Replikasi

Batas keras pada jumlah layanan adalah 2. Buat satu layanan dan buat yang lain dengan nama yang berbeda dari default. Layanan kedua tidak dibuat, dan pesan kesalahan menunjukkan bahwa kuota objek telah terlampaui, seperti yang ditunjukkan pada Gambar 12.18.



Gambar 12.18 Pesan Kesalahan Menunjukkan Bahwa Kuota Objek Telah Terlampaui Untuk Layanan

Selanjutnya, kami akan menunjukkan cara melampaui kuota ConfigMaps. Bab 11 menunjukkan cara membuat ConfigMap, tetapi saya akan mengulangi prosedurnya secara singkat di sini. Kita akan membuat beberapa ConfigMap dari file dalam sebuah direktori. Nama file harus sama dengan kunci ConfigMap, dan nilainya adalah konten file. Buat direktori dan tetapkan izinnya:

```
sudo mkdir /mysql/env
sudo chmod -R 777 /mysql/env
cd /mysql/env
```

Tambahkan lima berkas yang tercantum dalam Tabel 12.3 ke direktori.

Tabel 12.3. Berkas untuk Membuat ConfigMap

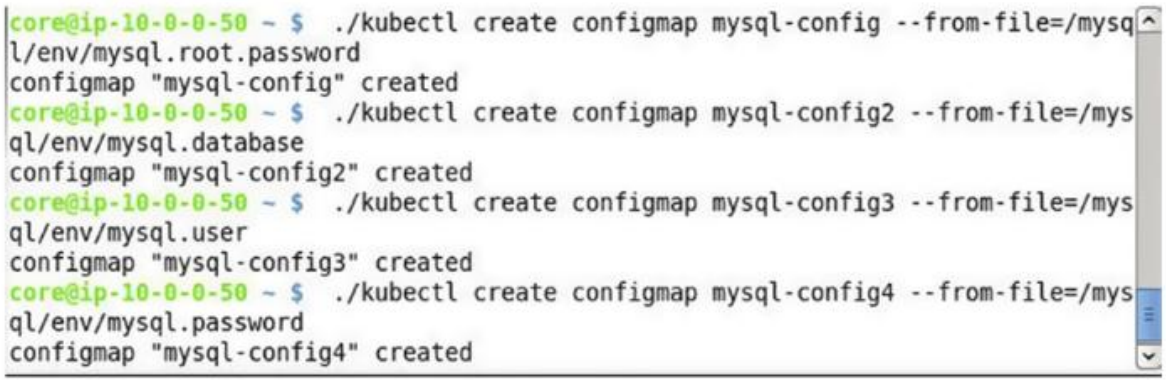
File	Konten
mysql.root.password	mysql
mysql.database	mysqldb
mysql.user	mysql
mysql.password	mysql
mysql.allow.empty.password	no

Buat lima ConfigMap dari lima file.

```
./kubectl create configmap mysql-config --from-
file=/mysql/env/mysql.root.password
./kubectl create configmap mysql-config2 --from-
file=/mysql/env/mysql.database
./kubectl create configmap mysql-config3 --from-
file=/mysql/env/mysql.user
./kubectl create configmap mysql-config4 --from-
file=/mysql/env/mysql.password
./kubectl create configmap mysql-config5 --from-
```

```
file=/mysql/env/allow.empty.password
```

Kelima ConfigMap dibuat seperti yang ditunjukkan pada Gambar 12.19.

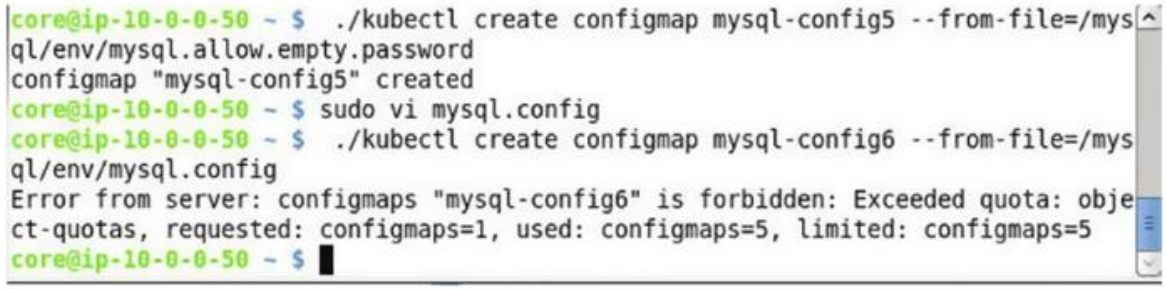


Gambar 12.19 Membuat ConfigMap

Batasan keras pada jumlah ConfigMap adalah 5. Buat file lain, bernama mysql.config, dan atur kontennya ke mysql. Buat, atau coba buat, ConfigMap keenam:

```
./kubectl create configmap mysql-config6 --from-
file=/mysql/env/mysql.config
```

Pesan kesalahan menunjukkan bahwa jumlah ConfigMap telah terlampaui, seperti yang ditunjukkan pada Gambar 12.20.



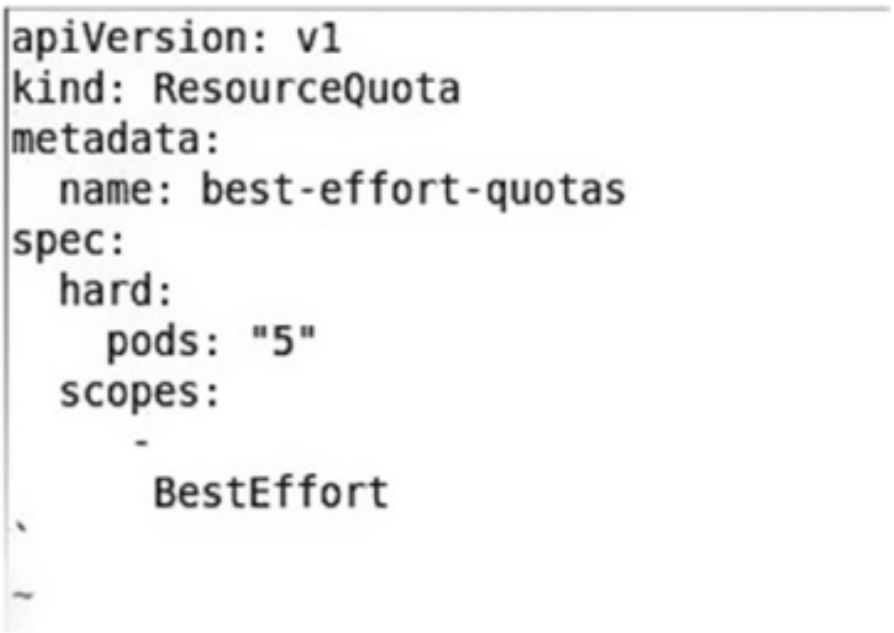
Gambar 12.20 Pesan Kesalahan Menunjukkan Bahwa Kuota Objek-Kuota Telah Terlampaui Untuk Configmap

12.5 MENDEFINISIKAN KUOTA CAKUPAN BEST-EFFORT

Kuota cakupan BestEffort hanya digunakan untuk melacak pod. Dan jika sumber daya berlebih tersedia, pod yang melebihi batas keras dapat dijadwalkan, meskipun pod (yang melampaui batas keras) akan menjadi yang pertama dihentikan jika sumber daya diperlukan untuk objek lain. Untuk mendemonstrasikannya, buat file definisi ResourceQuota best-effort-quotas.yaml. Tetapkan batasan ketat pada jumlah pod menjadi 5. Tetapkan cakupan ke BestEffort.

```
apiVersion: v1
kind: ResourceQuota
metadata:
   name: best-effort-quotas
spec:
    hard:
    pods: “5”
scopes:
    -
     BestEffort
```

Berkas definisi ditampilkan di editor vi pada Gambar 12.21.



Gambar 12.21 Berkas definisi ResourceQuota dengan cakupan BestEffort

Kita akan menggunakan kuota di namespace terpisah. Buat namespace 268ernama best-effort.

```
./kubectl create namespace best-effort
```

Buat ResourceQuota dari berkas definisi seperti yang ditampilkan pada Gambar 12.22.

```
./kubectl create -f best-effort-quotas.yaml –namespace=best-effort
```

Daftar kuota dan jelaskan:

```
./kubectl get quota --namespace=best-effort
./kubectl describe quota best-effort-quotas --namespace=best-effort
```

Kuota cakupan BestEffort dibuat, dicantumkan, dan dijelaskan seperti yang ditunjukkan pada Gambar 12.22.

```
core@ip-10-0-0-50 ~ $ ./kubectl create namespace best-effort
namespace "best-effort" created
core@ip-10-0-0-50 ~ $ ./kubectl create -f  best-effort-quotas.yaml --namespace=b
est-effort
resourcequota "best-effort-quotas" created
core@ip-10-0-0-50 ~ $ ./kubectl get quota --namespace=best-effort
NAME                 AGE
best-effort-quotas   13s
core@ip-10-0-0-50 ~ $ ./kubectl describe quota best-effort-quotas --namespace=be
st-effort
Name:           best-effort-quotas
Namespace:      best-effort
Scopes:         BestEffort
 * Matches all pods that have best effort quality of service.
Resource        Used    Hard
--------        ----    ----
pods            0       5


core@ip-10-0-0-50 ~ $
```

Gambar 12.22 Membuat Dan Menjelaskan Resourcequota Dengan Cakupan Besteffort Dalam Namespace Best-Effort

Dengan menggunakan file definisi RC yang sama mysql.yaml, buat RC dan daftarkan tiga pod seperti yang ditunjukkan pada Gambar 12.23.

```
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql.yaml --namespace=best-effort
replicationcontroller "mysql-rc" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide --namespace=best-effort
NAME             READY     STATUS    RESTARTS   AGE       IP          NODE
mysql-rc-fcrcq   1/1       Running   0          16s       10.2.98.7   ip-10-0-0-
181.ec2.internal
mysql-rc-i9ynz   1/1       Running   0          16s       10.2.47.6   ip-10-0-0-
182.ec2.internal
mysql-rc-s55kd   1/1       Running   0          16s       10.2.26.8   ip-10-0-0-
180.ec2.internal
core@ip-10-0-0-50 ~ $
core@ip-10-0-0-50 ~ $
```

Gambar 12.23 Membuat RC Dan Mencantumkan Pod

Skala RC menjadi lima pod, yang juga merupakan batas keras. RC diskalakan seperti yang ditunjukkan pada Gambar 12.24.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale rc mysql-rc --replicas=5
replicationcontroller "mysql-rc" scaled
core@ip-10-0-0-50 ~ $ ./kubectl scale rc mysql-rc --replicas=5 --namespace=best-
effort
replicationcontroller "mysql-rc" scaled
```

Gambar 12.24 Skala Replika Ke Batas Keras 5

Skala RC menjadi enam pod, yang akan melampaui batas keras. RC diskalakan seperti yang ditunjukkan pada Gambar 12.25.

```
core@ip-10-0-0-50 ~ $ ./kubectl scale rc mysql-rc --replicas=6 --namespace=best-
effort
replicationcontroller "mysql-rc" scaled
```

Gambar 12.25 Skala Replika Untuk Melampaui Batas Keras 5

Jelaskan RC, dan nilai Replika dicantumkan sebagai 6 saat ini / 6 yang diinginkan, seperti yang ditunjukkan pada Gambar 12.26. Meskipun batas keras pada jumlah pod terlampaui, pod tambahan dijadwalkan karena cakupannya ditetapkan ke BestEffort.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe rc --namespace=best-effort
Name:           mysql-rc
Namespace:      best-effort
Image(s):       mysql
Selector:       app=mysql-app,deployment=v1
Labels:         app=mysql-app
Replicas:       6 current / 6 desired
Pods Status:    6 Running / 0 Waiting / 0 Succeeded / 0 Failed
No volumes.
Events:
  FirstSeen     LastSeen        Count   From                            Subobjec
tPath   Type            Reason                  Message
  ---------     --------        -----   ----                            --------
-----   --------        ------                  -------
  7m            7m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-s55kd
  7m            7m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-fcrcq
  7m            7m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-i9ynz
  5m            5m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-5cnmg
  5m            5m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-dnkuf
  1m            1m              1       {replication-controller }               N
ormal           SuccessfulCreate        Created pod: mysql-rc-739se


core@ip-10-0-0-50 ~ $
```

Gambar 12.26 Replika Melampaui Batas Yang Ditetapkan Karena Cakupannya Adalah Besteffort

Ringkasan

Dalam bab ini, kami memperkenalkan kuota sumber daya, spesifikasi untuk membatasi alokasi sumber daya tertentu ke namespace tertentu dengan tujuan mendistribusikan sumber daya secara adil dan bersama. Kuota dapat ditetapkan pada sumber daya komputasi dan objek. Dalam bab berikutnya, kami akan membahas penskalaan otomatis.

BAB 13
MENGGUNAKAN PENSKALAAN OTOMATIS

13.1 PENSKALAAN OTOMATIS POD DI KUBERNETES

Memulai pod baru terkadang diperlukan dalam kluster Kubernetes, misalnya, untuk memenuhi persyaratan peningkatan beban. Pengontrol replikasi memiliki kemampuan untuk memulai ulang kontainer, yang sebenarnya memulai kontainer pengganti, jika kontainer dalam pod gagal.

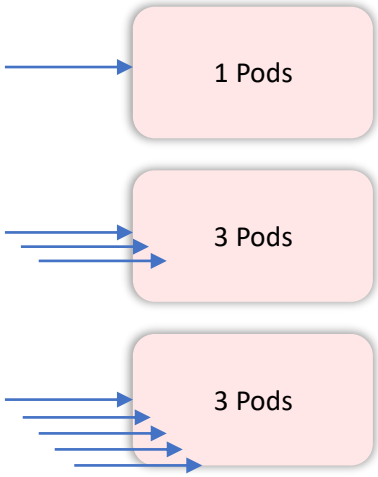
Masalah

Beban kluster bersifat variabel, dan bergantung pada persyaratan penggunaan aplikasi, beban kluster dapat meningkat atau menurun. Salah satu manfaat pengelola kluster Kubernetes adalah kemampuan untuk menskalakan kluster kontainer (pod) sesuai kebutuhan. Jika beban yang diharapkan lebih besar, pengguna dapat meningkatkan skala (menambah jumlah pod) dan jika beban yang diharapkan lebih kecil, pengguna dapat menurunkan skala (mengurangi jumlah pod).

Namun, model penskalaan yang diintervensi pengguna hanya cocok untuk pengembangan dan kluster skala kecil. Untuk kluster tingkat produksi yang bebannya tidak dapat diprediksi dan ketersediaan tinggi merupakan persyaratan, penskalaan yang diinisiasi pengguna mungkin tidak tepat waktu atau proporsional dengan persyaratan beban.

Solusi

Untuk kluster tingkat produksi, Kubernetes menyediakan pola desain manajemen penskalaan otomatis. Penskalaan otomatis didasarkan pada pola konfigurasi yang mudah berubah.



Gambar 13.1 Meningkatkan Beban Akan Meningkatkan Jumlah Pod

Penskalaan otomatis pod horizontal (HPA) dapat dibuat dengan jumlah pod minimum dan maksimum yang telah dikonfigurasi sebelumnya untuk menskalakan kluster. Saat beban meningkat pada kluster yang sedang berjalan, HPA secara otomatis meningkatkan jumlah pod

secara proporsional dengan persyaratan beban hingga jumlah pod maksimum yang dikonfigurasi, seperti yang ditunjukkan pada Gambar 13-1, dan saat beban menurun, HPA mengurangi jumlah pod secara proporsional tanpa campur tangan pengguna.

HPA memiliki dua manfaat utama dibandingkan penskalaan yang dilakukan pengguna: penskalaan dilakukan secara otomatis, dan pod tambahan tidak terus berjalan dengan menghabiskan sumber daya yang dapat digunakan untuk aplikasi lain. Autoscaler dapat dibuat untuk pengontrol replikasi, set replika, atau penerapan. Autoscaler menggunakan heapster untuk mengumpulkan penggunaan CPU dari suatu sumber daya, yang menjadi dasar penentuan apakah lebih banyak atau lebih sedikit pod yang harus dijalankan. Penskalaan otomatis didasarkan pada penggunaan CPU target, yang menyiratkan bahwa penggunaan CPU dari suatu sumber daya seperti penerapan harus x%.

Ikhtisar

Dalam bab ini, kami akan menunjukkan penggunaan autoscaling. Bab ini membahas topik-topik berikut.

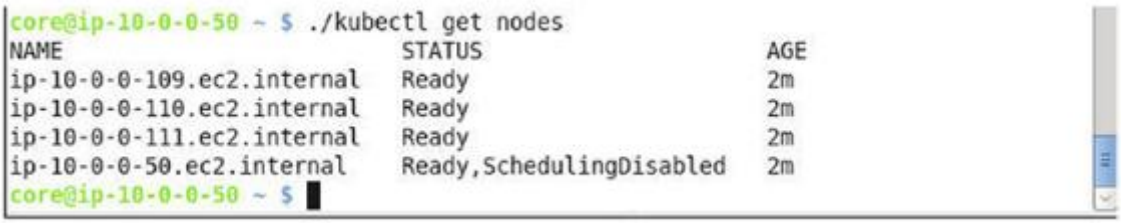
1. Menetapkan lingkungan
2. Menjalankan penerapan PHP Apache Server
3. Membuat layanan
4. Membuat autoscaler pod horizontal
5. Menambah beban

Menetapkan Lingkungan

Membuat kluster Kubernetes yang berjalan sebagai CoreOS AWS CloudFormation yang terdiri dari satu simpul pengontrol dan tiga simpul pekerja. Daftarkan node:

```
./kubectl get nodes
```

Node pengontrol tunggal dan node pekerja harus dicantumkan seperti yang ditunjukkan pada Gambar 13.2.



Gambar 13.2 Mencantumkan Node Kubernetes

Mencantumkan layanan di semua namespace seperti yang ditunjukkan pada Gambar 13.3. Layanan heapster, yang memantau penggunaan CPU, harus dicantumkan di namespace kube-system.

```
core@ip-10-0-0-50 ~ $ ./kubectl get svc --all-namespaces
NAMESPACE     NAME          CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
default       kubernetes    10.3.0.1     <none>        443/TCP         35m
kube-system   heapster      10.3.0.52    <none>        80/TCP          35m
kube-system   kube-dns      10.3.0.10    <none>        53/UDP,53/TCP   35m
core@ip-10-0-0-50 ~ $
```

Gambar 13.3 Mencantumkan Layanan Di Semua Namespace

Mencantumkan pod di semua namespace, dan pod heapster harus dicantumkan seperti yang ditunjukkan pada Gambar 13.4.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods --namespace=kube-system
NAME                                                READY     STATUS    RESTARTS
   AGE
heapster-v1.0.2-3151619174-v7r2x                    2/2       Running   0
   56m
kube-apiserver-ip-10-0-0-50.ec2.internal            1/1       Running   0
   56m
kube-controller-manager-ip-10-0-0-50.ec2.internal   1/1       Running   0
   56m
kube-dns-v11-1xc5y                                  4/4       Running   0
   55m
kube-proxy-ip-10-0-0-109.ec2.internal               1/1       Running   0
   55m
kube-proxy-ip-10-0-0-110.ec2.internal               1/1       Running   0
   55m
kube-proxy-ip-10-0-0-111.ec2.internal               1/1       Running   0
   56m
kube-proxy-ip-10-0-0-50.ec2.internal                1/1       Running   0
   56m
kube-scheduler-ip-10-0-0-50.ec2.internal            1/1       Running   0
   56m
mysql-rc-utaqt                                      1/1       Running   0
   4m
core@ip-10-0-0-50 ~ $
```

Gambar 13.4 Mencantumkan Pod Di Semua Namespace

Menjalankan Deployment Server PHP Apache

Pertama, kita perlu membuat sumber daya untuk diskalakan. Buat sumber daya deployment menggunakan image Docker gcr.io/google_containers/hpa-example. Atur permintaan CPU ke 200m.

```
./kubectl run php-apache --image=gcr.io/google_containers/hpa-example --requests=cpu=200m
```

Deployment yang disebut php-apache dibuat seperti yang ditunjukkan pada Gambar 13.5.

```
core@ip-10-0-0-50 ~ $ ./kubectl run php-apache --image=gcr.io/google_containers/
hpa-example --requests=cpu=200m
deployment "php-apache" created
```

Gambar 13.5 Membuat Deployment Untuk PHP Dan Apache

Daftar deployment:

```
./kubectl get deployment
```

Deployment php-apache harus dicantumkan seperti yang ditunjukkan pada Gambar 13.6.

```
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
php-apache   1         1         1            0           1m
core@ip-10-0-0-50 ~ $
```

Gambar 13.6 Mencantumkan Deployment

13.2 Membuat Layanan

Buat layanan bertipe LoadBalancer dengan mengekspos deployment pada port 80.

```
./kubectl exposed deployment php-apache --port=80 --
type=LoadBalancer
```

Sebuah layanan dibuat lalu dicantumkan seperti yang ditunjukkan pada Gambar 13.7.

```
core@ip-10-0-0-50 ~ $ ./kubectl expose deployment  php-apache --port=80 --type=L
oadBalancer
service "php-apache" exposed
core@ip-10-0-0-50 ~ $ ./kubectl get svc
NAME         CLUSTER-IP   EXTERNAL-IP        PORT(S)   AGE
kubernetes   10.3.0.1     <none>             443/TCP   3h
php-apache   10.3.0.172   a0c9a24254c76...   80/TCP    8s
core@ip-10-0-0-50 ~ $
```

Gambar 13.7 Membuat Layanan

Cantumkan pod, dan satu pod dicantumkan seperti yang ditunjukkan pada Gambar 13.8.

```
core@ip-10-0-0-50 ~ $ ./kubectl get svc
NAME         CLUSTER-IP   EXTERNAL-IP        PORT(S)   AGE
kubernetes   10.3.0.1     <none>             443/TCP   3h
php-apache   10.3.0.172   a0c9a24254c76...   80/TCP    30s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                          READY     STATUS    RESTARTS   AGE
php-apache-16292324-1hy2a     1/1       Running   0          2m
core@ip-10-0-0-50 ~ $
```

Gambar 13.8 Mencantumkan Pod

Kita akan memanggil layanan, yang menyediakan IP eksternal, untuk memberi beban pada penyebaran dan menguji apakah beban yang bervariasi membuat autoscaler mengubah jumlah pod secara proporsional. Dapatkan LoadBalancer Ingress untuk layanan:

```
./kubectl describe services php-apache | grep "LoadBalancer Ingress"
```

DNS publik tempat layanan dapat dipanggil tercantum, seperti yang ditunjukkan pada Gambar 13.9.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe services php-apache | grep "LoadBalance
r Ingress"
LoadBalancer Ingress:   a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.elb.
amazonaws.com
core@ip-10-0-0-50 ~ $
```

Gambar 13.9 Mendapatkan Loadbalancer Ingress

Panggil layanan menggunakan Load Balancer Ingress. Output seperti "OK!" dihasilkan seperti yang ditunjukkan pada Gambar 13.10; penyebaran php-apache dirancang hanya untuk pengujian dan output yang lebih rumit tidak dihasilkan.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe services php-apache | grep "LoadBalance
r Ingress"
LoadBalancer Ingress:   a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.elb.
amazonaws.com
core@ip-10-0-0-50 ~ $ curl a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.e
lb.amazonaws.com
OK!core@ip-10-0-0-50 ~ $
```

Gambar 13.10 Memanggil Loadbalancer Ingress

Membuat Horizontal Pod Autoscaler

Selanjutnya, kita akan membuat horizontal pod autoscaler untuk penyebaran. HPA dapat dibuat menggunakan salah satu dari dua metode yang tersedia:

1. Objek HorizontalPodAutoscaler
2. Perintah kubectl autoscale

Spesifikasi HorizontalPodAutoscaler menyediakan kolom yang ditunjukkan pada Tabel 13.1.

Tabel 13.1 Bidang Spesifikasi HorizontalPodAutoscaler

Bidang	Deskripsi
skalaTargetRef	Sumber daya target yang akan diskalakan. Dapat berupa Deployment, ReplicaSet, atau ReplicationController.
replikaMinimum	Jumlah minimum pod. Defaultnya adalah 1.
replikaMaks	Jumlah maksimum pod. Tidak boleh kurang dari minReplicas.
persentasePemanfaatanCPUTarget	Rata-rata target penggunaan CPU. Jika tidak ditentukan, kebijakan autoscaler default digunakan,

Perintah kubectl autoscale memiliki sintaksis berikut, yang pada dasarnya menyediakan pengaturan yang sama dengan spesifikasi.

```
kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--
min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags]
```

Beberapa opsi yang didukung oleh kubectl autoscale seperti yang dibahas dalam Tabel 13.2.

Tabel 13.2 Opsi Kubectl Autoscale

Opsi	Deskripsi	Nilai Default	Diperlukan (eksplisit atau default)
--cpu-persen	Rata-rata target penggunaan CPU di semua pod dalam sumber daya yang direpresentasikan sebagai persentase permintaan CPU. Jika tidak ditetapkan atau negatif, kebijakan penskalaan otomatis default digunakan.	-1	Ya
-f, --namafile	Nama file, direktori, atau URL untuk sumber daya yang akan diskalakan otomatis.	[]	Ya
--maks	Batas atas untuk jumlah pod.	-1	Ya
--min	Batas bawah untuk jumlah pod. Jika tidak ditentukan atau –ve, nilai default digunakan.	-1	Ya
--nama	Nama objek yang baru dibuat.	“”	Ya

Dengan menggunakan perintah kubectl autoscale, buatlah penskalaan otomatis pod horizontal. Tetapkan target penggunaan CPU ke 100% dan tetapkan jumlah minimum pod ke 3 dan jumlah maksimum pod ke 10.

```
./kubectl autoscale rc php-apache --cpu-percent=100 --min=3 --
max=10
```

Penerapan diskalakan otomatis dan HPA dibuat, seperti yang ditunjukkan pada Gambar 13.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl autoscale deployment php-apache --cpu-percent=10
0 --min=3 --max=10
deployment "php-apache" autoscaled
core@ip-10-0-0-50 ~ $
```

Gambar 13.11 Membuat Penskalaan Otomatis Pod Horizontal

Daftar HPA:

```
./kubectl get hpa
```

HPA tunggal harus dicantumkan seperti yang ditunjukkan pada Gambar 13.12. Kolom TARGET mencantumkan target penggunaan CPU, kolom CURRENT mencantumkan penggunaan CPU saat ini, kolom MINPODS mencantumkan jumlah minimum pod, dan MAXPODS mencantumkan jumlah maksimum pod. Karena penggunaan CPU memerlukan waktu untuk dipantau oleh heapster, kolom CURRENT menunjukkan nilai <menunggu>.

```
core@ip-10-0-0-50 ~ $ ./kubectl autoscale deployment php-apache --cpu-percent=10
0 --min=3 --max=10
deployment "php-apache" autoscaled
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                TARGET    CURRENT     MINPODS   MAXPODS   A
GE
php-apache   Deployment/php-apache    100%      <waiting>   3         10        1
9s
core@ip-10-0-0-50 ~ $
```

Gambar 13.12 Mencantumkan Penskalaan Otomatis Pod Horizontal

Daftar pod. Jumlah pod telah meningkat dari 1 pada penyebaran awal menjadi 3 (jumlah minimum pod dalam HPA) seperti yang ditunjukkan pada Gambar 13.13.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME                        READY     STATUS    RESTARTS   AGE
php-apache-16292324-1hy2a   1/1       Running   0          10m
php-apache-16292324-hpwqo   1/1       Running   0          3m
php-apache-16292324-wmg6k   1/1       Running   0          3m
core@ip-10-0-0-50 ~ $
```

Gambar 13.13. Jumlah Pod Diskalakan Ke Jumlah Minimum Pod Dalam Penskala Otomatis Pod Horizontal

Daftar HPA lagi, dan penggunaan CPU SAAT INI berada pada 0% karena tidak ada beban yang diberikan pada penyebaran seperti yang ditunjukkan pada Gambar 13.14.

```
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache    100%      0%        3         10        3m
core@ip-10-0-0-50 ~ $
```

Gambar 13.14 Pemanfaatan CPU Pada 0%

13.3 PENINGKATAN BEBAN

Selanjutnya, kami akan menunjukkan apa yang terjadi pada jumlah pod dan pemanfaatan CPU akibat peningkatan beban pada penerapan. Jalankan perintah berikut dengan mengganti LoadBalancer Ingress untuk memberikan beban pada penerapan:

```
curl <LoadBalancer Ingress>
```

Di terminal lain, dapatkan HPA. Jumlah pod tercantum sebagai 3 karena pemanfaatan CPU pada 22% berada di bawah target Pemanfaatan CPU sebesar 100%, seperti yang ditunjukkan pada Gambar 13.15.

```
[root@localhost ~]# ssh -i "docker.pem"  ec2-user@52.91.91.148
Last login: Sun Jul 17 19:27:48 2016 from d64-180-241-52.bchsia.telus.net

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/
5 package(s) needed for security, out of 14 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-0-196 ~]$ ssh -i "kubernetes-coreos.pem"  core@52.205.169.82
^C
[ec2-user@ip-10-0-0-196 ~]$ ssh -i "kubernetes-coreos.pem"  core@52.20.201.138
CoreOS stable (1068.6.0)
Last login: Sun Jul 17 23:40:14 2016 from 52.91.91.148
Update Strategy: No Reboots
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache    100%      22%       3         10        7m
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache    100%      22%       3         10        7m
core@ip-10-0-0-50 ~ $
```

Gambar 13.15 Pemanfaatan CPU Meningkat Hingga 22%

Jalankan loop perintah berikut dengan mengganti LoadBalancer Ingress untuk menambah beban pada penerapan:

```
while true; do curl <loadbalancer ingress>; done
```

Memanggil layanan dalam loop akan menghasilkan pesan yang sama berulang kali, seperti yang ditunjukkan pada Gambar 13.16.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe services php-apache | grep "LoadBalance
r Ingress"
LoadBalancer Ingress:    a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.elb.
amazonaws.com
core@ip-10-0-0-50 ~ $ curl a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.e
lb.amazonaws.com
OK!core@ip-10-0-0-50 ~ $ while true;
> do curl a0c9a24254c7611e68ee50a92211bd41-40787046.us-east-1.elb.amazonaws.com;
done
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!O
K!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK
!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Gambar 13.16 Memanggil Layanan Dalam Loop

Daftar penskala otomatis pod horizontal:

```
./kubectl get hpa
```

Nilai kolom CURRENT telah menjadi 224%, yang menunjukkan peningkatan beban pada penerapan, seperti yang ditunjukkan pada Gambar 13.17. Pemanfaatan CPU berada di atas target pemanfaatan CPU.

```
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                 TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache     100%      224%      3         10        12m
core@ip-10-0-0-50 ~ $
```

Gambar 13.17 Pemanfaatan CPU Saat Ini Di Atas Target

Jumlah pod masih 3 karena butuh beberapa saat agar jumlah pod meningkat dan klaster menjadi stabil, seperti yang ditunjukkan pada Gambar 13.18.

```
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
php-apache   3         3         3            3           15m
```

Gambar 13.18 Jumlah Pod Masih 3 Karena Butuh Beberapa Saat Bagi Klaster Untuk Stabil Saat Beban Ditingkatkan

Daftar penerapan setelah beberapa detik lagi, dan jumlah pod telah meningkat menjadi 5 seperti yang ditunjukkan pada Gambar 13.19. Autoscaler telah meningkatkan klaster dengan meningkatkan jumlah pod.

```
core@ip-10-0-0-50 ~ $ ./kubectl get hpa
NAME         REFERENCE                TARGET    CURRENT   MINPODS   MAXPODS   AGE
php-apache   Deployment/php-apache   100%      224%      3         10        12m
core@ip-10-0-0-50 ~ $ ./kubectl get deployment
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
php-apache   5         5         5            5           19m
core@ip-10-0-0-50 ~ $
```

Gambar 13.19 Jumlah Pod Bertambah Menjadi 5 Saat Beban Ditingkatkan

Ringkasan

Bab ini memperkenalkan penskalaan otomatis. Untuk mendemonstrasikan penskalaan otomatis, kami membuat penyebaran PHP Apache Server dan membuat layanan untuk penyebaran tersebut. Selanjutnya, kami membuat penskalaan otomatis pod horizontal dan menguji penskalaan otomatis dengan meningkatkan beban pada server Apache. Di bab berikutnya, kami akan membahas konfigurasi pencatatan log.

BAB 14
MENGONFIGURASI PENCATATAN LOG

14.1 POLA SIDECAR DAN ADAPTOR UNTUK LOG DI KUBERNETES

Pencatatan log adalah proses pengumpulan dan penyimpanan pesan log yang dihasilkan oleh berbagai komponen sistem (yang akan menjadi kluster Kubernetes) dan oleh aplikasi yang berjalan di kluster tersebut.

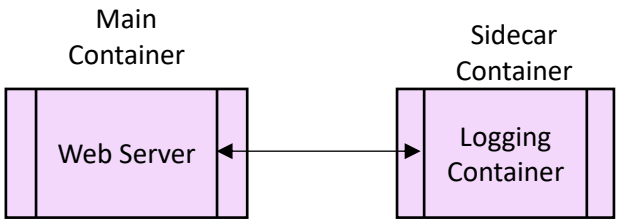
Masalah

Salah satu masalah yang terkait dengan pencatatan log adalah membatasi komponen dan aplikasi yang menghasilkan pesan pencatatan log. Masalah lainnya adalah memisahkan pencatatan log dari komponen/aplikasi. Komponen yang menghasilkan log dalam aplikasi Kubernetes akan menjadi kontainer dalam pod.

Solusi

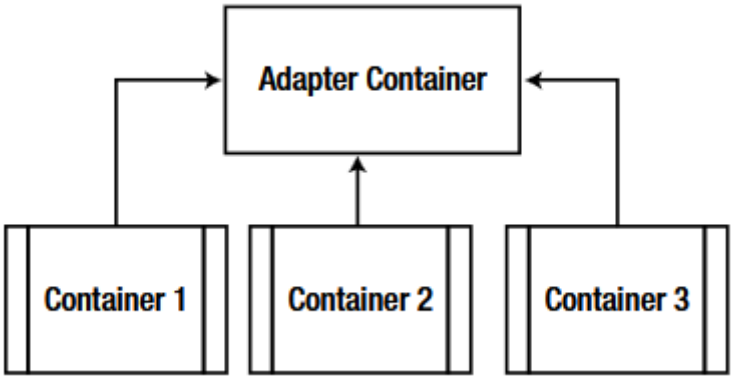
Kontainer berjalan secara terpisah pada Docker Engine dan merupakan objek tempat pola manajemen kontainer tunggal untuk melacak aplikasi yang berjalan dalam kontainer, termasuk pencatatan log khusus aplikasi, dapat digunakan. Demikian pula, pod adalah objek yang menghasilkan pesan pencatatan lognya sendiri, dan begitu pula pengontrol replikasi dan layanan. Pola desain yang diperkenalkan dalam publikasi terkini Design Patterns for Container-based Distributed Systems, oleh Brendan Burns dan David Oppenheimer (https://www.usenix.org/node/196347) adalah pola aplikasi multikontainer simpul tunggal yang disebut Pola Sidecar, yang dengannya kontainer utama.

Dapat dipasangkan dengan kontainer sidecar "logsaver", seperti yang ditunjukkan pada Gambar 14-1, untuk mengumpulkan log server web dari sistem berkas disk lokal dan mengalirkannya ke sistem penyimpanan kluster. Kontainer sidecar dapat dibuat layak karena kontainer pada mesin yang sama dapat berbagi volume disk lokal.



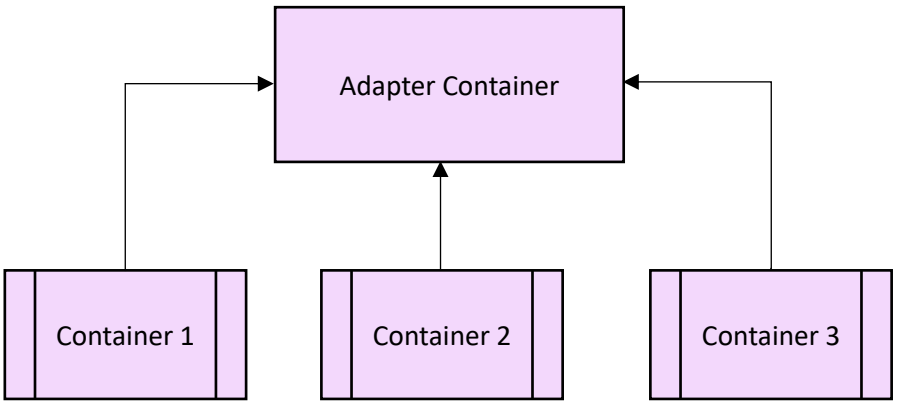
Gambar 14.1 Kontainer Sidecar Untuk Pencatatan

Pola aplikasi node tunggal dan beberapa kontainer lainnya, yang disebut Pola Adaptor, dapat digunakan untuk membuat kontainer Adaptor (Gambar 14.2) guna menyediakan antarmuka terpadu untuk menggabungkan log dari beberapa kontainer (atau pod) pada node yang menjalankan aplikasi yang sama atau berbeda.



Gambar 14.1 Kontainer Sidecar Untuk Pencatatan

Pola aplikasi node tunggal dan beberapa kontainer lainnya, yang disebut Pola Adaptor, dapat digunakan untuk membuat kontainer Adaptor (Gambar 14.2) guna menyediakan antarmuka terpadu untuk menggabungkan log dari beberapa kontainer (atau pod) pada node yang menjalankan aplikasi yang sama atau berbeda.



Gambar 14.2 Mencantumkan Node Kubernetes

Penggunaan kontainer khusus pencatatan menyediakan pemisahan masalah, prinsip desain modular.

Ikhtisar

Secara default, komponen Kubernetes seperti apiserver dan kubelet menggunakan pustaka pencatatan "glog". Untuk pencatatan tingkat klaster, tersedia berbagai opsi, dua di antaranya adalah sebagai berikut:

a. Pencatatan ke Google Cloud Pencatatan
b. Pencatatan ke Elasticsearch dan Kibana

Dalam bab ini, kita akan membahas cara mendapatkan log pod/kontainer tunggal dan juga pencatatan tingkat klaster dengan Elasticsearch dan Kibana. Prosedur untuk menggunakan pencatatan tingkat klaster dengan Elasticsearch dan Kibana adalah sebagai berikut.

1. Jalankan Elasticsearch.
2. Jalankan Replication Controller tempat log akan dikumpulkan.

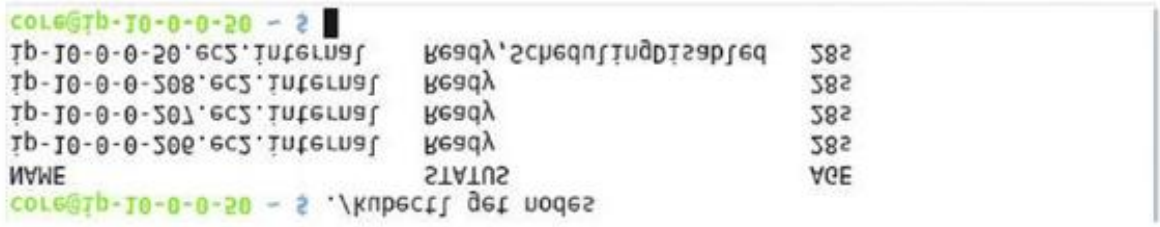
3. Jalankan Fluentd dan Elasticsearch untuk mengumpulkan log.
4. Jalankan Kibana untuk melihat log.

Bab ini membahas topik-topik berikut:

- Menetapkan lingkungan
- Mendapatkan log yang dihasilkan oleh logger default
- File log Docker
- Logging tingkat kluster dengan Elasticsearch dan Kibana
- Memulai Elastic Search
- Memulai pengontrol replikasi
- Memulai Fluentd dan Elasticsearch untuk mengumpulkan log
- Memulai Kibana

Menetapkan Lingkungan

Buat klaster Kubernetes menggunakan AWS CloudFormation berbasis CoreOS. Cantumkan node dengan kubectl get nodes. Node pengontrol dan pekerja harus dicantumkan; kami telah menggunakan klaster node pengontrol tunggal, tiga pekerja seperti yang ditunjukkan pada Gambar 14.3.



Gambar 14.3 Mencantumkan Node Kubernetes

14.2 MENDAPATKAN LOG YANG DIHASILKAN OLEH LOGGER DEFAULT

Log yang dihasilkan oleh pod yang sedang berjalan dapat diperoleh dengan perintah kubectl logs <POD>. Jika pod memiliki lebih dari satu kontainer, log untuk kontainer tertentu dapat diperoleh dengan perintah kubectl logs <POD> <kontainer>. Kubernetes melakukan rotasi log, dan hanya log terbaru yang tersedia untuk kubectl logs. Pertama, buat pod sampel untuk mendapatkan log. Gunakan daftar berikut untuk membuat file definisi pod counter-pod.yaml; pod menghasilkan pesan menggunakan penghitung.

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: my-counter
```

```
    image: busybox
    command:
    - sh
    - -c
    - while true; do echo $(date); sleep 1; done
args:
  - bash
  - -c
  - "for ((i = 0; i++; )); do echo \"$i: $(date)\"; sleep 1;
done"
image: "ubuntu:14.04"
name: count
```

Berkas definisi pod ditunjukkan dalam editor vi pada Gambar 14.4.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
    -
      args:
        - bash
        - "-c"
        - "for ((i = 0; ; i++)); do echo \"$i: $(date)\"; sleep 1; done"
      image: "ubuntu:14.04"
      name: count
~
```

Gambar 14.4 Berkas Definisi Pod Penghitung

Buat pod dari berkas definisi pod:

```
./kubectl create -f counter-pod.yaml
```

Pod penghitung dibuat. Daftarkan pod. Dapatkan log untuk penghitung pod:

```
./kubectl logs counter
```

Log tercantum seperti yang ditunjukkan pada Gambar 14.5.

```
core@ip-10-0-0-50 ~ $ sudo vi counter-pod.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f  counter-pod.yaml
pod "counter" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
counter   0/1       ContainerCreating   0          10s
core@ip-10-0-0-50 ~ $ ./kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
counter   0/1       ContainerCreating   0          23s
core@ip-10-0-0-50 ~ $ ./kubectl logs counter
0: Wed Jul 27 19:21:05 UTC 2016
1: Wed Jul 27 19:21:06 UTC 2016
2: Wed Jul 27 19:21:07 UTC 2016
3: Wed Jul 27 19:21:08 UTC 2016
4: Wed Jul 27 19:21:09 UTC 2016
5: Wed Jul 27 19:21:10 UTC 2016
6: Wed Jul 27 19:21:11 UTC 2016
7: Wed Jul 27 19:21:12 UTC 2016
8: Wed Jul 27 19:21:13 UTC 2016
9: Wed Jul 27 19:21:14 UTC 2016
10: Wed Jul 27 19:21:15 UTC 2016
11: Wed Jul 27 19:21:16 UTC 2016
core@ip-10-0-0-50 ~ $
```

Gambar 14.5 Membuat Pod Penghitung Dan Mendapatkan Log Pod

Berkas Log Docker

Secara default direktori berkas log kontainer Docker berada di direktori /var/lib/docker/containers. CD (ubah direktori) ke direktori /var/lib/docker/containers dan cantumkan berkas dan direktori. Direktori log tersedia untuk setiap kontainer Docker, seperti yang ditunjukkan pada Gambar 14.6.

```
core@ip-10-0-0-50 /var/lib/docker/containers $ sudo ls -l
total 88
drwx------. 2 root root 4096 Jul 27 18:58 041501f4cecbfa26c89c8369282fe13a7e01c8
1c28152d248e6575a22d20de24
drwx------. 3 root root 4096 Jul 27 18:57 1f58566a7617c7327808297e280ec803dae2bf
dbf1b23d29da52c2095dbe4e76
drwx------. 3 root root 4096 Jul 27 18:57 33fb20dfb63881369d02566eb2eb8a3ba5cc1b
e07daf0e70381aa13ffaed6e57
drwx------. 3 root root 4096 Jul 27 18:57 365da624bcd5590fea6f9fb5d7e1b4712cdc79
a3c3c62724a422ab0d064e6f6a
drwx------. 3 root root 4096 Jul 27 18:58 68bcd06c7c718a04d97a94bc58309f4832fef5
f7cf38bd54cb90b6a1dafbe64f
drwx------. 2 root root 4096 Jul 27 18:58 6ef0f0bdaac8ac53dfa9dfb6af853978365774
2932e25349c7f4f5277eaaf276
drwx------. 3 root root 4096 Jul 27 18:57 9479ac4a8bd76f560fa5ba6156f060264081ee
70d37ebc7c4b42c73370f5a313
drwx------. 2 root root 4096 Jul 27 18:59 d4dbad8595cf3f16c6e6b64b4f747c66cb2f27
92fcf31611ff3a3a2ab7379019
drwx------. 2 root root 4096 Jul 27 18:58 dda5e849d5063a2f9fa9c72aa51d76174bf2ed
```

Gambar 14.6 Direktori Log Kontainer Docker

Untuk mengakses direktori kontainer, kita perlu mengatur izin dengan chmod +x seperti yang

ditunjukkan pada Gambar 14.7. Kemudian CD ke direktori kontainer.

```
core@ip-10-0-0-50 /var/lib/docker/containers $ sudo chmod +x 041501f4cecbfa26c89
c8369282fe13a7e01c81c28152d248e6575a22d20de24
core@ip-10-0-0-50 /var/lib/docker/containers $ cd 041501f4cecbfa26c89c8369282fe1
3a7e01c81c28152d248e6575a22d20de24
```

Gambar 14.7 Mengatur Izin Pada Direktori Log Kontainer Docker

Cantumkan berkas dalam direktori kontainer seperti yang ditunjukkan pada Gambar 14.8. Berkas containerid-json.log adalah berkas log yang dibuat oleh kontainer.

```
core@ip-10-0-0-50 /var/lib/docker/containers/041501f4cecbfa26c89c8369282fe13a7e0
1c81c28152d248e6575a22d20de24 $ sudo ls -l
total 24
-rw-r-----. 1 root root 1030 Jul 27 18:58 041501f4cecbfa26c89c8369282fe13a7e01c8
1c28152d248e6575a22d20de24-json.log
-rw-r--r--. 1 root root 3135 Jul 27 18:58 config.v2.json
-rw-r--r--. 1 root root 1229 Jul 27 18:58 hostconfig.json
core@ip-10-0-0-50 /var/lib/docker/containers/041501f4cecbfa26c89c8369282fe13a7e0
1c81c28152d248e6575a22d20de24 $
```

Gambar 14.8 Mencantumkan Berkas Log Untuk Kontainer Docker

Buka file `-json.log` di editor `vi`. `Log JSON` akan ditampilkan seperti yang ditunjukkan pada Gambar 14.9.

```
core@ip-10-0-0-50:/var/lib/docker/containers/041501f4cecbfa26c89c83692 _ ❐
File  Edit  View  Search  Terminal  Help
{"log":"I0727 18:58:24.249275       1 server.go:200] Using iptables Proxier.\n",
"stream":"stderr","time":"2016-07-27T18:58:24.264395246Z"}
{"log":"I0727 18:58:24.249401       1 server.go:213] Tearing down userspace rule
s.\n","stream":"stderr","time":"2016-07-27T18:58:24.264432779Z"}
{"log":"I0727 18:58:24.331715       1 conntrack.go:36] Setting nf_conntrack_max
to 262144\n","stream":"stderr","time":"2016-07-27T18:58:24.353325373Z"}
{"log":"I0727 18:58:24.331784       1 conntrack.go:41] Setting conntrack hashsiz
e to 65536\n","stream":"stderr","time":"2016-07-27T18:58:24.353351574Z"}
{"log":"I0727 18:58:24.332047       1 conntrack.go:46] Setting nf_conntrack_tcp_
timeout_established to 86400\n","stream":"stderr","time":"2016-07-27T18:58:24.35
3358813Z"}
{"log":"E0727 18:58:24.332656       1 event.go:202] Unable to write event: 'Post
 http://127.0.0.1:8080/api/v1/namespaces/default/events: dial tcp 127.0.0.1:8080
: connection refused' (may retry after sleeping)\n","stream":"stderr","time":"20
16-07-27T18:58:24.35336476Z"}
~
~
~
~
~
~
~
<f4cecbfa26c89c8369282fe13a7e01c81c28152d248e6575a22d20de24-json.log" 6L, 1030C
```

Gambar 14.9 Log Kontainer Docker Dalam Format JSON

Log komponen sistem berada di direktori /var/log seperti yang ditunjukkan pada Gambar 14.10.

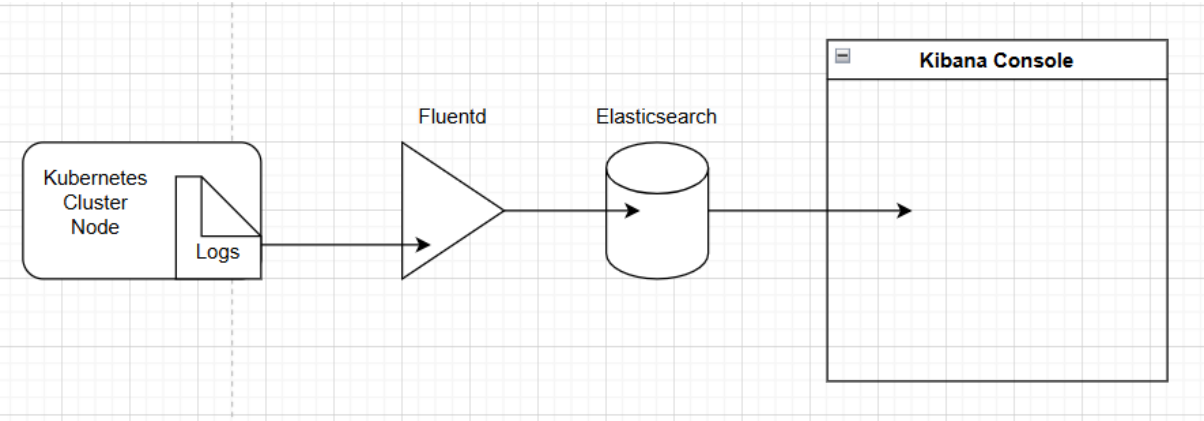
```
core@ip-10-0-0-50 ~ $ cd /var/log
core@ip-10-0-0-50 /var/log $ ls -l
total 48
-rw-------. 1 root utmp                0 Jul 27 18:52 btmp
-rw-r--r--. 1 root root                0 Jul 27 18:52 faillog
drwxr-sr-x. 4 root systemd-journal  4096 Jul 27 18:51 journal
-rw-r--r--. 1 root root           146292 Jul 27 19:11 lastlog
drwx------. 2 root root             4096 Jul 18 06:27 sssd
-rw-------. 1 root root            32064 Jul 27 19:11 tallylog
-rw-rw-r--. 1 root utmp             2688 Jul 27 19:11 wtmp
core@ip-10-0-0-50 /var/log $
```

Gambar 14.10 Log Komponen Sistem

14.3 PENCATATAN LOG TINGKAT KLASTER DENGAN ELASTICSEARCH DAN KIBANA

Pencatatan log tingkat klaster mengumpulkan log keluaran standar dan log galat standar aplikasi yang berjalan di kontainer. Untuk menggabungkan file log aplikasi yang berjalan di dalam kontainer, agregator Fluentd dapat digunakan. Di bagian ini, kita akan mengonfigurasi dan menggunakan pencatatan log tingkat klaster dengan Fluentd, Elasticsearch, dan Kibana. Fluentd adalah pengumpul data sumber terbuka untuk lapisan pencatatan terpadu.

Pencatatan terpadu menyiratkan bahwa Fluentd memisahkan sumber data dari sistem backend. Sumber data untuk contoh tersebut adalah log yang dihasilkan di klaster Kubernetes, dan backendnya adalah Elasticsearch. Elasticsearch adalah mesin pencari RESTful berorientasi dokumen, terdistribusi, dan sangat tersedia yang dirancang untuk lingkungan cloud dan dibangun di atas Lucene. Kibana adalah dasbor analitik dan pencarian sumber terbuka untuk Elasticsearch dan diakses dari browser web. Tiga komponen pencatatan tingkat kluster ditunjukkan pada Gambar 14.11.



Gambar 14.11 Komponen Pencatatan Log Tingkat Kluster

Untuk mengonfigurasi pencatatan log, gunakan prosedur berikut:

1. Jalankan pengontrol replikasi MySQL dan pod.
2. Jalankan layanan Elasticsearch.
3. Jalankan Fluentd.
4. Jalankan Kibana.
5. Akses log di KIbana.

Bagian berikut membahas setiap langkah sebelumnya secara terperinci.

Memulai Pengontrol Replikasi

Untuk membuat beberapa log aplikasi di pod, kita akan memulai pengontrol replikasi contoh. Buat file definisi RC untuk kontainer berbasis citra Docker MySQL. RC dibuat di namespace kube-system.

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: mysqlapp
  name: mysql-rc
  namespace: kube-system
spec:
  replicas: 3
  selector:
    app: mysqlapp
  template:
    metadata:
      labels:
        app: mysqlapp
spec:
  containers:
  - env:
    - name: MYSQL_ROOT_PASSWORD
      value: mysql
    image: mysql
    name: mysql
    ports:
    - containerPort: 3306
```

Berkas definisi RC ditampilkan dalam editor vi pada Gambar 14.12.

```
---
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    app: mysqlapp
  name: mysql-rc
  namespace: kube-system
spec:
  replicas: 3
  selector:
    app: mysqlapp
  template:
    metadata:
      labels:
        app: mysqlapp
    spec:
      containers:
        -
          env:
            -
              name: MYSQL_ROOT_PASSWORD
              value: mysql
          image: mysql
          name: mysql
          ports:
            -
              containerPort: 3306

~
:wq
```

Gambar 14.12 Berkas Definisi Pengontrol Replikasi

Buat RC dengan kubectl create menggunakan file definisi:

```
./kubectl create -f mysql-rc.yaml
```

Daftar RC:

```
./kubectl get rc –namespace=kube-system
```

RC mysql-rc harus dicantumkan seperti yang ditunjukkan pada Gambar 14.13.

```
core@ip-10-0-0-50 ~ $ sudo vi mysql-rc.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f mysql-rc.yaml
replicationcontroller "mysql-rc" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc --namespace=kube-system
NAME                      DESIRED   CURRENT   AGE
elasticsearch-logging-v1  2         2         21m
kibana-logging-v1         1         1         9m
kube-dns-v11              1         1         26m
mysql-rc                  3         3         11s
```

Gambar 14.13 Membuat Dan Mencantumkan Pengontrol Replikasi Di Namespace Kube-System

Daftar pod di namespace kube-system, dan pod mysql harus dicantumkan seperti yang ditunjukkan pada Gambar 14.14.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods --namespace=kube-system
NAME                                                READY     STATUS
  RESTARTS   AGE
elasticsearch-logging-v1-3eqmk                      1/1       Running
  0          22m
elasticsearch-logging-v1-mnjy8                      1/1       Running
  0          22m
fluentd-elasticsearch                               1/1       Running
  0          13m
heapster-v1.0.2-3151619174-kl0ek                    2/2       Running
  0          25m
kibana-logging-v1-sdb3m                             1/1       Running
  0          9m
kube-apiserver-ip-10-0-0-50.ec2.internal            1/1       Running
  0          26m
kube-controller-manager-ip-10-0-0-50.ec2.internal   1/1       Running
  0          26m
kube-dns-v11-or90k                                  4/4       Running
  0          25m
kube-proxy-ip-10-0-0-206.ec2.internal               1/1       Running
  0          25m
kube-proxy-ip-10-0-0-207.ec2.internal               1/1       Running
```

Gambar 14.14 Mencantumkan Pod Di Namespace Kube-System

Memulai Elastic Search

Di bagian ini, kita akan membuat pengontrol replikasi dan layanan untuk Elasticsearch menggunakan image Docker gcr.io/google_containers/elasticsearch:1.9. Buat file definisi RC es-controller.yaml dan salin daftar berikut ke dalamnya.

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    version: v1
  name: elasticsearch-logging-v1
  namespace: kube-system
spec:
  replicas: 2
  selector:
    k8s-app: elasticsearch-logging
    version: v1
  template:
    metadata:
```

```
      labels:
        k8s-app: elasticsearch-logging
        kubernetes.io/cluster-service: "true"
        version: v1
spec:
  containers:
  - image: "gcr.io/google_containers/elasticsearch:1.9"
    name: elasticsearch-logging
    ports:
    - containerPort: 9200
      name: db
      protocol: TCP
    - containerPort: 9300
      name: transport
      protocol: TCP
    resources:
      limits:
        cpu: "0.1"
      requests:
        cpu: "0.1"
    volumeMounts:
    - mountPath: /data
      name: es-persistent-storage
volumes:
- name: es-persistent-storage
  emptyDir: {}
```

Buat RC menggunakan file definisi:

```
./kubectl create -f es-controller.yaml
```

Buat file definisi layanan es-service.yaml untuk Elasticsearch RC. Paparkan layanan di port 9200. Label pemilih harus sesuai dengan label di pod.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: Elasticsearch
  name: elasticsearch-logging
  namespace: kube-system
```

```
spec:
  ports:
  - port: 9200
    protocol: TCP
    targetPort: db
  selector:
    k8s-app: elasticsearch-logging
```

Buat layanan dari berkas definisi:

```
./kubectl create -f es-service.yaml
```

RC, pod, dan layanan untuk Elasticsearch dibuat dalam namespace kube-system dan dapat dicantumkan dan dijelaskan seperti yang ditunjukkan pada Gambar 14.15.

```
core@ip-10-0-0-50 ~ $ sudo vi es-service.yaml
core@ip-10-0-0-50 ~ $ sudo vi es-controller.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f  es-controller.yaml
replicationcontroller "elasticsearch-logging-v1" created
core@ip-10-0-0-50 ~ $ ./kubectl create -f  es-service.yaml
service "elasticsearch-logging" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc --namespace=kube-system
NAME                     DESIRED   CURRENT   AGE
elasticsearch-logging-v1   2         2         32s
kube-dns-v11               1         1         4m
core@ip-10-0-0-50 ~ $ ./kubectl get svc --namespace=kube-system
NAME                    CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
elasticsearch-logging   10.3.0.79    <none>        9200/TCP        34s
heapster                10.3.0.18    <none>        80/TCP          5m
kube-dns                10.3.0.10    <none>        53/UDP,53/TCP   5m
core@ip-10-0-0-50 ~ $ ./kubectl get pods --namespace=kube-system
NAME                                                READY     STATUS
  RESTARTS   AGE
elasticsearch-logging-v1-3eqmk                      0/1       ContainerCreating
  0          1m
elasticsearch-logging-v1-mnjy8                      0/1       ContainerCreating
  0          1m
heapster-v1.0.2-3151619174-kl0ek                    2/2       Running
  0          4m
```

Gambar 14.15 Membuat RC Dan Layanan Untuk Elasticsearch

Jelaskan layanan Elasticsearch untuk mencantumkan titik akhir layanan seperti yang ditunjukkan pada Gambar 14.16.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc elasticsearch-logging --namespace=k
ube-system
Name:                   elasticsearch-logging
Namespace:              kube-system
Labels:                 k8s-app=elasticsearch-logging
                        kubernetes.io/cluster-service=true
                        kubernetes.io/name=Elasticsearch
Selector:               k8s-app=elasticsearch-logging
Type:                   ClusterIP
IP:                     10.3.0.79
Port:                   <unset> 9200/TCP
Endpoints:              10.2.15.2:9200,10.2.69.3:9200
Session Affinity:       None
No events.

core@ip-10-0-0-50 ~ $
```

Gambar 14.16 Menjelaskan Layanan Elasticsearch

Panggil titik akhir layanan untuk memanggil layanan Elasticsearch, seperti yang ditunjukkan pada Gambar 14.17.

```
core@ip-10-0-0-50 ~ $ curl 10.2.15.2:9200
{
  "status" : 200,
  "name" : "Jason",
  "cluster_name" : "kubernetes-logging",
  "version" : {
    "number" : "1.5.2",
    "build_hash" : "62ff9868b4c8a0c45860bebb259e21980778ab1c",
    "build_timestamp" : "2015-04-27T09:21:06Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}
core@ip-10-0-0-50 ~ $ curl 10.2.69.3:9200
{
  "status" : 200,
  "name" : "Maestro",
  "cluster_name" : "kubernetes-logging",
  "version" : {
    "number" : "1.5.2",
    "build_hash" : "62ff9868b4c8a0c45860bebb259e21980778ab1c",
    "build_timestamp" : "2015-04-27T09:21:06Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}
core@ip-10-0-0-50 ~ $
```

Gambar 14.17 Memanggil Pengujian Untuk Layanan Elasticsearch

Info kluster kubectl harus mencantumkan Elasticsearch sebagai berjalan, seperti yang ditunjukkan pada Gambar 14.18.

Gambar 14.18 Layanan Elasticsearch Tercantum Sebagai Berjalan Di Info Kluster Kubernetes

14.4 MEMULAI FLUENTD UNTUK MENGUMPULKAN LOG

Setelah memulai sumber data (aplikasi kluster Kuebrnetes) dan basis data backend (Elasticsearch), selanjutnya kita akan memulai lapisan pemersatu antara keduanya, Fluentd. Buat file definisi pod fluentd-es.yaml untuk Fluentd dan salin daftar berikut ke file definisi. Citra Docker fabric8/fluentd-kubernetes:v1.9 digunakan dalam kontainer pod.

URL titik akhir Elasticsearch dan port untuk berinteraksi juga ditentukan. Pod memasang direktori log sistem /var/log dan direktori kontainer Docker /var/lib/docker/containers dari jalur host. Volume bertipe hostPath digunakan. Direktori log yang berbeda juga dapat dipasang.

```
apiVersion: v1
kind: Pod
metadata:
  name: fluentd-elasticsearch
spec:
  containers:
  - env:
    - name: ELASTICSEARCH_HOST
      value: "10.2.15.2"
    - name: ELASTICSEARCH_PORT
      value: "9200"
    image: "fabric8/fluentd-kubernetes:v1.9"
    name: fluentd-elasticsearch
    resources:
      limits:
        cpu: "0.1"
    securityContext:
      privileged: true
    volumeMounts:
    - name: varlog
      mountPath: /var/log
    - name: varlibdockercontainers
```

```
      mountPath: /var/lib/docker/containers
      readOnly: true
  volumes:
    - name: varlog
      emptyDir: {}
    - name: varlibdockercontainers
      hostPath:
        path: /var/lib/docker/containers
volumes:
- name: varlog
  hostPath:
    path: /var/log
- name: varlibdockercontainers
  hostPath:
    path: /var/lib/docker/containers
```

Berkas definisi pod ditampilkan di editor vi pada Gambar 14.19.

Gambar 14.19 Berkas Definisi Pod Untuk Fluentd

Buat pod untuk Fluentd:

```
./kubectl create -f fluentd-es.yaml
```

Pod dibuat di namespace kube-system seperti yang ditampilkan pada Gambar 14.20.

```
core@ip-10-0-0-50 ~ $ sudo vi fluentd-es.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f  fluentd-es.yaml
pod "fluentd-elasticsearch" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods --namespace=kube-system
NAME                                                READY     STATUS    RESTARTS
   AGE
elasticsearch-logging-v1-3eqmk                      1/1       Running   0
   8m
elasticsearch-logging-v1-mnjy8                      1/1       Running   0
   8m
fluentd-elasticsearch                               1/1       Running   0
   21s
heapster-v1.0.2-3151619174-kl0ek                    2/2       Running   0
   12m
kube-apiserver-ip-10-0-0-50.ec2.internal            1/1       Running   0
   12m
kube-controller-manager-ip-10-0-0-50.ec2.internal   1/1       Running   0
   12m
kube-dns-v11-or90k                                  4/4       Running   0
   12m
kube-proxy-ip-10-0-0-206.ec2.internal               1/1       Running   0
   12m
kube-proxy-ip-10-0-0-207.ec2.internal               1/1       Running   0
   12m
kube-proxy-ip-10-0-0-208.ec2.internal               1/1       Running   0
   12m
kube-proxy-ip-10-0-0-50.ec2.internal                1/1       Running   0
   12m
kube-scheduler-ip-10-0-0-50.ec2.internal            1/1       Running   0
   12m
core@ip-10-0-0-50 ~ $
```

Gambar 14.20 Membuat Dan Mencantumkan Pod Untuk Fluentd

14.5 MEMULAI KIBANA

Selanjutnya, kita akan memulai Kibana untuk melihat log. Buat file definisi RC kibana-rc.yaml dan salin daftar berikut ke file tersebut. Gambar kontainer untuk Kibana adalah gcr.io/google_containers/kibana:1.3. URL untuk Elasticsearch juga perlu ditentukan.

```
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    k8s-app: kibana-logging
    kubernetes.io/cluster-service: "true"
    version: v1
  name: kibana-logging-v1
  namespace: kube-system
spec:
  replicas: 1
  selector:
    k8s-app: kibana-logging
    version: v1
```

```
template:
  metadata: labels:
     k8s-app: kibana-logging
     kubernetes.io/cluster-service: "true"
apiVersion: v1
kind: ReplicationController
metadata:
  labels:
    k8s-app: kibana-logging
    kubernetes.io/cluster-service: "true"
    version: v1
  name: kibana-logging-v1
  namespace: kube-system
spec:
  replicas: 1
  selector:
    k8s-app: kibana-logging
    version: v1
  template:
    metadata:
      labels:
        k8s-app: kibana-logging
        kubernetes.io/cluster-service: "true"
        version: v1
    spec:
      containers:
      - name: kibana
        image: "gcr.io/google_containers/kibana:1.3"
        env:
        - name: ELASTICSEARCH_URL
          value: "http://10.2.15.2:9200"
        ports:
        - containerPort: 5601
          name: ui
          protocol: TCP
        resources:
          limits:
            cpu: "0.1"
          requests:
            cpu: "0.1"
```

Berkas definisi RC ditunjukkan dalam editor vi pada Gambar 14.21.

```
  replicas: 1
  selector:
    k8s-app: kibana-logging
    version: v1
  template:
    metadata:
      labels:
        k8s-app: kibana-logging
        kubernetes.io/cluster-service: "true"
        version: v1
    spec:
      containers:
        -
          env:
            -
              name: ELASTICSEARCH_URL
              value: "http://10.2.15.2:9200"
          image: "gcr.io/google_containers/kibana:1.3"
          name: kibana-logging
          ports:
            -
              containerPort: 5601
              name: ui
              protocol: TCP
          resources:
            limits:
              cpu: "0.1"
            requests:
              cpu: "0.1"

:wq
```

Gambar 14.21 Berkas Definisi Pengontrol Replikasi Untuk Kibana

Buat berkas definisi layanan kibana-service.yaml untuk RC dan salin daftar berikut ke berkas tersebut. Layanan Kibana ditampilkan di port 5601.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: Kibana
  name: kibana-logging
  namespace: kube-system
spec:
  ports:
  - port: 5601
    protocol: TCP
  selector:
    k8s-app: kibana-logging
```

Berkas definisi layanan dalam editor vi ditunjukkan pada Gambar 14.22.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: Kibana
  name: kibana-logging
  namespace: kube-system
spec:
  ports:
    -
      port: 5601
      protocol: TCP
  selector:
    k8s-app: kibana-logging
~
```

Gambar 14.22 Berkas Definisi Layanan Untuk Kibana

Buat RC untuk Kibana:

```
./kubectl create -f kibana-rc.yaml
```

Buat juga layanan Kibana:

```
./kubectl create -f kibana-service.yaml
```

RC dan layanan Kibana dibuat seperti yang ditunjukkan pada Gambar 14.23. Daftar RC dan layanan, yang ada di namespace
kube-system.

```
core@ip-10-0-0-50 ~ $ sudo vi kibana-rc.yaml
core@ip-10-0-0-50 ~ $ sudo vi kibana-service.yaml
core@ip-10-0-0-50 ~ $ ./kubectl create -f kibana-rc.yaml
replicationcontroller "kibana-logging-v1" created
core@ip-10-0-0-50 ~ $ ./kubectl create -f kibana-service.yaml
service "kibana-logging" created
core@ip-10-0-0-50 ~ $ ./kubectl get rc --namespace=kube-system
NAME                     DESIRED   CURRENT   AGE
elasticsearch-logging-v1   2         2         13m
kibana-logging-v1          1         1         50s
kube-dns-v11               1         1         17m
core@ip-10-0-0-50 ~ $ ./kubectl get svc --namespace=kube-system
NAME                    CLUSTER-IP   EXTERNAL-IP   PORT(S)         AGE
elasticsearch-logging   10.3.0.79    <none>        9200/TCP        13m
heapster                10.3.0.18    <none>        80/TCP          18m
kibana-logging          10.3.0.213   <none>        5601/TCP        1m
kube-dns                10.3.0.10    <none>        53/UDP,53/TCP   18m
core@ip-10-0-0-50 ~ $
```

Gambar 14.23 Membuat Dan Mencantumkan RC Dan Layanan Untuk Kibana

Jelaskan layanan untuk mendapatkan titik akhir layanan, yaitu 10.2.15.4:5601, seperti yang ditunjukkan pada Gambar 14.24.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc kibana-logging --namespace=kube-sys
tem
Name:                   kibana-logging
Namespace:              kube-system
Labels:                 k8s-app=elasticsearch-logging
                        kubernetes.io/cluster-service=true
                        kubernetes.io/name=Kibana
Selector:               k8s-app=kibana-logging
Type:                   ClusterIP
IP:                     10.3.0.213
Port:                   <unset> 5601/TCP
Endpoints:              10.2.15.4:5601
Session Affinity:       None
No events.

core@ip-10-0-0-50 ~ $
```

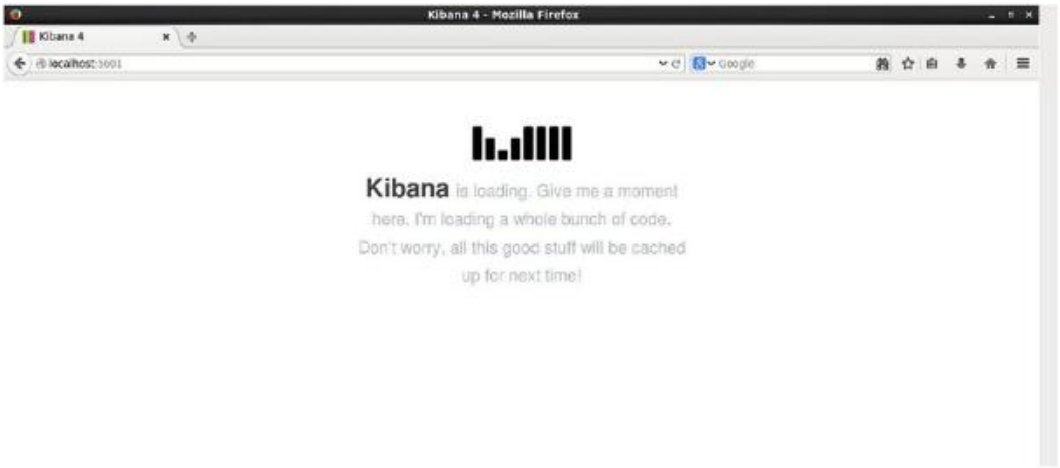
Gambar 14.24 Menjelaskan Layanan Pencatatan Kibana

Untuk mengakses Kibana Dashboard dari peramban web, atur penerusan porta dari mesin lokal. Pertama, kita perlu menyalin pasangan kunci untuk instans pengontrol CoreOS Kubernetes agar dapat melakukan SSH ke instans pengontrol untuk mengatur penerusan porta:

```
scp -i docker.pem ec2-user@ec2-54-208-177-36.compute-1.amazonaws.com:~/kubernetes-coreos.pem~/kubernetes-coreos.pem
ssh -i kubernetes-coreos.pem -f -nNT -L 5601:10.2.15.4:5601:5601 core@ec2-52-207-33-106. compute-1.amazonaws.com
```

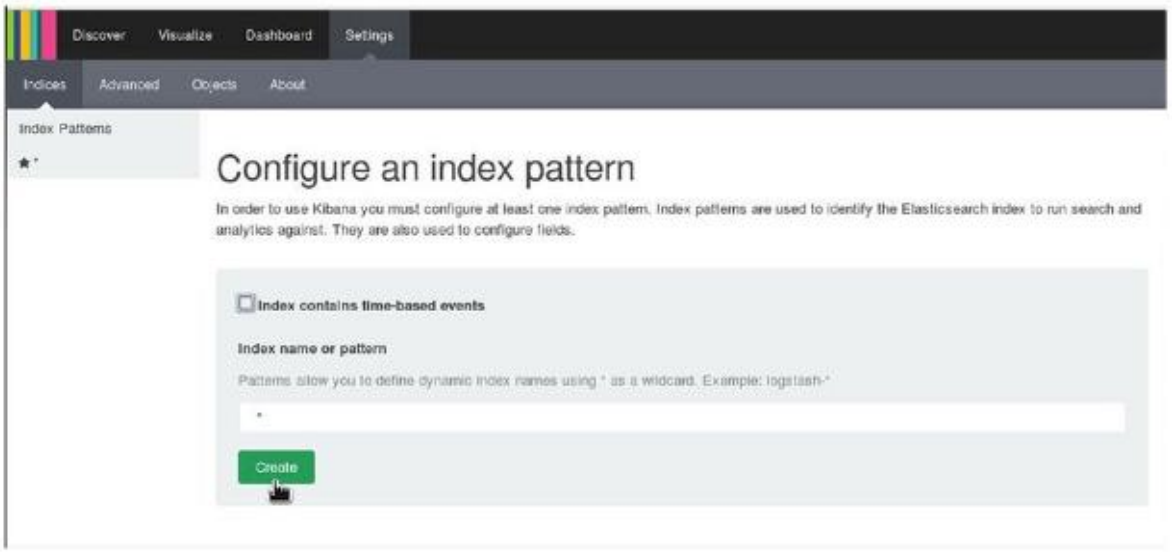
Penerusan porta telah diatur.

Akses Dasbor Kibana dari peramban di komputer lokal dengan URL http://localhost:5601. Dasbor Kibana dimulai, seperti yang ditunjukkan pada Gambar 14.25.



Gambar 14.25 Dasbor Kibana

Untuk menggunakan Kibana, rujuk dokumentasi Kibana. Pola indeks dapat dikonfigurasi untuk pencarian seperti yang ditunjukkan pada Gambar 14.26.



Gambar 14.26 Mengonfigurasi Pola Indeks

Log yang dikumpulkan dari kluster Kubernetes oleh Fluentd dan pesan log yang dihasilkan oleh Fluentd sendiri ditampilkan seperti yang ditunjukkan pada Gambar 14.27.

Gambar 14.27 Menampilkan Pesan Log Di Kibana

Kolom dapat dinavigasi dari daftar Kolom Populer seperti yang ditunjukkan pada Gambar 14.28.

Gambar 14.28 Kolom Populer Dalam Indeks

Info klaster Kubernetes juga harus mencantumkan Kibana sebagai tambahan pada layanan Elasticsearch seperti yang ditunjukkan pada Gambar 14.29.

```
core@ip-10-0-0-50 ~ $ ./kubectl cluster-info
Kubernetes master is running at http://localhost:8080
Elasticsearch is running at http://localhost:8080/api/v1/proxy/namespaces/kube-s
ystem/services/elasticsearch-logging
Heapster is running at http://localhost:8080/api/v1/proxy/namespaces/kube-system
/services/heapster
Kibana is running at http://localhost:8080/api/v1/proxy/namespaces/kube-system/s
ervices/kibana-logging
KubeDNS is running at http://localhost:8080/api/v1/proxy/namespaces/kube-system/
services/kube-dns

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
core@ip-10-0-0-50 ~ $
```

Gambar 14.29 Kibana Dicantumkan Sebagai Berjalan

Pod untuk MySQL, Elasticsearch, Fluentd, dan Kibana dicantumkan dalam namespace kube-system seperti yang ditunjukkan pada Gambar 14.30.

```
core@ip-10-0-0-50 ~ $ ./kubectl get pods --namespace=kube-system
NAME                                                READY     STATUS    RESTARTS
   AGE
elasticsearch-logging-v1-3eqmk                      1/1       Running   0
   22m
elasticsearch-logging-v1-mnjy8                      1/1       Running   0
   22m
fluentd-elasticsearch                               1/1       Running   0
   14m
heapster-v1.0.2-3151619174-kl0ek                    2/2       Running   0
   26m
kibana-logging-v1-sdb3m                             1/1       Running   0
   9m
kube-apiserver-ip-10-0-0-50.ec2.internal            1/1       Running   0
   27m
kube-controller-manager-ip-10-0-0-50.ec2.internal   1/1       Running   0
   27m
kube-dns-v11-or90k                                  4/4       Running   0
   26m
kube-proxy-ip-10-0-0-206.ec2.internal               1/1       Running   0
   26m
kube-proxy-ip-10-0-0-207.ec2.internal               1/1       Running   0
   26m
kube-proxy-ip-10-0-0-208.ec2.internal               1/1       Running   0
   26m
kube-proxy-ip-10-0-0-50.ec2.internal                1/1       Running   0
   27m
kube-scheduler-ip-10-0-0-50.ec2.internal            1/1       Running   0
   27m
mysql-rc-5jfdy                                      1/1       Running   0
   59s
mysql-rc-nnhmt                                      1/1       Running   0
```

Gambar 14.30 Pod Untuk Mysql, Elasticsearch, Fluentd, Dan Kibana

Ringkasan

Dalam bab ini, kami memperkenalkan pencatatan log, termasuk logger default dan berkas log Docker. Selanjutnya, kami mendemonstrasikan penggunaan pencatatan log tingkat kluster untuk mengumpulkan dan memantau log dengan Elasticsearch, Fluentd, dan Kibana. Dalam bab berikutnya, kami akan membahas penggunaan master ketersediaan tinggi dengan OpenShift.

BAB 15
MENGGUNAKAN MASTER HA DENGAN OPENSHIFT

15.1 KETERSEDIAAN TINGGI PADA KLUSTER KUBERNETES DENGAN OPENSHIFT

Platform as a Service (PaaS) adalah platform cloud tempat aplikasi dapat dikembangkan, dijalankan, dan dikelola hampir tanpa konfigurasi karena platform menyediakan infrastruktur aplikasi termasuk jaringan, penyimpanan, OS, middleware runtime, basis data, dan layanan dependensi lainnya. Kubernetes adalah pengelola kluster kontainer yang paling umum digunakan dan dapat digunakan sebagai fondasi untuk mengembangkan PaaS.

OpenShift adalah contoh PaaS. OpenShift Origin adalah platform aplikasi kontainer sumber terbuka yang menyediakan manajemen siklus hidup aplikasi secara penuh. OpenShift Origin menyediakan standarisasi melalui kontainerisasi. OpenShift menyertakan pengelola kluster Kubernetes tertanam untuk mengatur kontainer Docker.

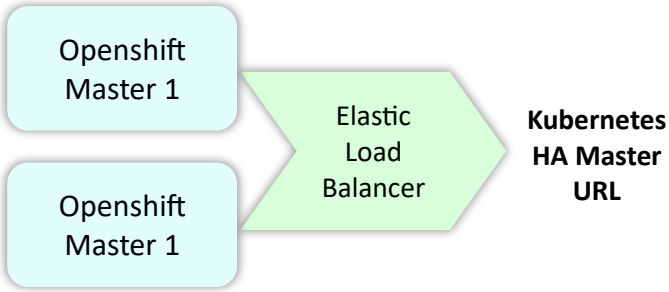
Masalah

Satu master dalam kluster Kubernetes adalah satu titik kegagalan (SPOF). Kegagalan node tempat pengontrol master berjalan menyebabkan kluster Kubernetes gagal dan tidak dapat diakses. Pada saat mengembangkan buku ini, CoreOS tidak mendukung pengontrol ketersediaan tinggi (HA) secara bawaan. CoreOS menyediakan Grup Penskalaan Otomatis dan Konfigurasi Peluncuran sehingga jika node pekerja dihentikan atau gagal, node pekerja lain akan dimulai.

Solusi

Menggunakan master ketersediaan tinggi (HA), yang terdiri dari lebih dari satu master Kubernetes dengan failover yang dikonfigurasi, menyediakan ketersediaan tinggi untuk kluster, dan kegagalan satu master tidak menyebabkan kluster gagal. Alternatif untuk kluster berbasis Linux CoreOS adalah menggunakan platform OpenShift, yang dapat mengonfigurasi beberapa node master. Amazon Elastic Load Balancer dapat digunakan untuk menyediakan failover dari node pengontrol yang berjalan di satu zona ke node pengontrol yang berjalan di zona lain dengan Arsitektur Ketersediaan Tinggi Antar-Zona AWS.

AWS tidak mendukung Arsitektur Ketersediaan Tinggi Antar-Wilayah AWS untuk Elastic Load Balancer. Master HA adalah pola desain Kubernetes yang diimplementasikan hanya oleh beberapa alat, seperti PaaS OpenShift berbasis Kubernetes. OpenShift HA Master didasarkan pada pola arsitektur Aktif-Aktif, di mana kedua node master aktif dan menyediakan redundansi. Elastic Load Balancer digunakan untuk mendistribusikan beban di kedua node master. Server API pengontrol master HA diekspos pada Load Balancer, seperti yang ditunjukkan pada Gambar 15.1, dan tidak langsung pada node master.



Gambar 15.1 Master Openshift HA

Gambaran Umum

Kluster OpenShift Origin tingkat produksi yang umum akan terdiri dari master ketersediaan tinggi. Dalam bab ini, kita akan membahas kluster OpenShift Origin master ketersediaan tinggi tersebut. Tahapannya adalah sebagai berikut:

- Mengatur lingkungan
- Memasang kredensial
- Memasang pengelola jaringan
- Memasang OpenShift Ansible
- Mengonfigurasi Ansible
- Menjalankan Ansible Playbook
- Menguji kluster
- Menguji HA

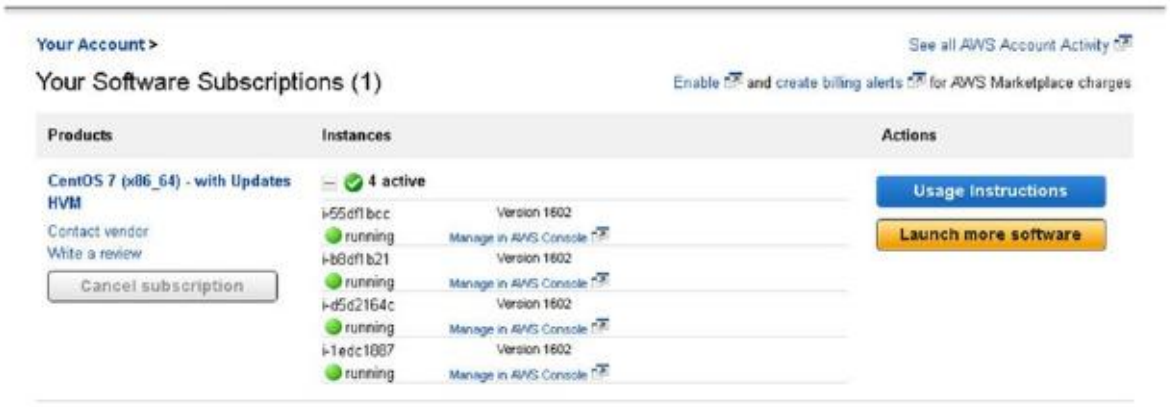
Mengatur Lingkungan

Kluster OpenShift yang akan kita buat terdiri dari instans EC2 berikut.

1 instans Ubuntu untuk OpenShift Ansible
2 CentOS 7 untuk OpenShift Masters
1 CentOS 7 untuk HAProxy
1 CentOS 7 untuk OpenShift Worker
1 Centos 7 untuk etcd

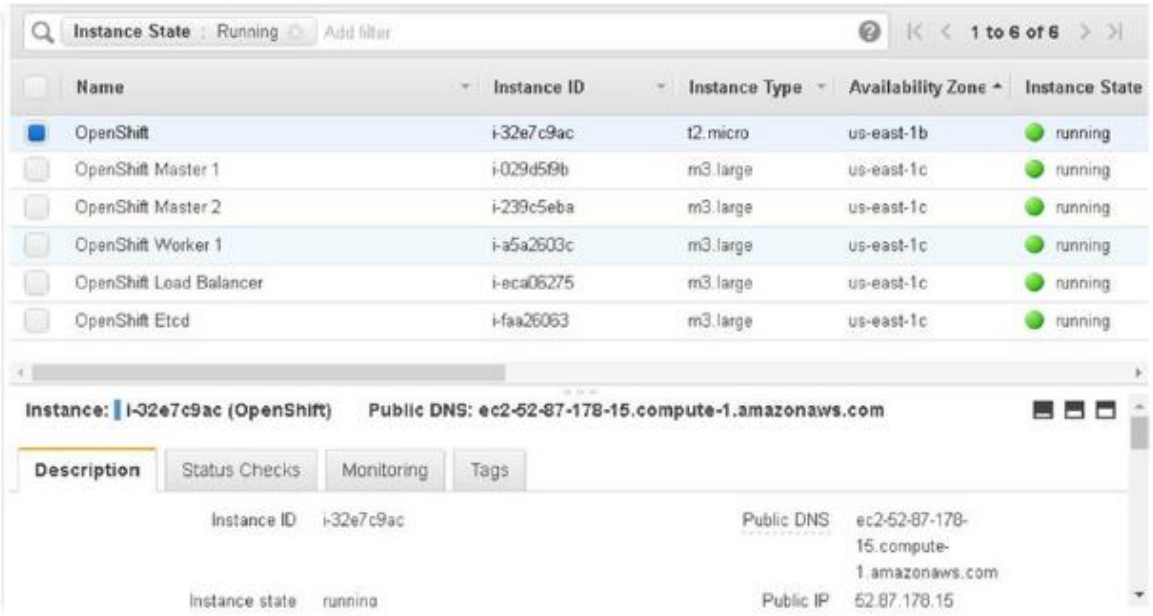
Instans CentOS 7 dapat diluncurkan dari :
https://aws.amazon.com/marketplace/pp/B00O7WM7QW.

Pilih Wilayah dan klik Lanjutkan. Dalam dialog Peluncuran pada EC2:CentOS 7 (x86_64) - dengan Pembaruan HVM yang muncul, pilih Jenis Instans EC2 m3.large atau yang lebih besar. Untuk VPC, pilih EC2 Klasik. Untuk Pasangan Kunci, pilih pasangan kunci yang sudah ada sebelumnya (docker.pem dalam contoh). Klik tombol Luncurkan Dengan 1-Klik. Instans CentOS ditunjukkan pada Gambar 15.2.



Gambar 15.2 Meluncurkan Instans Centoos

Luncurkan satu instans EC2 berdasarkan AMI Ubuntu. Instans yang diperlukan untuk kluster OpenShift ditunjukkan pada Gambar 15.3. Instans master, worker, dan Etcd tambahan dapat ditambahkan, tetapi kami telah menggunakan jumlah instans minimum untuk menentukan kluster master HA.



Gambar 15.3 Instans Centoos Untuk Kluster Openshift

Perangkat lunak berikut perlu diinstal:

- Docker pada setiap instans CoreOS
- Etcd pada instans etcd
- HAProxy pada instans LoadBalancer
- Network Manager pada setiap instans CentOS

Semua perangkat lunak sebelumnya kecuali Network Manager diinstal secara otomatis saat kita menjalankan playbook Ansible. Kita juga perlu menginstal kredensial docker.pem pada setiap instans CoreOS dan instans Ubuntu untuk OpenShift Ansible, yang akan kita instal

berikutnya.

15.2 MENGINSTAL KREDENSIAL

Dari mesin lokal, SCP salin docker.pem ke instans Ubuntu yang merupakan instans klien untuk meluncurkan kluster OpenShift menggunakan alamat IP Publik atau DNS Publik, yang dapat diperoleh dari Konsol EC2:

```
scp -i docker.pem docker.pem ubuntu@ec2-52-87-178-15.compute-1.amazonaws.com:~
```

Demikian pula, dapatkan DNS Publik untuk setiap instans CentOS, yang untuk master, worker, Etcd, dan LoadBalancer. SCP salin file docker.pem ke setiap instans CentOS. Perintah scp berikut menyalin berkas docker.pem ke instans master:

```
scp -i docker.pem docker.pem centos@ec2-54-90-107-98.compute-1.amazonaws.com:~ scp -i docker.pem docker.pem centos@ec2-54-221-182-68.compute-1.amazonaws.com:~
```

Perintah scp berikut menyalin docker.pem ke instans Worker.

```
scp -i docker.pem docker.pem centos@ec2-54-159-26-13.compute-1.amazonaws.com:~
```

Perintah scp berikut menyalin docker.pem ke instans LoadBalancer.

```
scp -i docker.pem docker.pem centos@ec2-54-226-7-241.compute-1.amazonaws.com:~
```

Perintah scp berikut menyalin docker.pem ke instans Etcd:

```
scp -i docker.pem docker.pem centos@ec2-54-160-210-253.compute-1.amazonaws.com:~
```

Perintah scp tidak menghasilkan output apa pun, seperti yang ditunjukkan pada Gambar 15.4.

```
[root@localhost ~]# scp -i docker.pem docker.pem centos@ec2-54-221-182-68.comput
e-1.amazonaws.com:~
docker.pem                                    100% 1696     1.7KB/s   00:00
[root@localhost ~]# scp -i docker.pem docker.pem centos@ec2-54-159-26-13.compute
-1.amazonaws.com:~
docker.pem                                    100% 1696     1.7KB/s   00:00
[root@localhost ~]# scp -i docker.pem docker.pem centos@ec2-54-226-7-241.compute
-1.amazonaws.com:~
docker.pem                                    100% 1696     1.7KB/s   00:00
[root@localhost ~]# scp -i docker.pem docker.pem centos@ec2-54-160-210-253.compu
te-1.amazonaws.com:~
docker.pem                                    100% 1696     1.7KB/s   00:00
```

Gambar 15.4 Menyalin Docker.Pem Ke Setiap Instans Coreos

Menginstal Network Manager

Untuk konektivitas jaringan, kluster OpenShift menggunakan Network Manager, yang perlu diinstal pada setiap instans CentOS. Login SSH ke setiap instans CentOS:

```
ssh -i docker.pem centos@ec2-54-90-107-98.compute-
1.amazonaws.com
ssh -i docker.pem centos@ec2-54-221-182-68.compute-
1.amazonaws.com
ssh -i docker.pem centos@ec2-54-159-26-13.compute-
1.amazonaws.com
ssh -i docker.pem centos@ec2-54-226-7-241.compute-
1.amazonaws.com
ssh -i docker.pem centos@ec2-54-160-210-253.compute-
1.amazonaws.com
```

Jalankan perintah berikut pada setiap instans CentOS untuk menginstal, memulai, dan mengaktifkan Network Manager serta menemukan status:

sudo yum install NetworkManager sudo systemctl start NetworkManager
sudo systemctl enable NetworkManager sudo systemctl status NetworkManager

Memasang OpenShift melalui Ansible pada Mesin Klien

Kita akan menggunakan platform otomatisasi perangkat lunak Ansible untuk memasang perangkat lunak OpenShift dari jarak jauh dari instans Ubuntu. Kita tidak perlu masuk ke setiap instans kluster OpenShift untuk meluncurkan perangkat lunak apa pun selain Network Manager, yang telah kita pasang. Masuk ke instans Ubuntu melalui SSH:

```
ssh -i "docker.pem" ubuntu@ec2-52-87-178-15.compute-
1.amazonaws.com
```

Versi Ubuntu untuk Ansible tersedia di Arsip Paket Pribadi Ubuntu (PPA). Untuk mengonfigurasi PPA dan menginstal Ansible, pertama-tama jalankan perintah berikut:

```
sudo apt-get install software-properties-common sudo apt-add-
repository ppa:ansible/ansible
```

PPA Ansible ditambahkan ke repositori seperti yang ditunjukkan pada Gambar 15.5.

```
ubuntu@ip-10-0-0-120:~$  sudo apt-get install software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
software-properties-common is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ip-10-0-0-120:~$  sudo apt-add-repository ppa:ansible/ansible
 Ansible is a radically simple IT automation platform that makes your applicatio
ns and systems easier to deploy. Avoid writing scripts or custom code to deploy
and update your applications— automate in a language that approaches plain Engli
sh, using SSH, with no agents to install on remote systems.

http://ansible.com/
 More info: https://launchpad.net/~ansible/+archive/ubuntu/ansible
Press [ENTER] to continue or ctrl-c to cancel adding it

gpg: keyring `/tmp/tmpiz0jc41y/secring.gpg' created
gpg: keyring `/tmp/tmpiz0jc41y/pubring.gpg' created
gpg: requesting key 7BB9C367 from hkp server keyserver.ubuntu.com
gpg: /tmp/tmpiz0jc41y/trustdb.gpg: trustdb created
gpg: key 7BB9C367: public key "Launchpad PPA for Ansible, Inc." imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
OK
ubuntu@ip-10-0-0-120:~$
```

Gambar 15.5 Menginstal Ansible PPA

Perbarui repositori dan instal Ansible:

```
sudo apt-get update
sudo apt-get install ansible
```

Ansible diinstal pada instans Ubuntu.

Unduh repositori git openshift-ansible. CD (ubah direktori) ke direktori openshift-ansible:

```
git clone https://github.com/openshift/openshift-ansible.git cd
openshift-ansible
```

Untuk mencantumkan pengaturan default untuk alamat IP dan nama host, jalankan perintah berikut:

```
ansible-playbook playbooks/byo/openshift_facts.yml
```

Keluaran perintah ditunjukkan pada Gambar 15.6.

```
ubuntu@ip-10-0-0-120:~/openshift-ansible$ ansible-playbook playbooks/byo/openshi
ft_facts.yml
 [WARNING]: provided hosts list is empty, only localhost is available


PLAY [localhost] ***************************************************************

TASK [Verify Ansible version is greater than or equal to 2.1.0.0] **************
skipping: [localhost]

PLAY [localhost] ***************************************************************

TASK [include_vars] ************************************************************
ok: [localhost]

TASK [add_host] ****************************************************************
[DEPRECATION WARNING]: Using bare variables is deprecated. Update your playbooks
 so that the environment value uses the full variable syntax
('{{g_all_hosts}}').
This feature will be removed in a future release.
Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.

PLAY [l_oo_all_hosts] **********************************************************
skipping: no hosts matched
```

Gambar 15.6 Mencantumkan Pengaturan Default Untuk Alamat IP Dan Nama Host

Pengaturan alamat IP/nama host default ditampilkan seperti yang ditunjukkan pada Gambar 15.7.

```
TASK [Evaluate oo_etcd_to_config] **********************************************

TASK [Evaluate oo_masters_to_config] *******************************************

TASK [Evaluate oo_nodes_to_config] *********************************************

TASK [Evaluate oo_nodes_to_config] *********************************************

TASK [Evaluate oo_first_etcd] **************************************************
skipping: [localhost]

TASK [Evaluate oo_first_master] ************************************************
skipping: [localhost]

TASK [Evaluate oo_lb_to_config] ************************************************

TASK [Evaluate oo_nfs_to_config] ***********************************************

PLAY [Gather Cluster facts] ****************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0

ubuntu@ip-10-0-0-120:~/openshift-ansible$
```

Gambar 15.7 Pengaturan Alamat IP/Nama Host Default

15.3 MENGONFIGURASI ANSIBLE

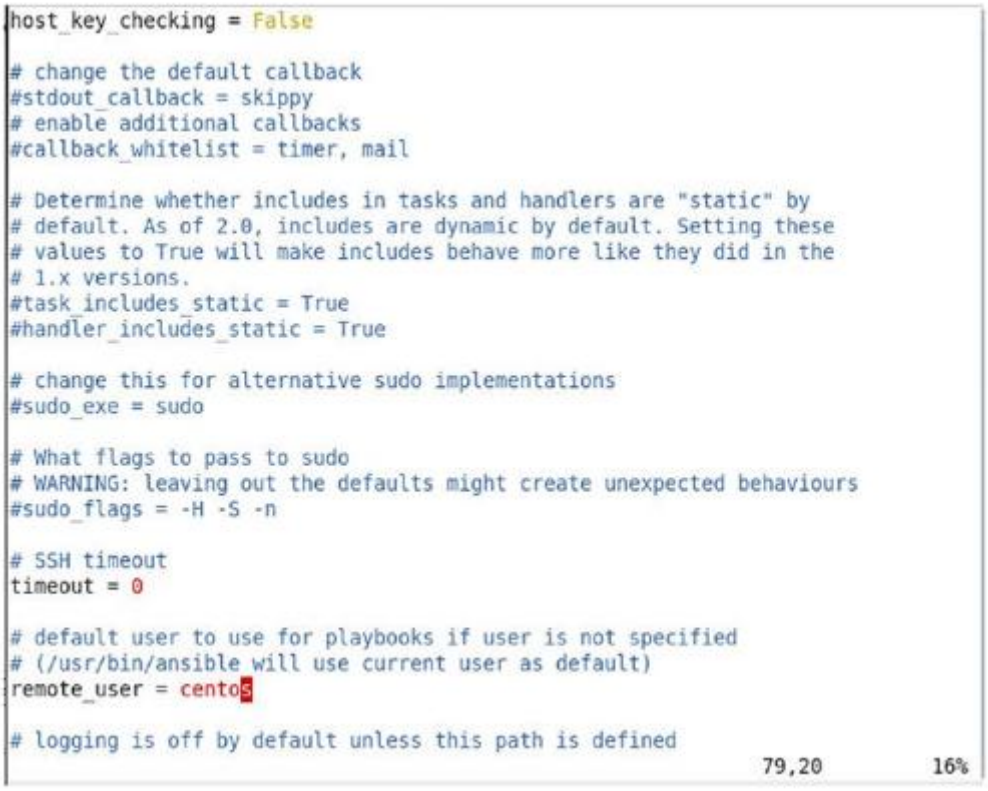
Beberapa pengaturan konfigurasi Ansible dapat dimodifikasi dalam berkas konfigurasi `/etc/ansible/ansible.cfg`. Kita perlu memodifikasi beberapa pengaturan ini untuk OpenShift Ansible. Buka berkas `/etc/ansible/ansible.cfg dalam editor vi.`

```
sudo vi /etc/ansible/ansible.cfg
```

Tambahkan/modifikasi pengaturan berikut dalam header [defaults].

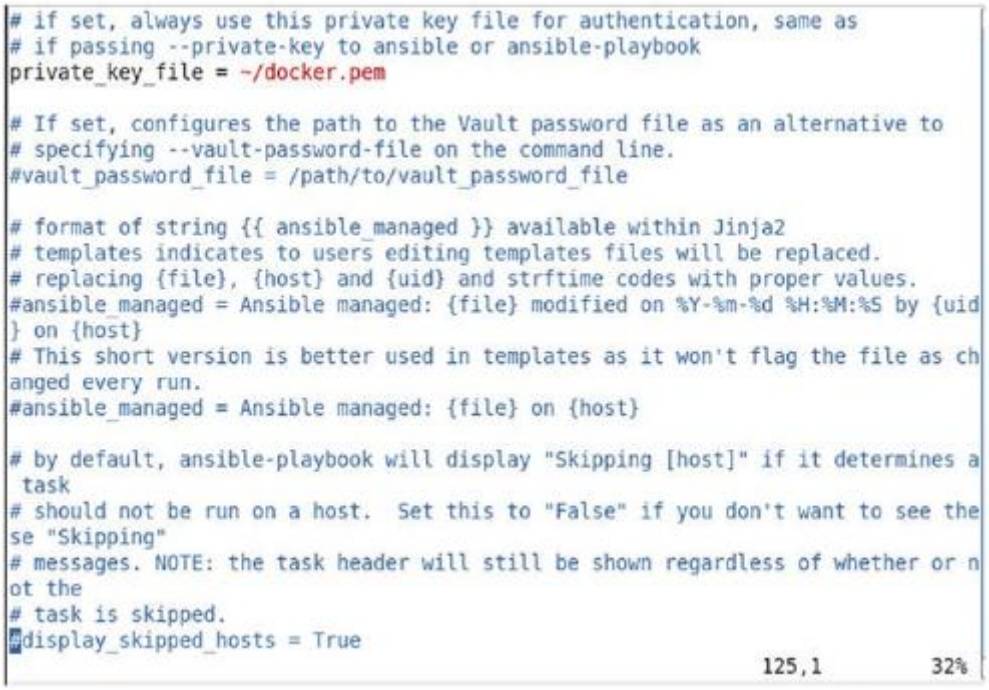
```
sudo=yes
ask_sudo_pass=False
ask_pass=False
remote_user = centos
host_key_checking = False timeout=0
private_key_file=~/.docker.pem
```

Beberapa pengaturan ditampilkan di ansible.cfg pada Gambar 15.8.



Gambar 15.8 Mengonfigurasi Ansible.Cfg

Properti ini tersebar di seluruh berkas dan tidak ditempatkan bersama, seperti yang ditunjukkan pada Gambar 15.9.



Gambar 15.9 Properti Konfigurasi Ansible.Cfg Tidak Ditempatkan Bersama

Berkas inventaris default yang digunakan oleh Ansible adalah `/etc/ansible/hosts`, yang digunakan untuk mengonfigurasi host untuk node master OpenShift, node pekerja, node etcd, dan node LoadBalancer. Buka berkas `/etc/ansible/hosts` di editor vi.

```
sudo vi /etc/ansible/hosts
```

Di bagian atas berkas, konfigurasikan yang berikut ini:

```
[0Sev3: children]
masters
etcd
lb
nodes
```

Berikutnya, tentukan beberapa variabel:

```
[0Sev3: children]
Ansible_user=centos
Ansible_sudo=true
Deployment_type=origin
Ansible_ssh_private_key_file=~/docker.pem
```

Bagian atas berkas /etc/ansible/hosts ditunjukkan pada Gambar 15.10.

Gambar 15.10 Berkas /Etc/Ansible/Hosts

Beberapa variabel klaster lainnya didukung (lihat Tabel 2. Variabel Klaster di https://docs.openshift.org/latest/install_config/install/advanced_install.html#multiple-masters), tetapi kami hanya menggunakan yang minimum yang diperlukan. Dengan beberapa master, metode HA (High Availability) asli didukung, yang memanfaatkan LoadBalancer yang dikonfigurasi dengan host [lb] dalam berkas hosts atau yang telah dikonfigurasi sebelumnya.

```
openshift_master_cluster_method=native
```

Kami akan menentukan host untuk penyeimbang beban dalam berkas /etc/ansible/hosts. Dapatkan nama host atau DNS Publik atau IP Publik untuk instans penyeimbang beban dari Konsol EC2 dan tentukan hal yang sama dalam pengaturan berikut di berkas /etc/ansible/hosts:

```
openshift_master_cluster_hostname=ec2-54-226-7-241.compute-
1.amazonaws.com openshift_master_cluster_public_hostname=ec2-
54-226-7-241.compute-1.amazonaws.com
```

Selanjutnya, tentukan master dalam berkas /etc/ansible/hosts.

Beberapa variabel host (lihat Tabel 1. Variabel Host di URL yang ditunjukkan di atas) didukung, tetapi kami hanya menggunakan variabel host yang ditunjukkan dalam Tabel 15-1 untuk master, worker, etcd, dan lb.

Tabel 15.1. Variabel Host

Variabel Host	Deskripsi	ContohNilai
openshift_ip	IP Pribadi yang dapat diperoleh dari Konsol EC2	10.156.14.183
openshift_public_ip	IP Publik yang dapat diperoleh dari Konsol EC2	54.90.107.98
openshift_hostname	Nama host untuk host yang dapat diperoleh dari DNS Pribadi di Konsol EC2	ip-10-156-14-183.ec2.internal
openshift_public_hostname	Nama host publik untuk host yang dapat diperoleh dari DNS Publik di Konsol EC2	ec2-54-90-107-98.compute-1. amazonaws.com

Konfigurasikan bagian [etcd], [lb], dan [nodes] dengan cara yang sama. Master juga tercantum dalam [nodes] tetapi dibuat tidak dapat dijadwalkan dengan openshift_schedulable yang disetel ke false dan labelnya ditambahkan dengan openshift_node_labels. Pengaturan host harus serupa dengan berikut ini; nama host dan alamat IP akan berbeda untuk setiap pengguna.

```
[masters]
ec2-54-90-107-98.compute-1.amazonaws.com
openshift_ip=10.156.14.183 openshift_public_ip=54.90.107.98
openshift_hostname=ip-10-156-14-183.ec2.internal
openshift_public_hostname=ec2-54-90-107-98.compute-
1.amazonaws.com
ec2-54-221-182-68.compute-1.amazonaws.com
openshift_ip=10.154.46.153 openshift_public_ip=54.221.182.68
openshift_hostname=ip-10-154-46-153.ec2.internal
openshift_public_hostname=ec2-54-221-182-68.compute-
1.amazonaws.com

[etcd]
ec2-54-160-210-253.compute-1.amazonaws.com
openshift_ip=10.153.195.121 openshift_public_ip=54.160.210.253
openshift_hostname=ip-10-153-195-121.ec2.internal
openshift_public_hostname=ec2-54-160-210-253.compute-
1.amazonaws.com

[lb]
ec2-54-226-7-241.compute-1.amazonaws.com
openshift_ip=10.154.38.224 openshift_public_ip=54.226.7.241
```

```
openshift_hostname=ip-10-154-38-224.ec2.internal
openshift_public_hostname=ec2-54-226-7-241.compute-
1.amazonaws.com

[nodes]
ec2-54-90-107-98.compute-1.amazonaws.com
openshift_ip=10.156.14.183 openshift_public_ip=54.90.107.98
openshift_hostname=ip-10-156-14-183.ec2.internal
openshift_public_hostname=ec2-54-90-107-98.compute-
1.amazonaws.com openshift_node_labels="{'region': 'primary',
'zone': 'east'}" openshift_schedulable=false
ec2-54-221-182-68.compute-1.amazonaws.com
openshift_ip=10.154.46.153 openshift_public_ip=54.221.182.68
openshift_hostname=ip-10-154-46-153.ec2.internal
openshift_public_hostname=ec2-54-221-182-68.c
```

15.4 MENJALANKAN ANSIBLE PLAYBOOK

File inventaris default adalah /etc/ansible/hosts tetapi file lain dapat dikonfigurasi dengan pengaturan inventaris di ansible.cfg, misalnya:

```
inventory = /etc/ansible/inventory/hosts
```

Kami telah mengonfigurasi file inventaris default /etc/ansible/hosts. Jalankan klaster OpenShift dengan menjalankan Ansible playbook:

```
ansible-playbook ~/openshift-ansible/playbooks/byo/config.yml
```

Perangkat lunak OpenShift seperti Docker, HAProxy, dan sebagainya diinstal dan dijalankan pada host yang dikonfigurasi, seperti yang ditunjukkan pada Gambar 15.11.

```
TASK [openshift_hosted : set_fact] *********************************************
skipping: [ec2-54-90-107-98.compute-1.amazonaws.com]

TASK [openshift_hosted : Determine if volume is already attached to dc/docker-re
gistry] ***
skipping: [ec2-54-90-107-98.compute-1.amazonaws.com]

TASK [openshift_hosted : set_fact] *********************************************
skipping: [ec2-54-90-107-98.compute-1.amazonaws.com]

TASK [openshift_hosted : Add volume to dc/docker-registry] *********************
skipping: [ec2-54-90-107-98.compute-1.amazonaws.com]

TASK [openshift_hosted : Delete temp directory] ********************************
ok: [ec2-54-90-107-98.compute-1.amazonaws.com]

PLAY RECAP *********************************************************************
ec2-54-159-26-13.compute-1.amazonaws.com : ok=143  changed=42   unreachable=0
 failed=0
ec2-54-160-210-253.compute-1.amazonaws.com : ok=97   changed=34   unreachable=0
   failed=0
ec2-54-221-182-68.compute-1.amazonaws.com : ok=274  changed=91   unreachable=0
  failed=0
ec2-54-226-7-241.compute-1.amazonaws.com : ok=71   changed=17   unreachable=0
 failed=0
ec2-54-90-107-98.compute-1.amazonaws.com : ok=411  changed=106  unreachable=0
 failed=0
localhost                  : ok=15   changed=9    unreachable=0    failed=0

ubuntu@ip-10-0-0-128:~/openshift-ansible$
```

Gambar 15.11 Menjalankan Ansible Playbook

Login SSH ke salah satu instans utama dan daftarkan node dalam klaster OpenShift:

```
oc get nodes
```

Tiga node, dua di antaranya tidak dapat dijadwalkan, dicantumkan seperti yang ditunjukkan pada Gambar 15.12.

```
[root@localhost ~]# ssh -i docker.pem centos@ec2-54-90-107-98.compute-1.amazonaw
s.com
Last login: Tue Aug  9 01:17:23 2016 from ec2-52-87-178-15.compute-1.amazonaws.c
om
[centos@ip-10-156-14-183 ~]$ sudo oc get nodes
NAME                           STATUS                     AGE
ip-10-113-176-99.ec2.internal  Ready                      33m
ip-10-154-46-153.ec2.internal  Ready,SchedulingDisabled   33m
ip-10-156-14-183.ec2.internal  Ready,SchedulingDisabled   33m
[centos@ip-10-156-14-183 ~]$
```

Gambar 15.12 Node Dalam Klaster Openshift

Menguji Klaster

Untuk menguji klaster OpenShift, masuklah ke klaster.

```
oc login
```

Tentukan Nama Pengguna sebagai sistem dan Kata Sandi sebagai admin. Klaster OpenShift

telah masuk. Awalnya tidak ada proyek yang dibuat, seperti yang ditunjukkan pada Gambar 15.13.

Gambar 15.13 Masuk Ke Klaster Openshift

Buat proyek baru, misalnya hello-openshift dengan perintah oc new-project:

```
oc new-project hello-openshift
```

Proyek hello-openshift dibuat seperti yang ditunjukkan pada Gambar 15.14.

Gambar 15.14 Membuat Proyek Hello-Openshift

Temukan status proyek:

```
oc status
```

Buat aplikasi OpenShift baru dengan perintah oc new-app.

```
oc new-app openshift/ruby-20-
centos7~https://github.com/openshift/ruby-hello-world.git
```

Aplikasi OpenShift baru dibuat.

Untuk menghapus semua objek untuk aplikasi, jalankan perintah berikut:

```
oc delete all -l app=appName
```

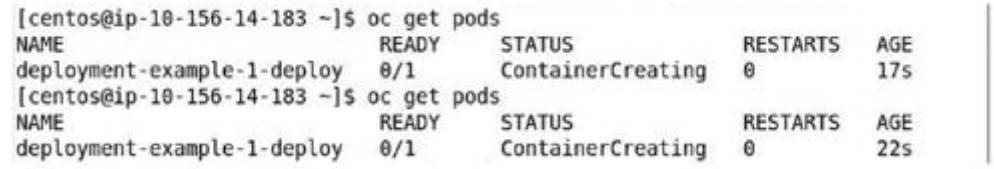
Misalnya, untuk menghapus semua objek untuk aplikasi hello-world, jalankan perintah berikut:

```
oc delete all -l app=hello-world
```

Buat aplikasi lain dengan perintah oc new-app. Tag gambar dapat ditentukan, misalnya untuk gambar Docker openshift/deployment-example.

```
oc new-app openshift/deployment-example:v1
```

Aplikasi OpenShift dibuat. Awalnya perintah oc get pods dapat mencantumkan pod sebagai tidak berjalan, tetapi dengan Status ContainerCreating seperti yang ditunjukkan pada Gambar 15.15.



Gambar 15.15 Mencantumkan Pod

Beberapa aplikasi dari citra Docker yang sama dapat dimulai secara bersamaan; misalnya, jalankan perintah yang sama lagi:

```
oc new-app openshift/deployment-example:v1
```

Ketika kedua aplikasi telah dimulai, dua pod akan dicantumkan, seperti yang ditunjukkan pada Gambar 15.16.

Gambar 15.16 Menjalankan Beberapa Aplikasi Secara Bersamaan

Node tempat pod berjalan dapat dicantumkan dengan perintah berikut:

```
oc get -o wide pods
```

Node untuk pod juga dicantumkan seperti yang ditunjukkan pada Gambar 15.17.

```
[centos@ip-10-156-14-183 ~]$ oc get -o wide pods
NAME                        READY     STATUS    RESTARTS   AGE       NODE
deployment-example-1-awoib  1/1       Running   0          1m        ip-10-113-
176-99.ec2.internal
[centos@ip-10-156-14-183 ~]$
```

Gambar 15.17 Mencantumkan Pod Termasuk Node

Perintah oc describe digunakan untuk mendeskripsikan deployment:

```
oc describe dc/deployment-example
```

Layanan dicantumkan dengan perintah berikut:

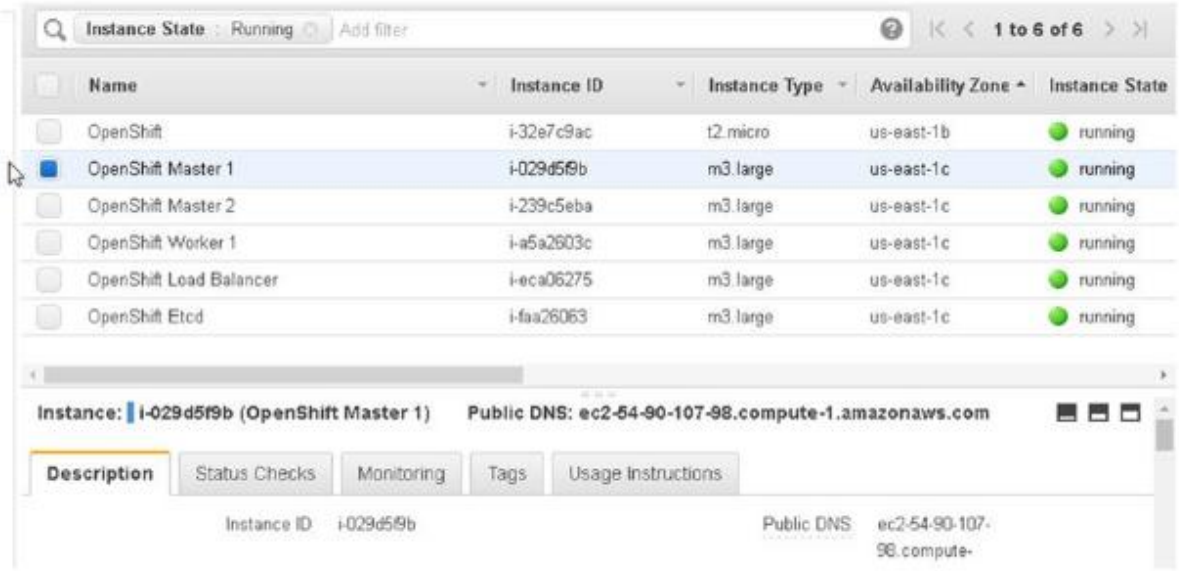
```
oc get services
```

Daftar semua objek OpenShift dengan perintah berikut:

```
oc get all
```

Menguji High Availability

Dengan beberapa master dan High availability yang dikonfigurasi dengan metode native, load balancer mendistribusikan beban master di antara master. Server API master diekspos pada Alamat IP load balancer, tetapi sebenarnya satu server API berjalan pada masing-masing master. Dua instans master dan satu instans pekerja ditunjukkan pada Gambar 15.18.

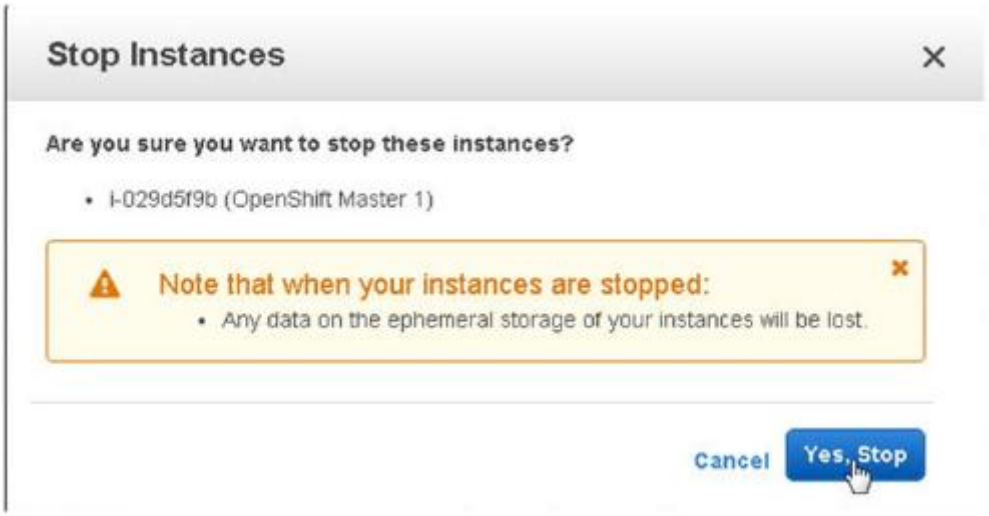


Gambar 15.18 Instans EC2 Yang Menjalankan Master Dan Pekerja Openshift

Untuk menunjukkan ketersediaan klaster yang tinggi, matikan salah satu master. Pilih instans master di Konsol EC2 dan di Tindakan pilih Status Instans ➤ Berhenti seperti yang ditunjukkan pada Gambar 15.19.

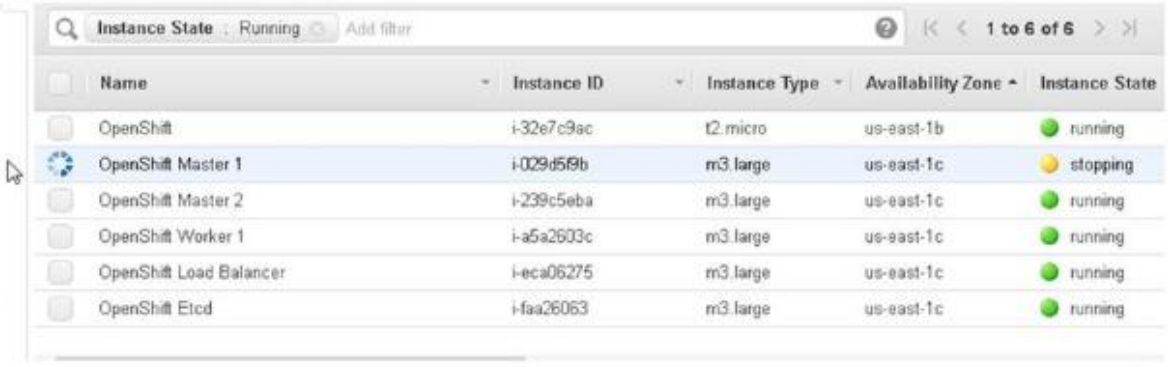
Gambar 15.19 Menghentikan Master Openshift

Dalam dialog Hentikan Instans, klik Ya, Berhenti seperti yang ditunjukkan pada Gambar 15.20.



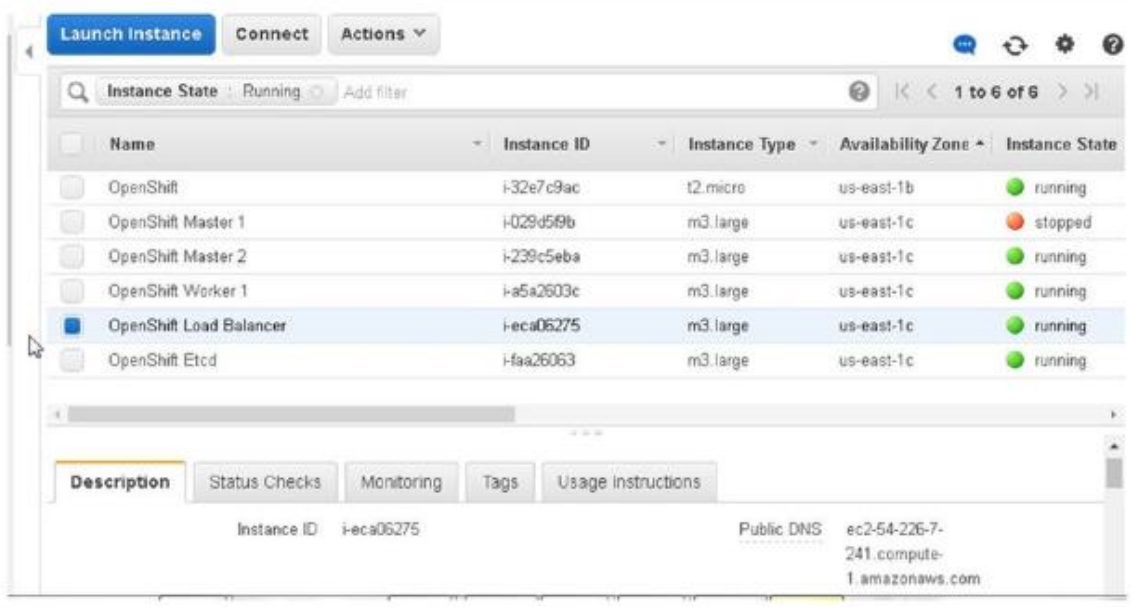
Gambar 15.20 Kotak Dialog Stop Instances

Salah satu master mulai mati, seperti yang ditunjukkan pada Gambar 15.21.



Gambar 15.21 Salah Satu Master Openshift Berhenti

Setelah master mati, penyeimbang beban dan master lainnya seharusnya masih berjalan seperti yang ditunjukkan pada Gambar 15.22.



Gambar 15.22 Openshift Load Balancer Dan Master Lainnya Masih Berjalan

Jalankan perintah berikut untuk mencantumkan konfigurasi kubeconfig klaster.

```
kubectl config view
```

Server API klaster tercantum sebagai DNS Publik dari load balancer seperti yang ditunjukkan pada Gambar 15.23.

```
[centos@ip-10-154-46-153 ~]$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: REDACTED
    server: https://ec2-54-226-7-241.compute-1.amazonaws.com:8443
  name: ec2-54-226-7-241-compute-1-amazonaws-com:8443
contexts:
- context:
    cluster: ec2-54-226-7-241-compute-1-amazonaws-com:8443
    namespace: default
    user: system:admin/ec2-54-226-7-241-compute-1-amazonaws-com:8443
  name: default/ec2-54-226-7-241-compute-1-amazonaws-com:8443/system:admin
current-context: default/ec2-54-226-7-241-compute-1-amazonaws-com:8443/system:ad
min
kind: Config
preferences: {}
users:
- name: system:admin/ec2-54-226-7-241-compute-1-amazonaws-com:8443
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
[centos@ip-10-154-46-153 ~]$
```

Gambar 15-23. Mencantumkan Konfigurasi Kubeconfig Klister

Atau, jalankan perintah berikut untuk mencantumkan info kluster.

```
kubectl cluster-info
```

URL master Kubernetes yang tercantum dibuat dari DNS Publik penyeimbang beban seperti yang ditunjukkan pada Gambar 15.24.

```
[centos@ip-10-154-46-153 ~]$ kubectl cluster-info
Kubernetes master is running at https://ec2-54-226-7-241.compute-1.amazonaws.com
:8443
[centos@ip-10-154-46-153 ~]$
```

Gambar 15-24. Mencantumkan Info Kluster

Log masuk SSH ke instans master lainnya dan cantumkan node dengan oc get nodes. Salah satu node master tercantum sebagai NotReady, sementara node master lainnya adalah Ready, seperti yang ditunjukkan pada Gambar 15.25. Jika master yang dihentikan dimulai ulang, node tersebut akan tercantum lagi sebagai Ready.

```
[centos@ip-10-154-46-153 ~]$ oc get nodes
NAME                          STATUS                        AGE
ip-10-113-176-99.ec2.internal   Ready                         1h
ip-10-154-46-153.ec2.internal   Ready,SchedulingDisabled      1h
ip-10-156-14-183.ec2.internal   NotReady,SchedulingDisabled   1h
[centos@ip-10-154-46-153 ~]$ oc get nodes
NAME                          STATUS                     AGE
ip-10-113-176-99.ec2.internal   Ready                      1h
ip-10-154-46-153.ec2.internal   Ready,SchedulingDisabled   1h
ip-10-156-14-183.ec2.internal   Ready,SchedulingDisabled   1h
[centos@ip-10-154-46-153 ~]$
```

Gambar 15.25 Mencantumkan Node, Yang Dapat Dijadwalkan Dan Tidak Dapat Dijadwalkan

Ringkasan

Dalam bab ini, kami memperkenalkan platform lain, yang disebut OpenShift, yang merupakan platform PaaS dengan Kubernetes tertanam. Satu master adalah satu titik kegagalan (SPOF). Kami membahas pembuatan master ketersediaan tinggi dengan OpenShift. Dalam bab berikutnya, kami akan membahas pembuatan situs web ketersediaan tinggi.

BAB 16
MENGEMBANGKAN SITUS WEB DENGAN KETERSEDIAAN TINGGI

16.1 KETERSEDIAAN TINGGI DI KUBERNETES DENGAN ROUTE 53

Pada Bab 4, kami menggunakan beberapa zona ketersediaan AWS untuk menyediakan toleransi kesalahan atas kegagalan suatu zona. Namun, master ketersediaan tinggi tidak digunakan, dan satu master merupakan satu titik kegagalan. Pada Bab 15, kami menggunakan master ketersediaan tinggi dengan OpenShift dan Ansible, tetapi satu elastic load balancer tetap menjadi satu titik kegagalan.

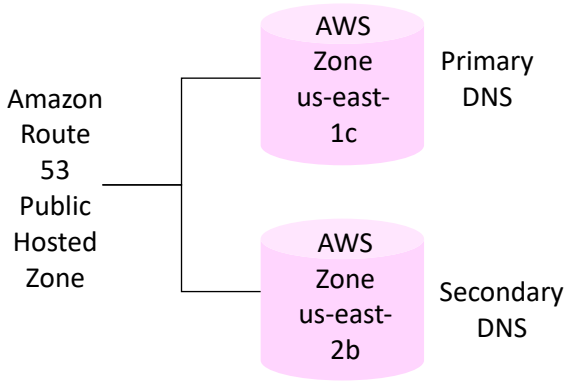
Masalah

Untuk situs web dengan ketersediaan tinggi, beberapa DNS publik perlu dikonfigurasi. Masalah lainnya adalah Amazon Elastic Load Balancer tidak mendukung Arsitektur Ketersediaan Tinggi Antar-Wilayah AWS, yang mana beberapa pengontrol master dalam master HA dapat ditempatkan di Wilayah AWS yang berbeda.

Amazon Elastic Load Balancer hanya mendukung Arsitektur Ketersediaan Tinggi Antar-Zona AWS dalam wilayah yang sama. Sementara zona AWS berada di lokasi fisik yang berbeda dan terisolasi satu sama lain (kegagalan satu zona tidak menyebabkan kegagalan di zona lain), HA tidak tersebar di wilayah geografis yang lebih luas.

Solusi

Amazon Route 53 menyediakan failover DNS, yang dapat digunakan untuk mengembangkan situs web dengan ketersediaan tinggi. Route 53 menyediakan failover DNS di seluruh wilayah AWS seperti yang ditunjukkan pada Gambar 16.1. Failover DNS Route 53 dapat digunakan untuk menjalankan aplikasi di seluruh zona atau wilayah AWS dan mengonfigurasi penyeimbang beban elastis alternatif untuk menyediakan failover di seluruh zona atau wilayah. Failover DNS Route 53 bukanlah pola desain Kubernetes, tetapi memanfaatkan pola arsitektur Primer-Sekunder Amazon Route 53.



Gambar 16.1 Failover DNS Amazon Route 53

Ikhtisar

Amazon Route 53 adalah layanan nama domain (DNS) cloud yang sangat tersedia dan dapat diskalakan yang menghubungkan permintaan pengguna ke infrastruktur yang berjalan di AWS, seperti instans Amazon EC2, penyeimbang beban, dan bucket Amazon S3. Klaster Kubernetes dapat digunakan menggunakan AWS CloudFormation, seperti yang dibahas dalam Bab 4. Namun, klaster yang dikembangkan di sana, menggunakan alat kube-aws CLI, adalah klaster master tunggal tanpa penyediaan failover.

Klaster yang sangat tersedia memiliki toleransi terhadap kegagalan node dalam klaster dengan failover bawaan ke node lain dalam klaster. Dalam bab ini, kami akan mengembangkan klaster Kubernetes yang sangat tersedia menggunakan AWS CloudFormation pada CoreOS. Kami akan menyediakan beberapa (tiga) AWS CloudFormation dan kemudian menghosting Layanan Kubernetes aplikasi contoh (hello-world) pada setiap CloudFormation. Kami akan menggunakan zona yang dihosting publik untuk domain contoh guna merutekan lalu lintas ke domain tersebut. Bab ini membahas topik-topik berikut.

- Menetapkan lingkungan
- Membuat CloudFormations
- Mengonfigurasi DNS eksternal
- Membuat layanan Kubernetes
- Membuat AWS Route 53
- Membuat zona hosting
- Mengonfigurasi name server
- Membuat set rekaman
- Menguji ketersediaan tinggi

Mengatur Lingkungan

Prosedur berikut digunakan untuk membuat aplikasi web dengan ketersediaan tinggi.

1. Buat tiga AWS CloudFormations di CoreOS dengan satu pengontrol Kubernetes di masing-masing. CloudFormations dapat berada di wilayah yang sama atau beberapa wilayah; kami telah menggunakan wilayah yang sama dalam contoh, karena beberapa sumber daya AWS mungkin tidak tersedia di semua wilayah dan zona ketersediaan. Tambahkan rekaman A untuk setiap IP pengontrol ke Domain oramagsearch.com (URL yang digunakan dalam bab ini, tetapi nama domain akan berbeda untuk pengguna yang berbeda).
2. Masuk ke setiap instans pengontrol CoreOS. Buat layanan Kubernetes untuk aplikasi contoh (hello-world) yang diekspos pada penyeimbang beban elastis. Dengan satu Penyeimbang Beban Elastis yang diekspos pada setiap CloudFormation, tiga DNS publik tersedia.
3. Buat AWS Route 53 dengan dua DNS yang dikonfigurasi untuk failover.
4. Buat zona hosting publik AWS untuk domain contoh seperti domain oramagsearch.com (nama domain akan berbeda untuk pengguna yang berbeda).
5. Tambahkan server nama yang ditetapkan ke Public Hosted Zone ke pendaftar domain oramagsearch.com.
6. Buat dua set rekaman sumber daya alias yang menunjuk ke dua penyeimbang beban

elastis yang berbeda. Set rekaman dikonfigurasi untuk failover, dengan satu menjadi yang utama dan yang lainnya menjadi yang sekunder dalam konfigurasi Failover.

Buat satu instans EC2 dengan Amazon Linux AMI. Instans tersebut digunakan untuk meluncurkan tiga CloudFormations, dan masuk SSH ke masing-masing pengontrol untuk membuat layanan Kubernetes.

16.2 MEMBUAT CLOUDFORMATIONS

Masuk SSH ke instans Amazon Linux dari tiga shell Linux yang berbeda di mesin lokal.

```
ssh -i docker.pem ec2-user@ec2-54-242-131-243.compute-
1.amazonaws.com
```

Seperti yang dibahas dalam Bab 3, prosedur untuk membuat AWS CloudFormation adalah sebagai berikut:

1. Instal Kube-aws (harus diinstal hanya sekali untuk instans Amazon Linux)
2. Siapkan Parameter Klaster seperti membuat pasangan kunci EC2, kunci KMS, dan nama DNS Eksternal. Pasangan kunci EC2 yang sama (kubernetes-coreos) dan nama DNS Eksternal (oramagsearch.com) digunakan untuk setiap CloudFormation.
3. Buat Direktori Aset untuk CloudFormation Klaster (direktori yang berbeda untuk setiap CloudFormation).
4. Inisialisasi CloudFormation klaster.
5. Render konten Direktori Aset.
6. Kustomisasi klaster untuk membuat tiga node pekerja, bukan satu.
7. Validasi tumpukan CloudFormation.
8. Luncurkan klaster CloudFormation.

Perintah umum untuk membuat pasangan kunci EC2 adalah sebagai berikut:

```
aws ec2 create-key-pair --key-name kubernetes-coreos --query
'KeyMaterial' --output text > kubernetes-coreos.pem
chmod 400 kubernetes-coreos.pem
```

Perintah untuk membuat kunci KMS adalah sebagai berikut:

```
aws kms --region=us-east-1 create-key --description="kube-aws
assets"
```

Salin string KeyMetadata.Arn dan gunakan untuk menginisialisasi tumpukan CloudFormation. Misalnya, kluster yang disebut kubernetes-coreos-cluster-1 dengan direktori aset sebagai `kube-coreos-cluster-1` diinisialisasi sebagai berikut:

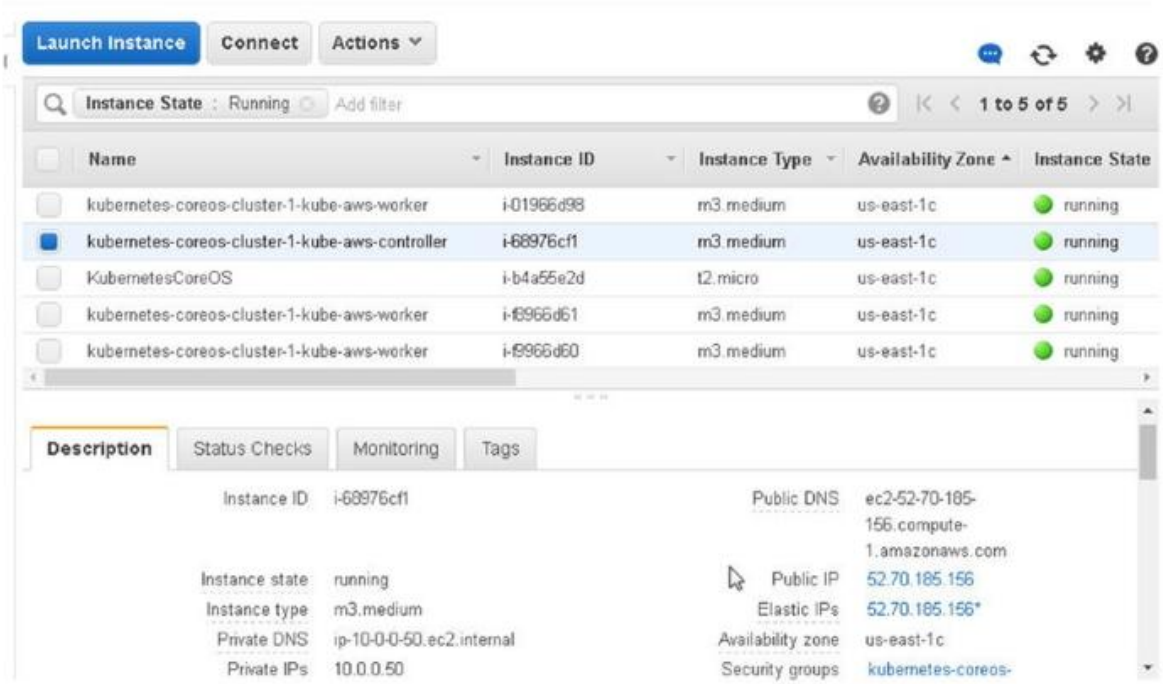
```
mkdir kube-coreos-cluster-1
```

```
cd kube-coreos-cluster-1

kube-aws init --cluster-name=kubernetes-coreos-cluster-1 --
external-dns-name=ORAMAGSEARCH.COM --region=us-east-1 --
availability-zone=us-east-1c --key-name=kubernetes-coreos --
kms-key-arn="arn:aws:kms:us-east-
1:xxxxxxxxxx:key/xxxxxxxxxxxxXXXXXXXXX"
```

Perintah untuk merender konten direktori aset, memvalidasi tumpukan CloudFormation, dan meluncurkan tumpukan CloudFormation adalah sebagai berikut:

```
kube-aws render
kube-aws vaidate
kube-aws up
```

Selanjutnya, luncurkan ke instans pengontrol untuk setiap kluster Kubernetes. IP Publik pengontrol dapat diperoleh dari Konsol EC2 seperti yang ditunjukkan pada Gambar 16.2.



Gambar 16.2 Cloudformation Untuk Kluster Kubernetes

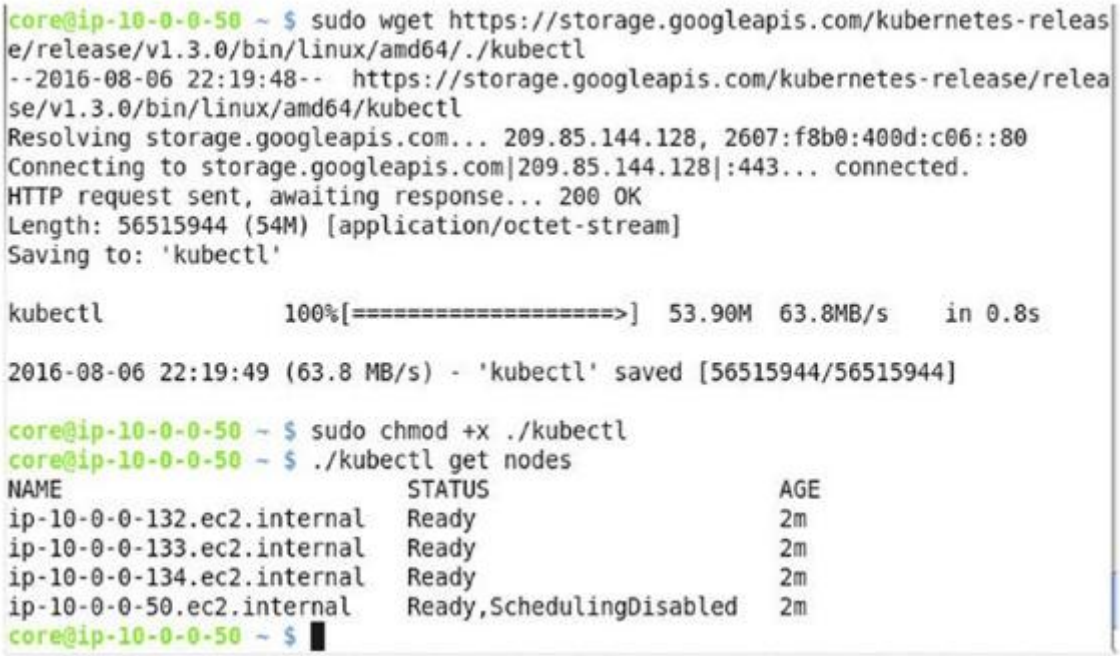
Log masuk SSH menggunakan pasangan kunci EC2:

```
ssh-i "kubernetes-coreos.pem" core@52.70.185.156
```

Prompt perintah CoreOS akan ditampilkan. Instal biner kubectl dan daftarkan node:

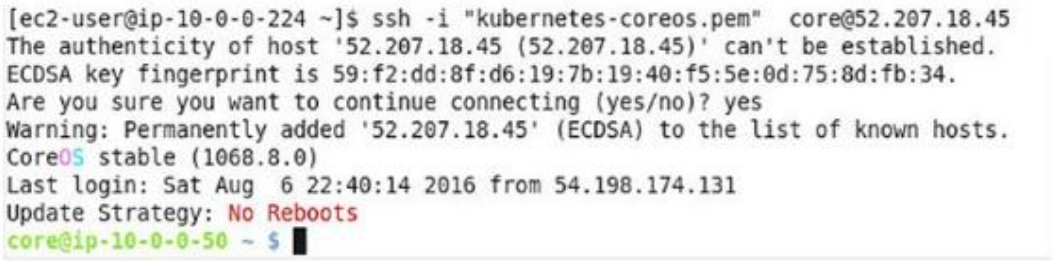
```
sudo wget https://storage.googleapis.com/kubernetes-
release/release/v1.23.0/bin/linux/amd64/kubectl
```

Node master tunggal dan tiga node pekerja dalam kluster harus dicantumkan, seperti yang ditunjukkan pada Gambar 16.3.



Gambar 16.3 Mencantumkan Node Dalam Kluster Kubernetes

Demikian pula, log masuk ke instans pengontrol kedua seperti yang ditunjukkan pada Gambar 16.4.



Gambar 16.4 Log Masuk SSH Ke Instans Pengontrol Kedua

Daftarkan node kluster seperti yang ditunjukkan pada Gambar 16.5.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-110.ec2.internal    Ready                      2m
ip-10-0-0-111.ec2.internal    Ready                      2m
ip-10-0-0-112.ec2.internal    Ready                      2m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   2m
core@ip-10-0-0-50 ~ $
```

Gambar 16.5 Node Untuk Kluster Kubernetes Kedua

Dan dengan cara yang sama, log masuk SSH ke instans pengontrol ketiga seperti yang ditunjukkan pada Gambar 16.6.

```
[ec2-user@ip-10-0-0-224 ~]$ ssh -i "kubernetes-coreos.pem"  core@52.204.178.21
The authenticity of host '52.204.178.21 (52.204.178.21)' can't be established.
ECDSA key fingerprint is 5e:69:e6:da:f0:d5:c7:4d:b2:1c:96:55:a9:f3:f5:b3.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.204.178.21' (ECDSA) to the list of known hosts.
CoreOS stable (1068.8.0)
Last login: Sat Aug  6 22:54:13 2016 from 54.198.174.131
Update Strategy: No Reboots
core@ip-10-0-0-50 ~ $
```

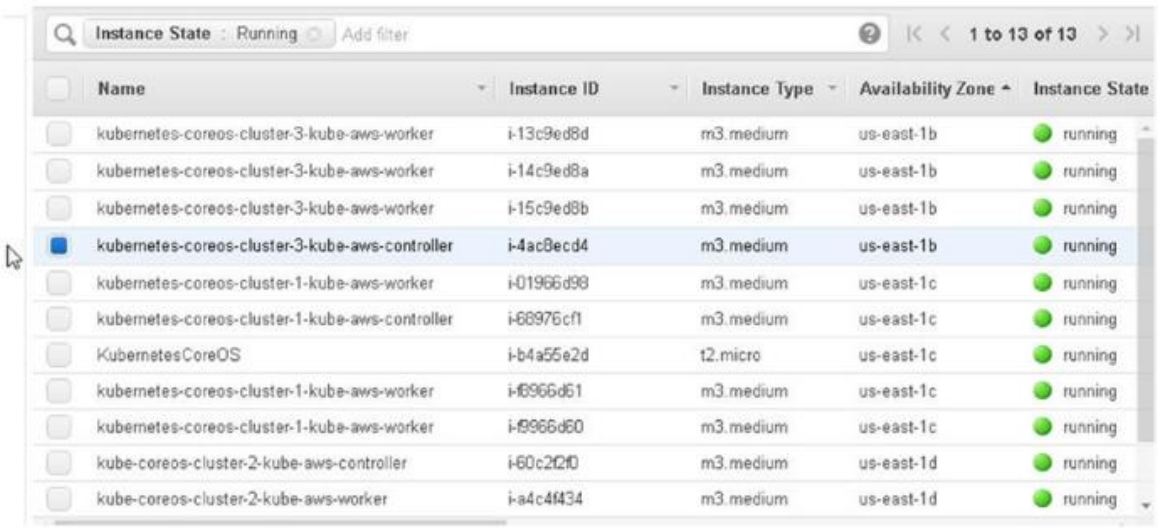
Gambar 16.6 Log Masuk SSH Ke Instans Pengontrol Ketiga

Daftarkan node kluster seperti yang ditunjukkan pada Gambar 16.7.

```
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
core@ip-10-0-0-50 ~ $ ./kubectl get nodes
NAME                          STATUS                     AGE
ip-10-0-0-189.ec2.internal    Ready                      1m
ip-10-0-0-190.ec2.internal    Ready                      1m
ip-10-0-0-191.ec2.internal    Ready                      1m
ip-10-0-0-50.ec2.internal     Ready,SchedulingDisabled   1m
core@ip-10-0-0-50 ~ $
```

Gambar 16.7 Node Untuk Kluster Kubernetes Ketiga

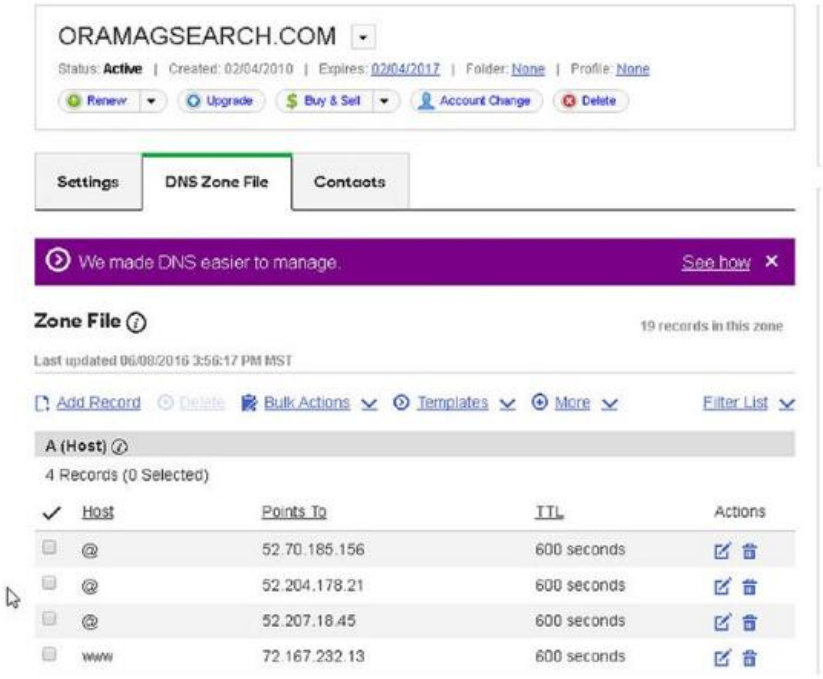
Setelah tiga CloudFormations dimulai, ketiga pengontrol harus berjalan di Konsol EC2, dengan setiap pengontrol mengelola tiga node pekerja seperti yang ditunjukkan pada Gambar 16.8.



Gambar 16.8 Instans EC2 Untuk Tiga Kluster Cloudformations For Kubernetes

16.3 MENGONFIGURASI DNS EKSTERNAL

Selanjutnya, tambahkan rekaman A untuk setiap instans pengontrol ke file zona domain oramagsearch.com (nama domain akan berbeda untuk pengguna yang berbeda) seperti yang ditunjukkan pada Gambar 16.9.



Gambar 16.9 Menambahkan Rekaman A Untuk Instans Pengontrol

Membuat Layanan Kubernetes

Pada bagian ini, kita akan membuat aplikasi hello-world dan mengekspos aplikasi tersebut sebagai layanan bertipe LoadBalancer pada masing-masing dari tiga kluster Kubernetes. Hasilnya, tiga penyeimbang beban elastis yang masing-masing mengekspos

layanan hello-world akan tersedia. Login SSH ke masing-masing instans pengontrol dan setelah memverifikasi bahwa node kluster terdaftar, buat aplikasi dengan citra Docker tutum/hello-world.

```
kubectl -s http://localhost:8080 run hello-world --
image=tutum/hello-world --replicas=2
--port=8
```

Daftar pod di seluruh klaster:

```
kubectl get pods –o wide
```

Daftar deployment:

```
kubectl get deployments
```

Selanjutnya, tampilkan deployment sebagai layanan bertipe LoadBalancer:

```
kubectl exposed deployment hello-world--port=80 --
type=LoadBalancer
```

Daftar layanan:

```
kubectl get services
```

Output dari perintah sebelumnya ditunjukkan pada Gambar 16.10.

```
core@ip-10-0-0-50 ~ $ ./kubectl -s http://localhost:8080 run hello-world --image
=tutum/hello-world --replicas=2 --port=80
deployment "hello-world" created
core@ip-10-0-0-50 ~ $ ./kubectl get pods -o wide
NAME                           READY     STATUS    RESTARTS   AGE       IP
    NODE
hello-world-3739649373-fz010   1/1       Running   0          33s       10.2.43.
3   ip-10-0-0-191.ec2.internal
hello-world-3739649373-xbrvl   1/1       Running   0          33s       10.2.6.2
    ip-10-0-0-190.ec2.internal
core@ip-10-0-0-50 ~ $ ./kubectl get deployments
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-world   2         2         2            2           1m
core@ip-10-0-0-50 ~ $ ./kubectl expose deployment hello-world --port=80 --type=L
oadBalancer
service "hello-world" exposed
core@ip-10-0-0-50 ~ $ ./kubectl get services
NAME          CLUSTER-IP   EXTERNAL-IP        PORT(S)   AGE
hello-world   10.3.0.66    a91023d495c29...   80/TCP    10s
kubernetes    10.3.0.1     <none>             443/TCP   3m
core@ip-10-0-0-50 ~ $
```

Gambar 16.10 Membuat Deployment Dan Layanan Hello-World

Jelaskan layanan:

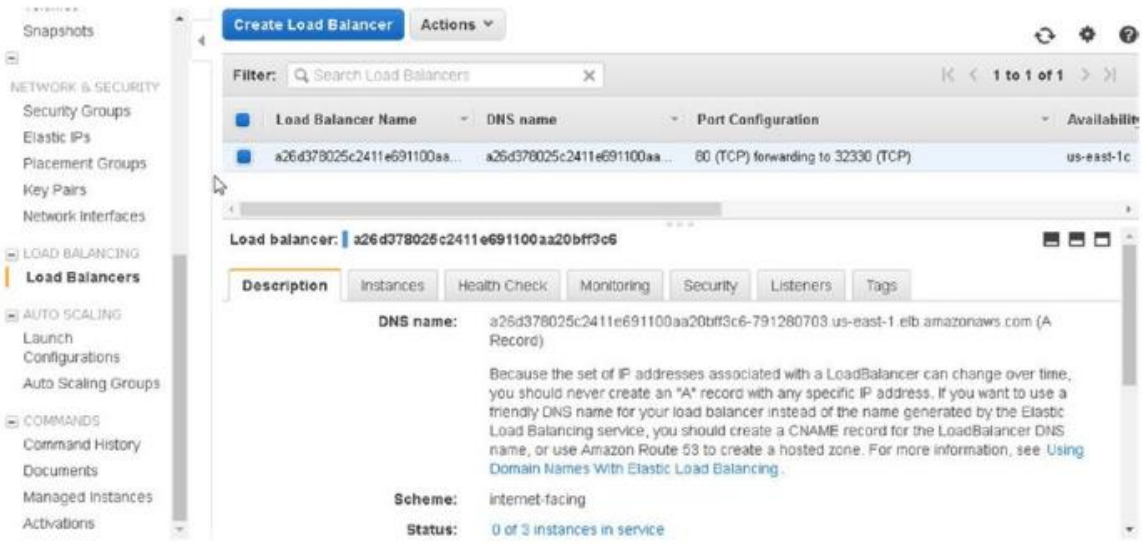
```
kubectl describe svc hello-world
```

Detail layanan, termasuk LoadBalancer Ingress, tercantum seperti yang ditunjukkan pada Gambar 16.11.

```
core@ip-10-0-0-50 ~ $ ./kubectl describe svc hello-world
Name:                   hello-world
Namespace:              default
Labels:                 run=hello-world
Selector:               run=hello-world
Type:                   LoadBalancer
IP:                     10.3.0.251
LoadBalancer Ingress:   a26d378025c2411e691100aa20bff3c6-791280703.us-east-1.elb
.amazonaws.com
Port:                   <unset> 80/TCP
NodePort:               <unset> 32330/TCP
Endpoints:              10.2.47.2:80,10.2.84.3:80
Session Affinity:       None
Events:
  FirstSeen     LastSeen        Count   From                    SubobjectPath  T
ype             Reason                  Message
  ---------     --------        -----   ----                    -------------  -
------- ------                  -------
  43s           43s             1       {service-controller }                  N
ormal           CreatingLoadBalancer    Creating load balancer
  41s           41s             1       {service-controller }                  N
ormal           CreatedLoadBalancer     Created load balancer

core@ip-10-0-0-50 ~ $
```

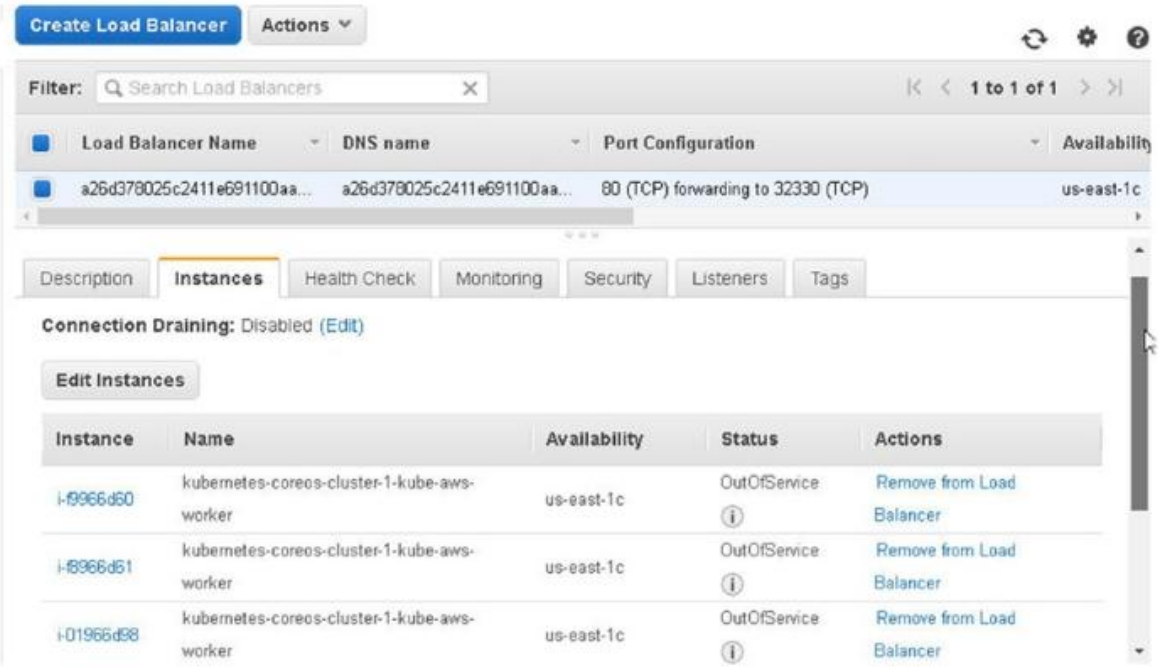
Gambar 16.11 Menjelaskan Layanan Hello-World

Load balancer elastis juga harus tercantum dalam tampilan EC2 Console ➤ LOAD BALANCING ➤ Load Balancers. Nama DNS Publik dari load balancer diperoleh dari EC2 Console seperti yang ditunjukkan pada Gambar 16.12.



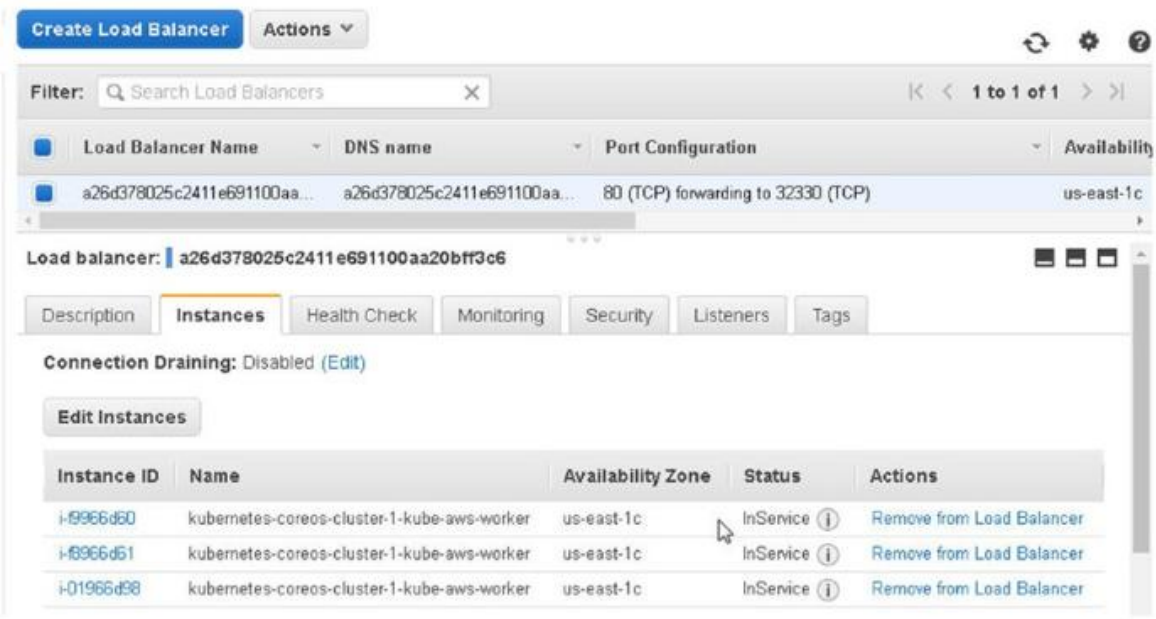
Gambar 16.12 Loadbalancer Untuk Suatu Layanan

Tab Instances mencantumkan instans EC2 yang sedang diseimbangkan bebannya oleh load balancer. Awalnya Statusnya mungkin OutOfService seperti yang ditunjukkan pada Gambar 16.13.



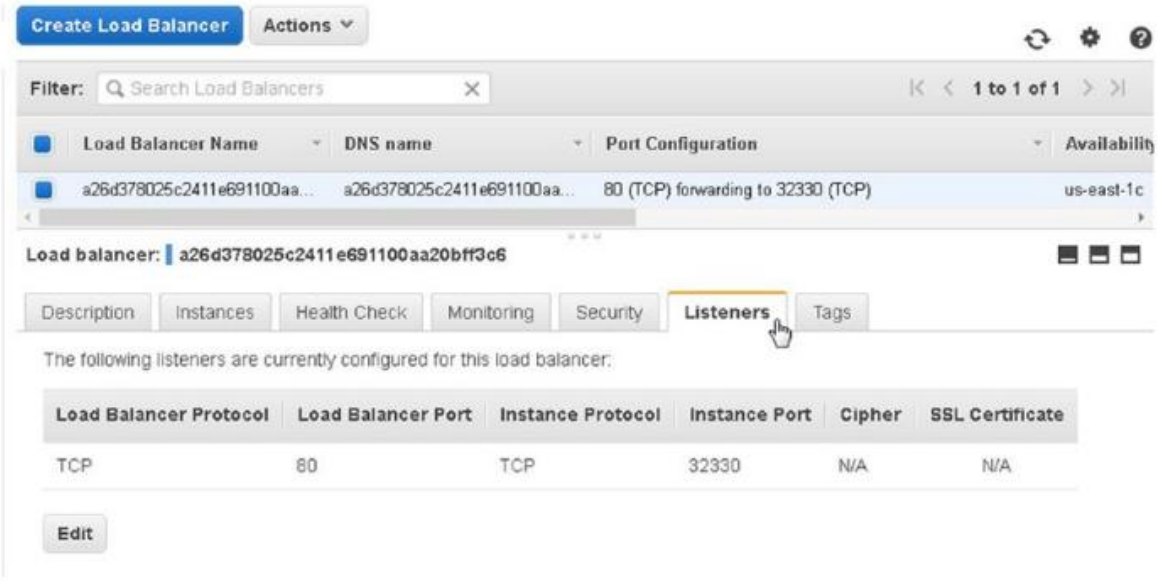
Gambar 16.13 Awalnya Instans Dalam Loadbalancer Mungkin Outofservice

Setelah sekitar satu menit, Status akan menjadi InService seperti yang ditunjukkan pada Gambar 16.14.



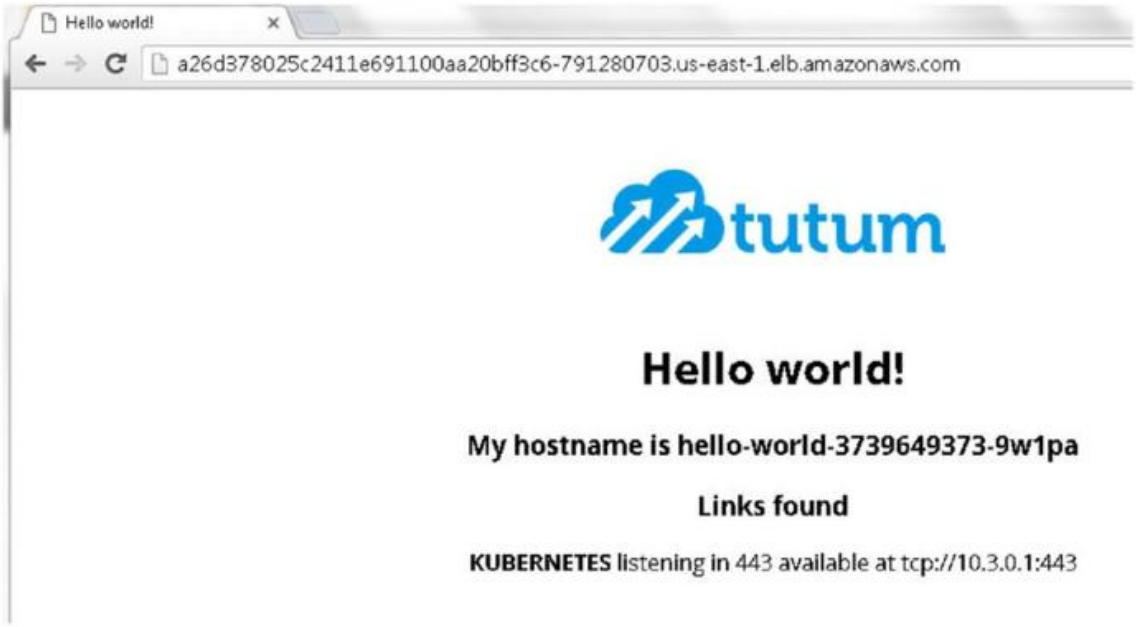
Gambar 16.14 Instans Loadbalancer Inservice

Tab Listeners akan mencantumkan listener load balancer seperti yang ditunjukkan pada Gambar 16.15.



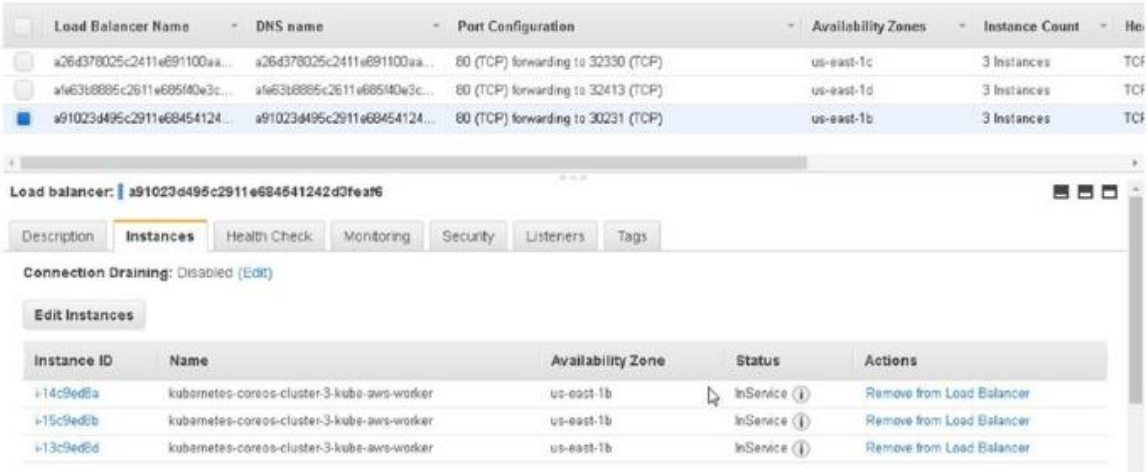
Gambar 16.15 Listeners Untuk Loadbalancer

Panggil nama DNS Publik di peramban web. Output aplikasi hello-world akan ditampilkan seperti yang ditunjukkan pada Gambar 16.16.



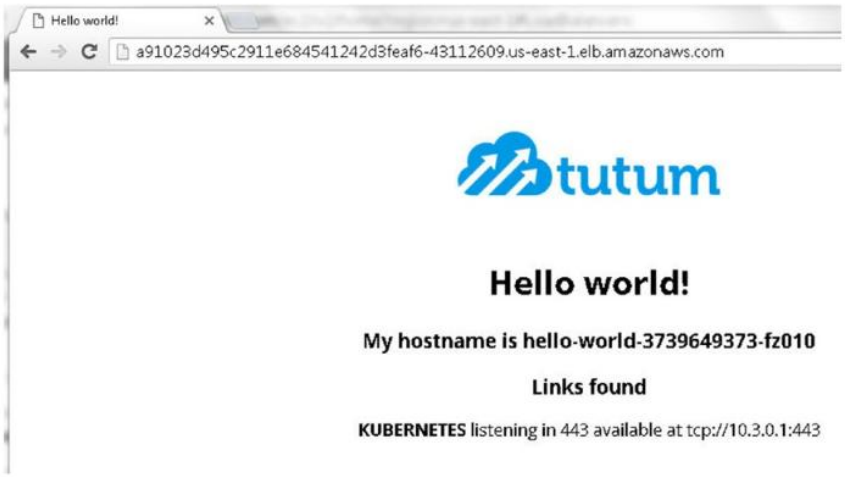
Gambar 16.16 Memanggil DNS Publik Di Peramban

Ketika layanan Kubernetes hello-world berjenis LoadBalancer telah dibuat di setiap kluster Kubernetes, tiga penyeimbang beban elastis harus dibuat seperti yang ditunjukkan di Konsol EC2 pada Gambar 16.17.



Gambar 16.17 Tiga Loadbalancer, Satu Untuk Setiap Cloudformation

Nama DNS Publik untuk setiap ELB harus menampilkan hasil untuk aplikasi hello-world seperti yang ditunjukkan pada Gambar 16.18.

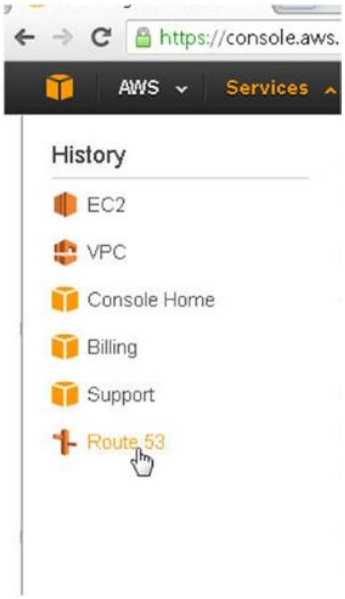


Gambar 16.18 Memanggil DNS Publik Untuk Elastic Load Balancer Lainnya

16.4 MEMBUAT LAYANAN AWS ROUTE 53

Di bagian ini, kita akan membuat layanan AWS Route 53 untuk merutekan permintaan pengguna ke domain oramagsearch.com ke elastic load balancer, khususnya nama DNS publik ELB. Kita akan membuat dua set rekaman sumber daya, yang menunjuk ke dua ELB berbeda yang dikonfigurasi untuk failover, dengan salah satu ELB menjadi set rekaman sumber daya utama dan yang lainnya menjadi set rekaman sekunder.

Saat domain oramagsearch.com dipanggil di browser web, layanan AWS Route 53 merutekan permintaan ke set rekaman sumber daya utama. Jika set rekaman utama menjadi tidak tersedia, layanan merutekan permintaan pengguna ke set rekaman sekunder, yang pada dasarnya menyediakan ketersediaan tinggi aplikasi web Hello World di domain oramagsearch.com. Untuk membuat layanan AWS Route 53, pilih Route 53 di Layanan AWS seperti yang ditunjukkan pada Gambar 16.19.



Gambar 16.19 Memilih Layanan Route 53

Membuat Hosted Zone

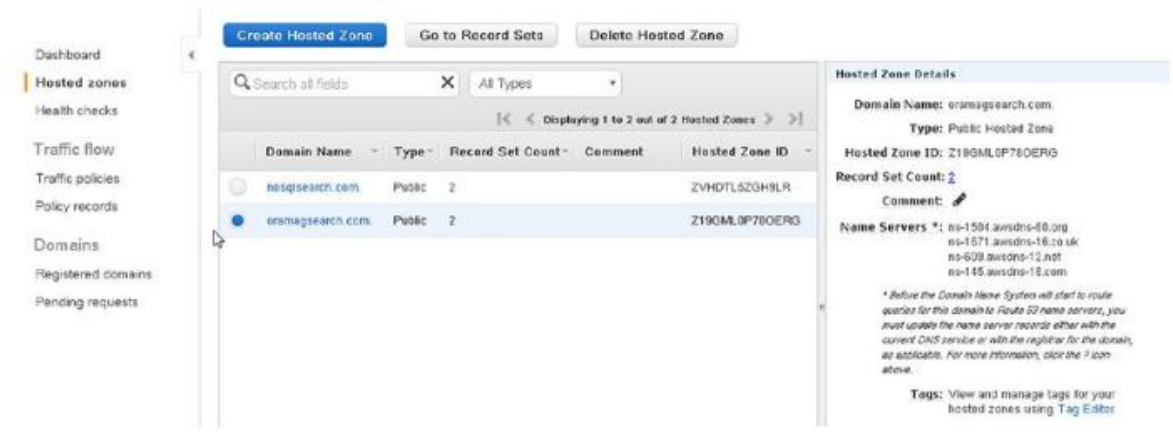
Untuk membuat hosted zone, pilih Hosted Zones di margin dan klik Create Hosted Zone seperti yang ditunjukkan pada Gambar 16.20.

Gambar 16.20 Membuat Hosted Zone

Dalam dialog Buat Zona Hosting, tentukan Nama Domain (oramagsearch.com). Nama domain harus didaftarkan dengan pengguna. Pilih Zona Hosting Publik sebagai jenisnya, seperti yang ditunjukkan pada Gambar 16.21.

Gambar 16.21 Mengonfigurasi Zona Host

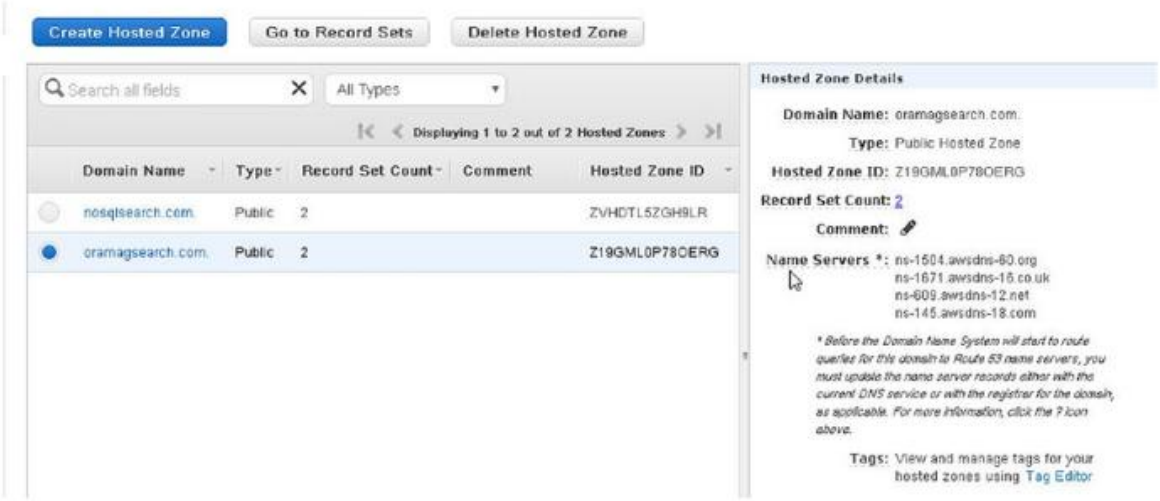
Zona host publik baru dibuat. Server nama untuk zona host juga ditetapkan, seperti yang ditunjukkan pada Gambar 16.22.



Gambar 16.22 Zona Host Publik Baru

Mengonfigurasi Server Nama pada Nama Domain

Selanjutnya, kita perlu memperbarui rekaman server nama untuk domain oramagsearch.com dengan pendaftar domain sehingga Sistem Nama Domain dapat merutekan permintaan untuk domain ke server nama Route 53. Salin server nama Route 53 seperti yang ditunjukkan pada Gambar 16.23.



Gambar 16.23 Name Server Route 53

Catatan name server default untuk domain biasanya disediakan oleh pendaftar domain seperti yang ditunjukkan pada Gambar 16.24.

Gambar 16.24 Name Server Domain

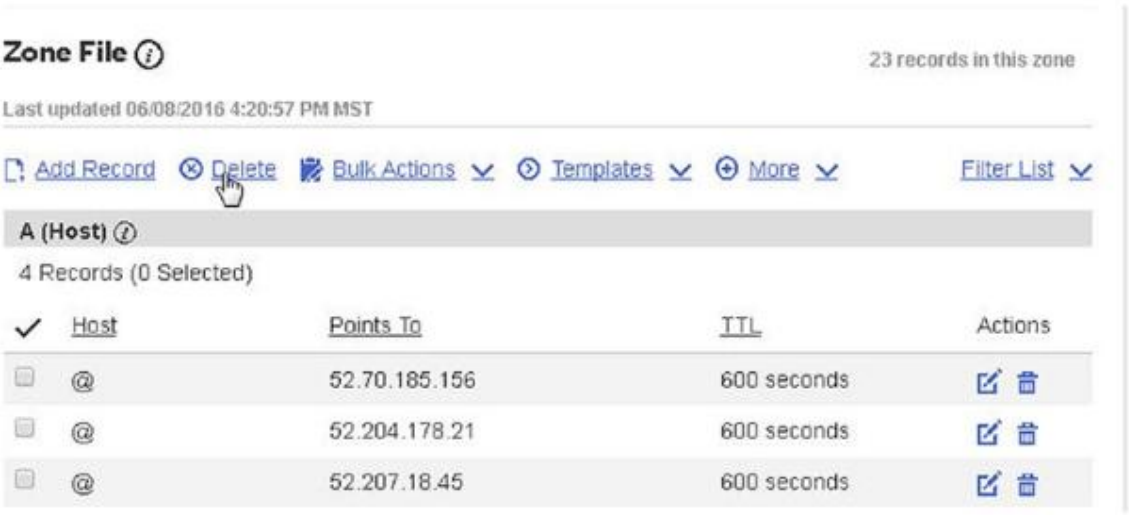
Tambahkan name server untuk layanan Route 53 ke catatan NS domain seperti yang ditunjukkan pada Gambar 16.25.

Gambar 16.25 Menambahkan Name Server Untuk Route 53 Ke Catatan DNS

Berikutnya, pilih server nama default yang disediakan oleh pendaftar domain seperti yang ditunjukkan pada Gambar 16.26.

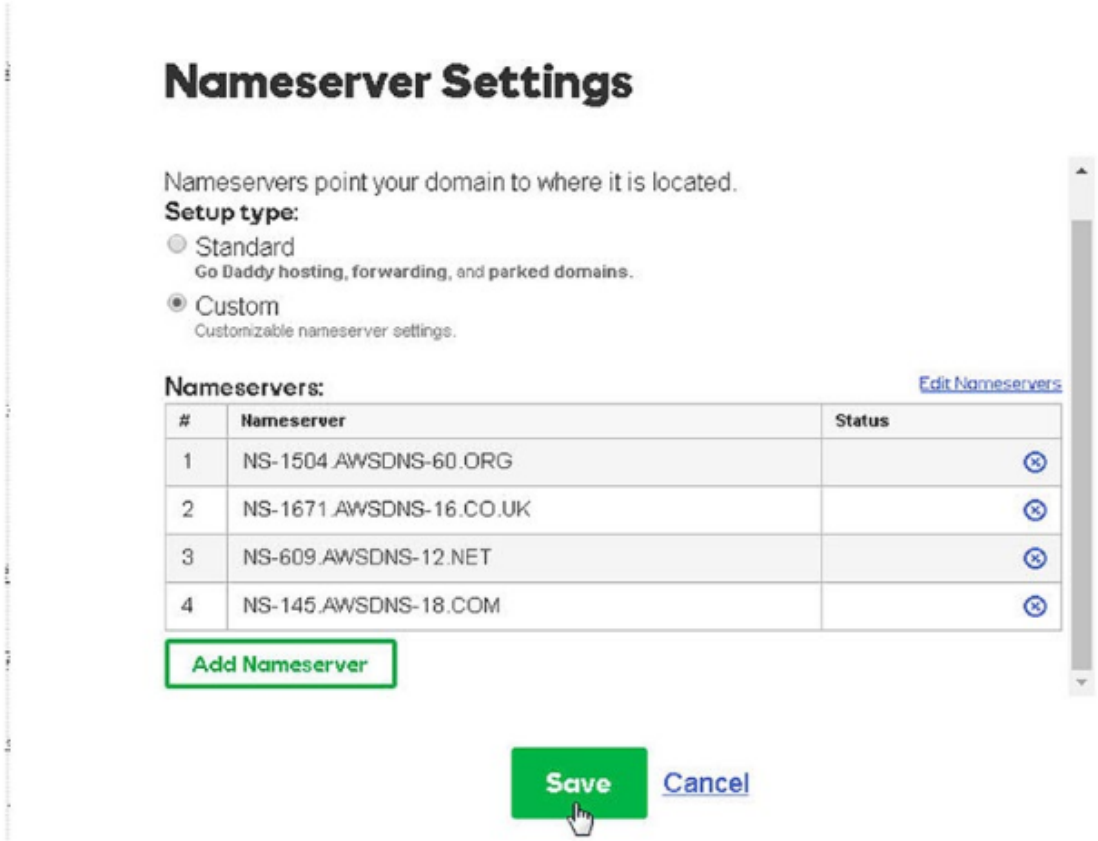
Gambar 16.26 Memilih Server Nama Default Pada Domain

Klik Hapus untuk menghapus server nama default seperti yang ditunjukkan pada Gambar 16.27.



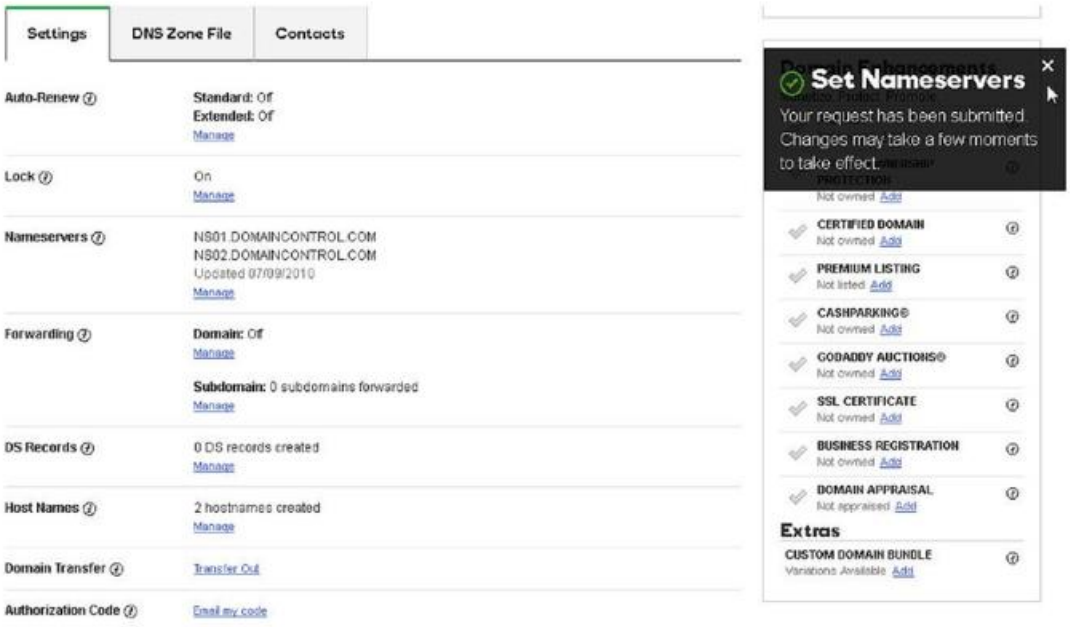
Gambar 16.27 Menghapus Server Nama Default

Simpan pengaturan server nama kustom seperti yang ditunjukkan pada Gambar 16.28.



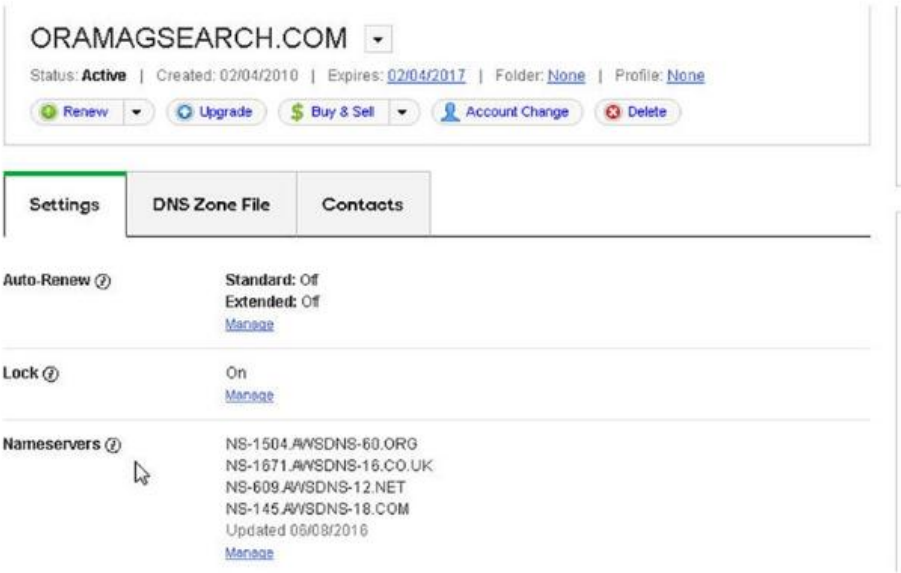
Gambar 16.28 Server Nama Domain

Pengaturan baru mungkin memerlukan waktu beberapa saat untuk diterapkan, seperti yang ditunjukkan oleh pesan pada Gambar 16.29.



Gambar 16.29 Memperbarui Nameserver Domain Dapat Memakan Waktu Cukup Lama

Saat rekaman nameserver baru telah berlaku, rekaman NS akan menunjukkan hal yang sama seperti yang ditunjukkan pada Gambar 16.30.



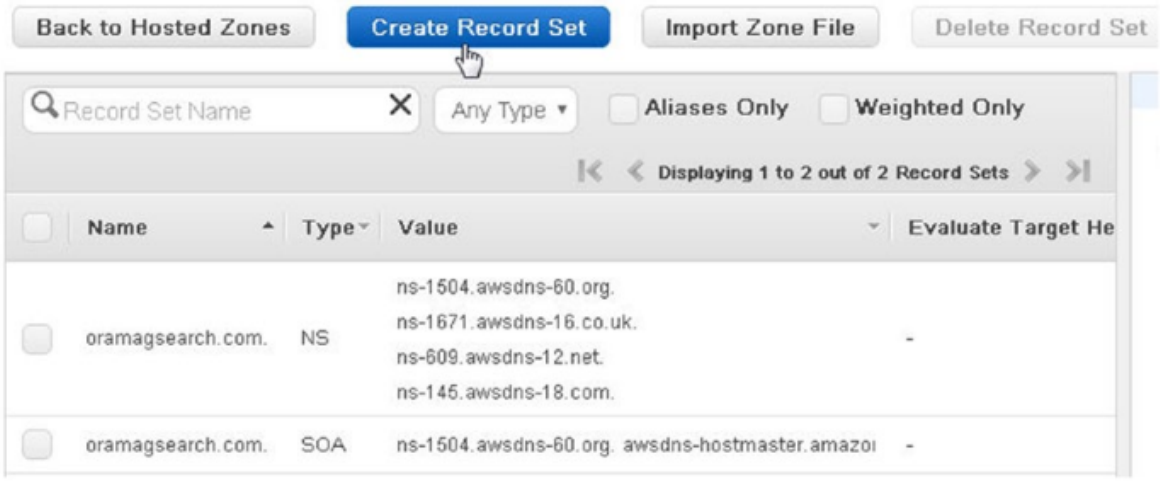
Gambar 16.30 Nameserver Domain Yang Dikonfigurasi

Membuat Set Rekaman

Selanjutnya, kita akan membuat set rekaman sumber daya untuk menunjuk ke penyeimbang beban elastis untuk layanan hello-world. Klik Buka Set Rekaman seperti yang ditunjukkan pada Gambar 16.31.

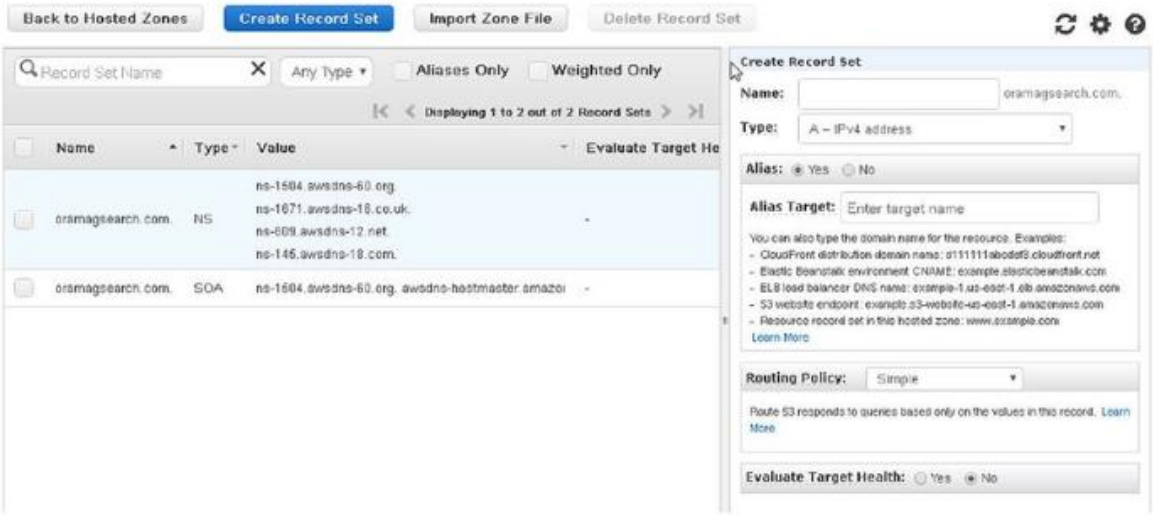
Gambar 16.31 Mulailah Membuat Set Rekaman Dengan Mengklik Buka Set Rekaman

Kemudian klik Buat Set Rekaman seperti yang ditunjukkan pada Gambar 16.32.



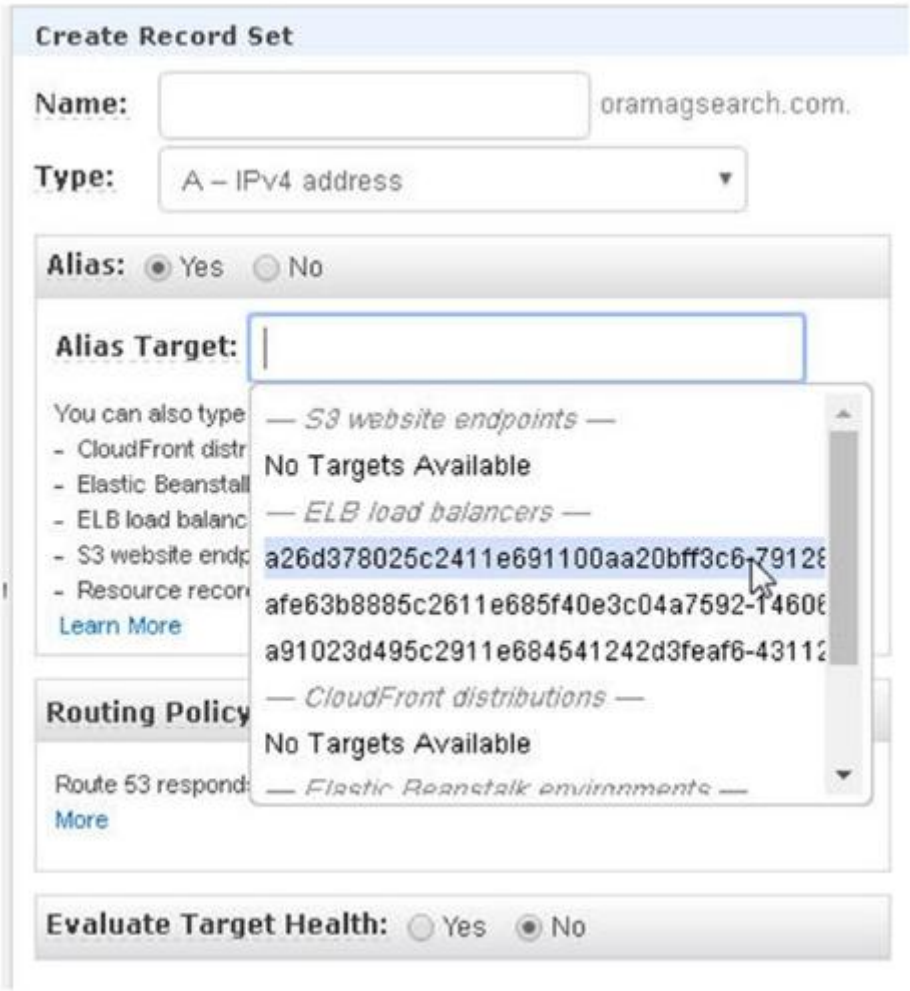
Gambar 16.32 Mengklik Buat Set Rekaman

Dalam dialog Buat Set Rekaman, tetapkan Jenis sebagai A - alamat IPv4 seperti yang ditunjukkan pada Gambar 16.33. Pilih tombol radio Alias.



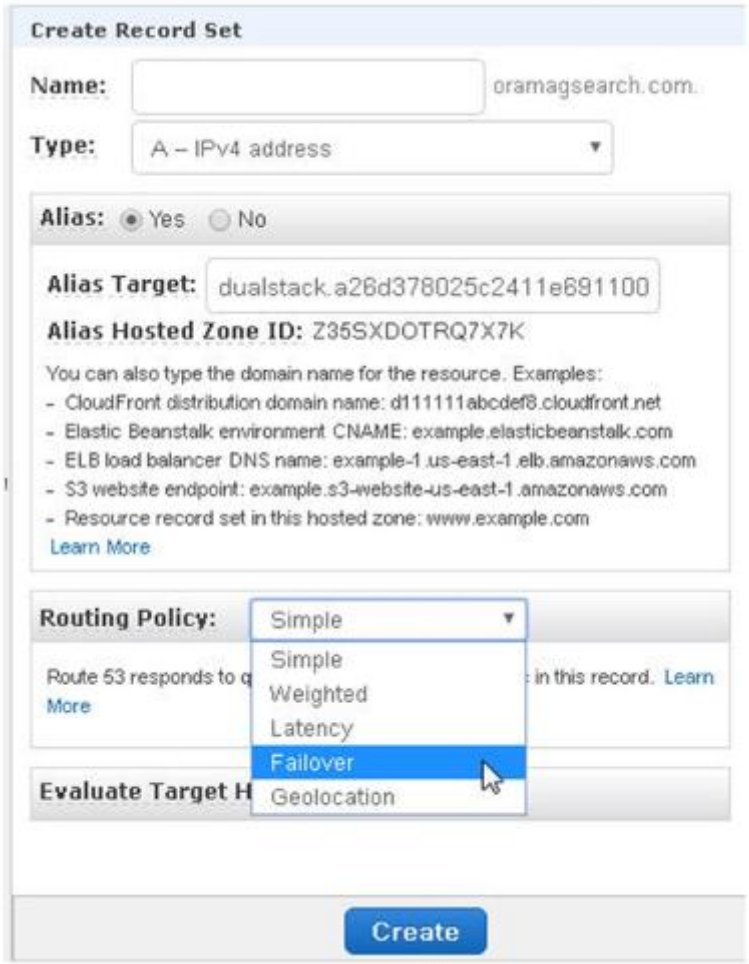
Gambar 16.33 Menetapkan Jenis Dalam Dialog Buat Kumpulan Rekaman

Klik di kolom Target Alias untuk menampilkan menu tarik-turun untuk target. Pilih salah satu Penyeimbang Beban ELB, dengan asumsi bahwa semua Penyeimbang Beban ELB ditujukan untuk layanan hello-world seperti yang ditunjukkan pada Gambar 16.34.



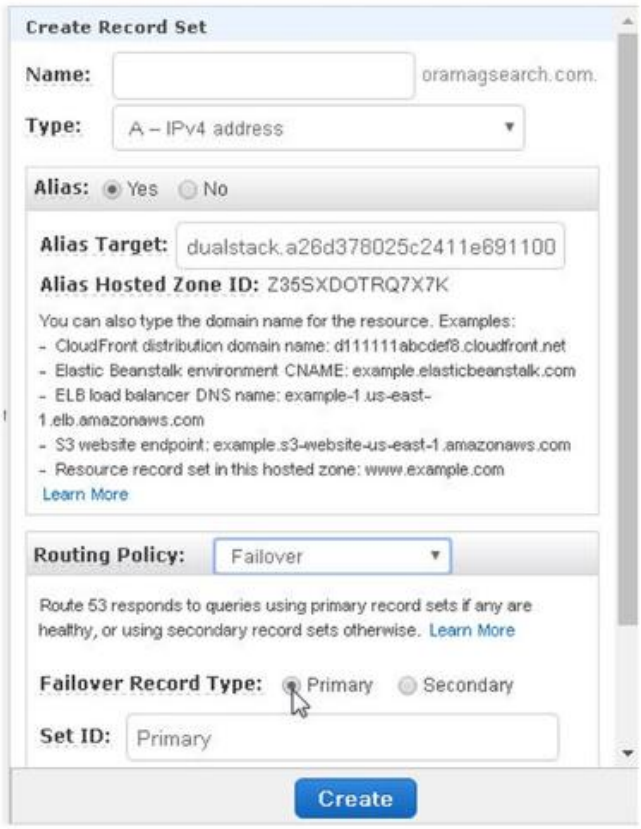
Gambar 16.34 Memilih Salah Satu ELB Load Balancer Sebagai Target Alias

Untuk Kebijakan Perutean, pilih Failover seperti yang ditunjukkan pada Gambar 16.35.



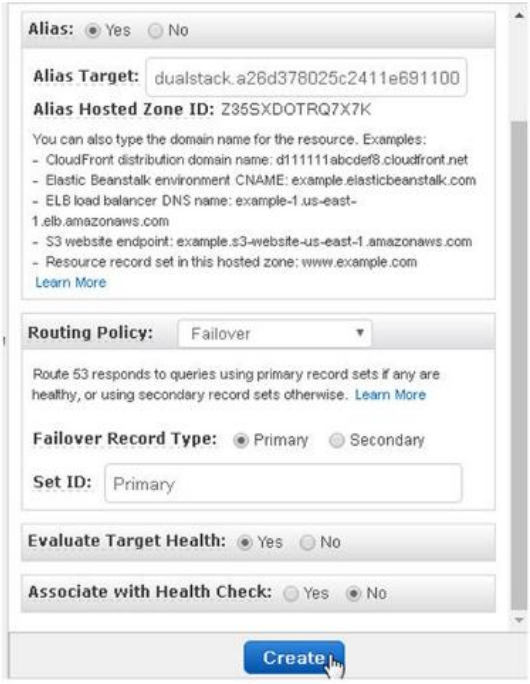
Gambar 16.35 Memilih Failover sebagai Kebijakan Perutean

Untuk Jenis Rekaman Failover, pilih Utama seperti yang ditunjukkan pada Gambar 16.36.



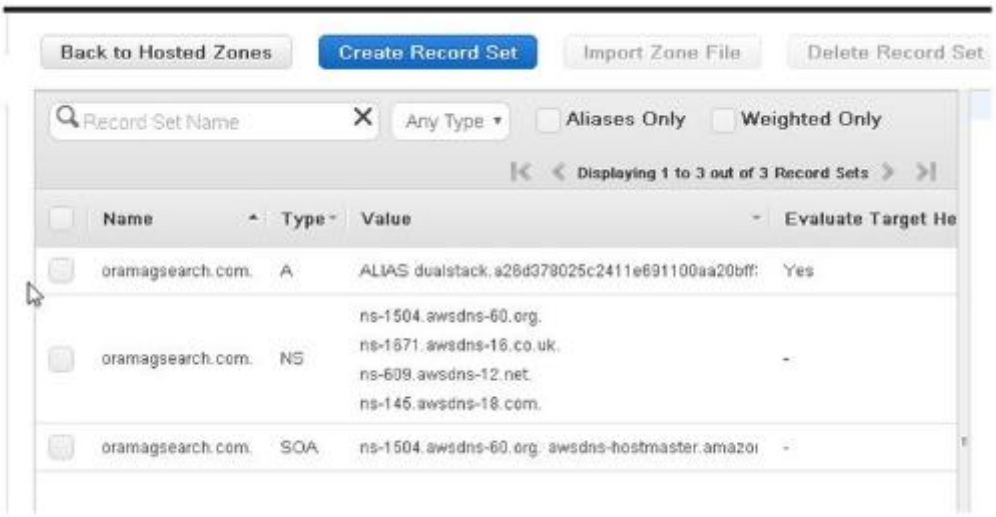
Gambar 16.36 Menetapkan Jenis Rekaman Failover Sebagai Utama

Untuk Mengevaluasi Kesehatan Target, pilih Ya. Untuk Mengaitkan dengan Pemeriksaan Kesehatan, pilih Tidak. Klik Buat seperti yang ditunjukkan pada Gambar 16.37.



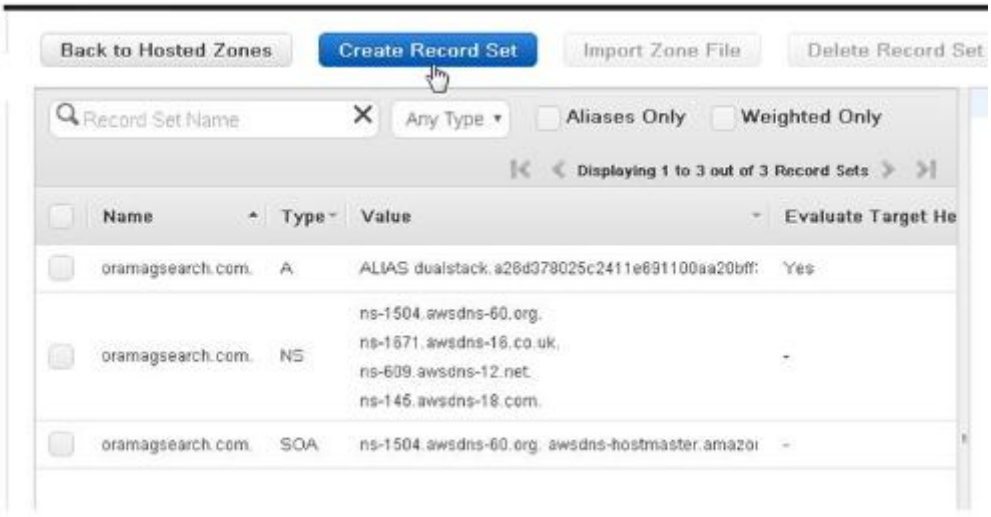
Gambar 16.37 Membuat Set Rekaman

Set rekaman sumber daya baru ditambahkan seperti yang ditunjukkan pada Gambar 16.38.



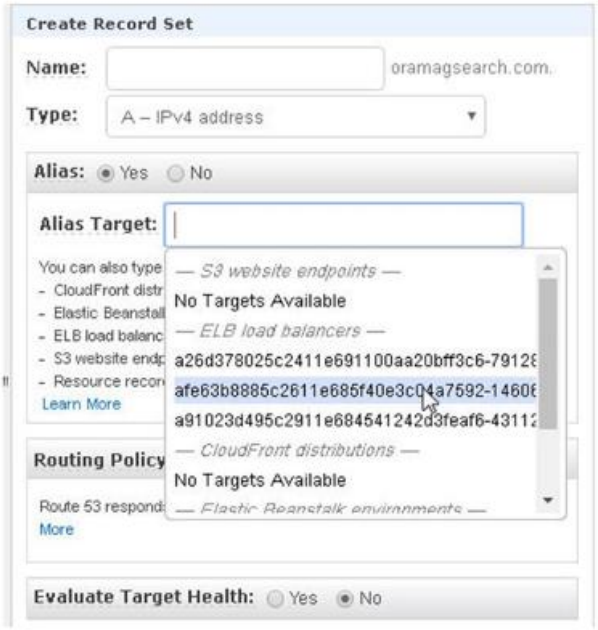
Gambar 16.38 Kumpulan Rekaman Sumber Daya

Klik Buat Kumpulan Rekaman untuk membuat kumpulan rekaman sumber daya lain seperti yang ditunjukkan pada Gambar 16.39. Dalam kebijakan perutean Failover, dua kumpulan rekaman sumber daya perlu dikonfigurasi, satu sebagai primer dan yang lainnya sebagai sekunder. Jika kumpulan rekaman primer tidak tersedia, Route 53 akan merutekan permintaan apa pun untuk zona yang dihosting ke kumpulan rekaman sekunder.



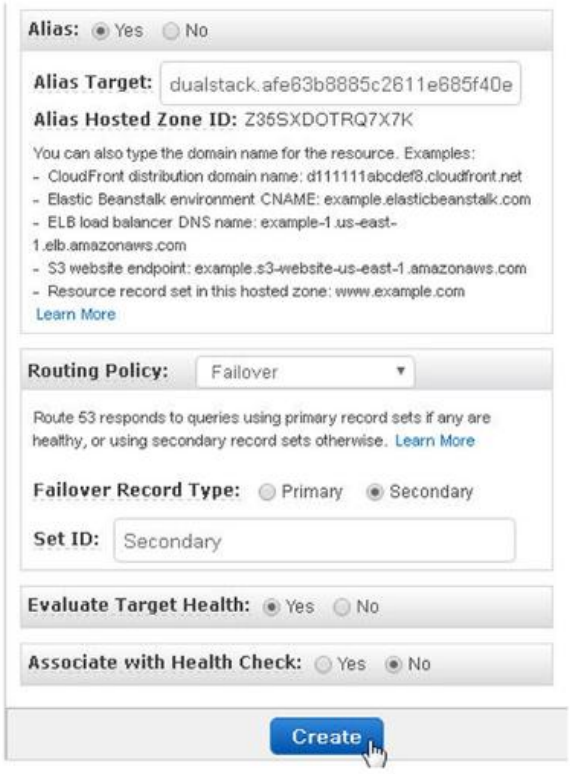
Gambar 16.39 Mengklik Buat Kumpulan Rekaman Untuk Membuat Kumpulan Rekaman Lain

Pada Create Record Set, tetapkan Type sebagai alamat A -IPv4 dan Alias sebagai Yes. Untuk Alias Target pilih ELB Load Balancer yang berbeda seperti yang ditunjukkan pada Gambar 16.40.



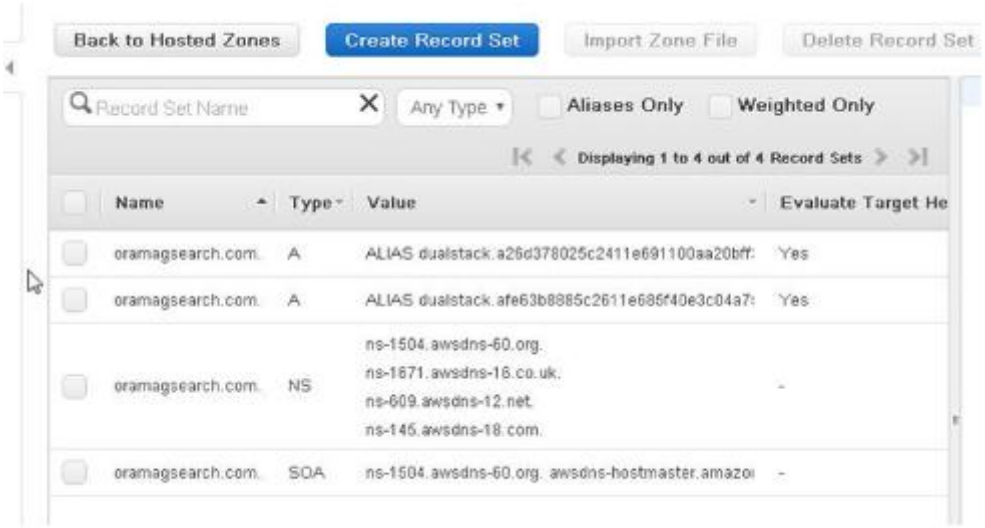
Gambar 16.40 Mengonfigurasi Record Set

Tetapkan Routing Policy sebagai Failover. Pilih Secondary sebagai Failover Record Type. Tetapkan Evaluate Target Health sebagai Yes dan Associate with Health Check sebagai No. Klik Create seperti yang ditunjukkan pada Gambar 16.41.



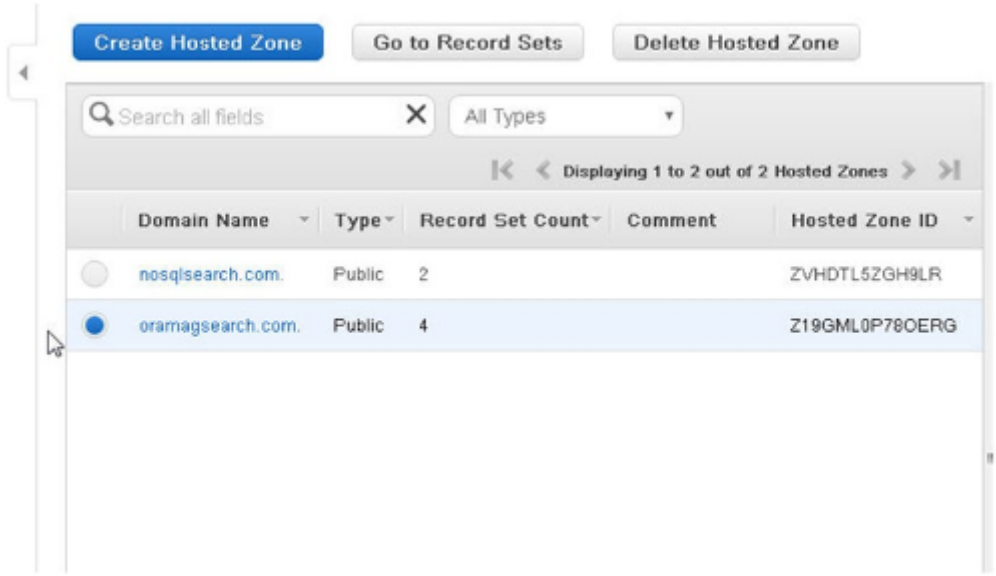
Gambar 16.41 Membuat Record Set Kedua

Record set sumber daya kedua ditambahkan, seperti yang ditunjukkan pada Gambar 16.42.



Gambar 16.42 Record Set Sumber Daya Kedua

Public Hosted Zone untuk domain oramagsearch.com (nama domain akan berbeda untuk pengguna yang berbeda) harus mencantumkan Record Set Count sebagai 4, bukan 2 untuk memulai, seperti yang ditunjukkan pada Gambar 16.43.

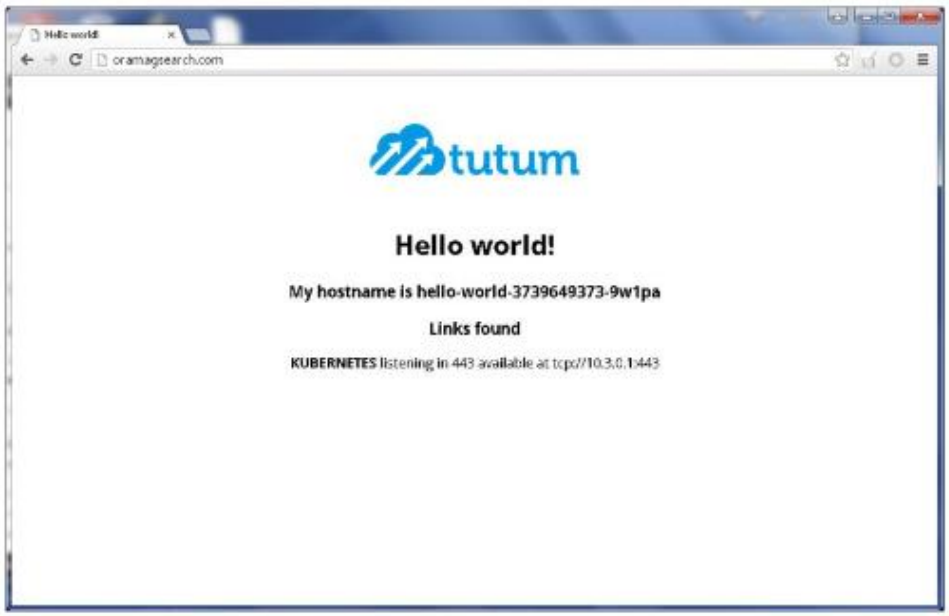


Gambar 16.43 Zona Yang Dihosting Publik Dengan Empat Set Rekaman

16.5 MENGUJI KETERSEDIAAN TINGGI

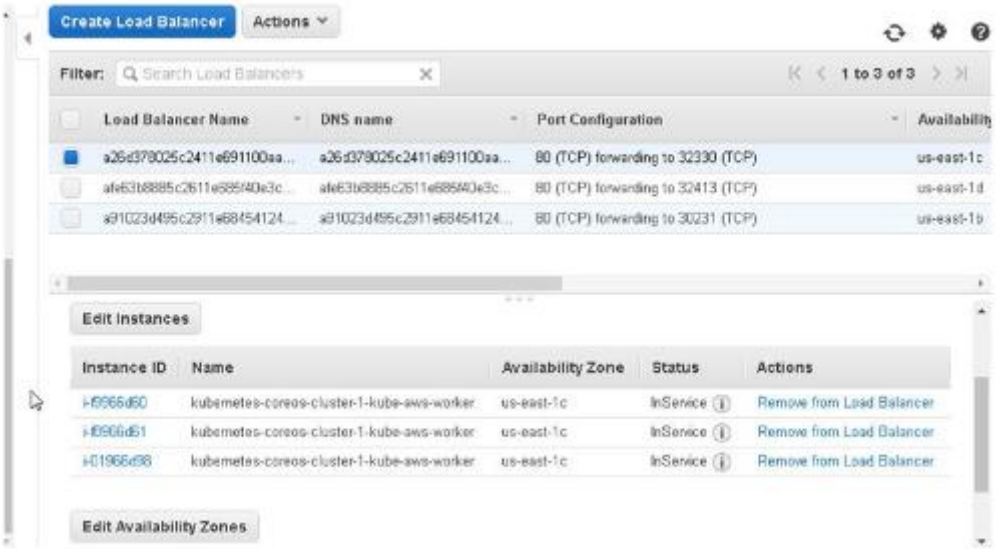
Selanjutnya, kami akan menunjukkan ketersediaan tinggi. Buka domain oramagsearch.com (nama domain akan berbeda untuk pengguna/grup pengguna yang berbeda) di browser web. Set rekaman sumber daya utama untuk zona yang dihosting publik dipanggil, yang menunjuk ke salah satu penyeimbang beban elastis untuk layanan Kubernetes

hello-service, dan hasil aplikasi tutum/hello-world ditampilkan seperti yang ditunjukkan pada Gambar 16.44.



Gambar 16.44 Memanggil Domain Di Browser

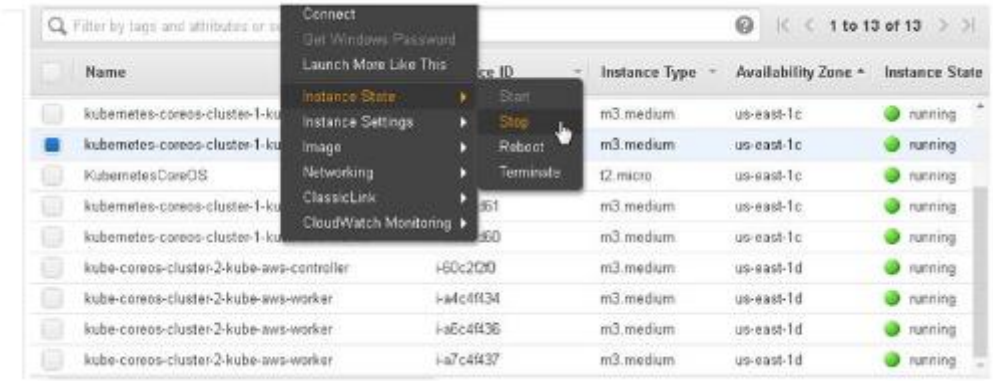
Set rekaman sumber daya utama menunjuk ke salah satu penyeimbang beban elastis, dan penyeimbang beban harus tersedia seperti yang ditunjukkan oleh semua instans terdaftar yang sedang dalam Layanan, seperti yang ditunjukkan pada Gambar 16.45.



Gambar 16.45 Loadbalancer Untuk Set Rekaman Sumber Daya Utama Dengan Semua Instans Inservice

Untuk menunjukkan ketersediaan tinggi, hentikan pengontrol untuk kluster Kubernetes yang

mengekspos penyeimbang beban elastis yang ditunjuk oleh set rekaman sumber daya utama, seperti yang ditunjukkan pada Gambar 16.46.



Gambar 16.46 Menghentikan Instans Pengontrol Untuk Kluster Yang Mengekspos ELB Yang Ditunjuk Oleh Set Rekaman Sumber Daya Utama

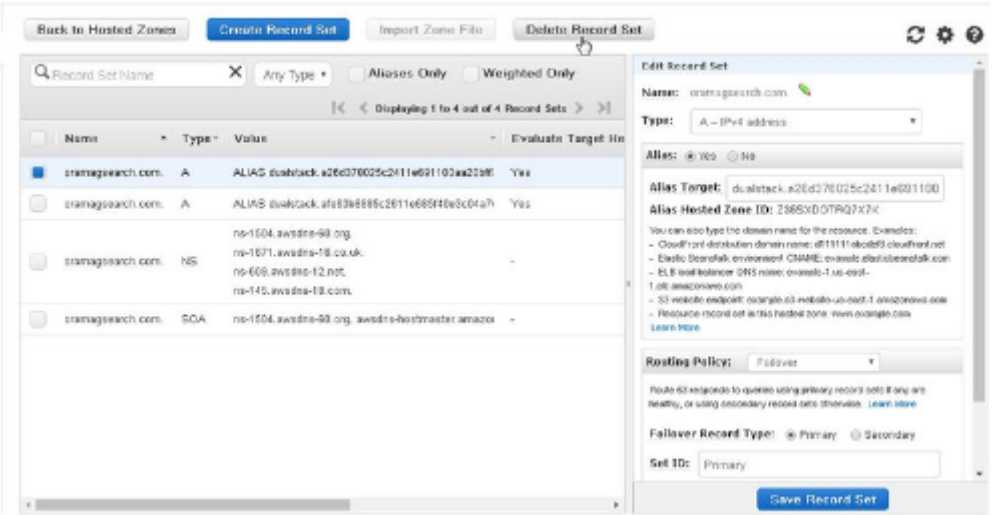
Instans pengontrol dan instans node pekerja harus dihentikan, seperti yang ditunjukkan pada Gambar 16.47, yang pada dasarnya membuat penyeimbang beban elastis untuk set rekaman sumber daya utama tidak tersedia. Jika zona yang dihosting hanya memiliki satu set rekaman sumber daya tanpa perutean Failover yang dikonfigurasi, domain oramagsearch.com akan menjadi tidak tersedia.

Gambar 16.47 Menghentikan Instans Pengontrol Dan Instans Pekerja Untuk Set Rekaman Sumber Daya Utama Cloudformation

Namun, zona yang dihosting oramagsearch.com gagal beralih ke set rekaman sumber daya sekunder dan terus melayani layanan hello-world, seperti yang ditunjukkan pada Gambar 16.48. Seperti yang ditunjukkan oleh output di browser, nama host telah berubah (nama host juga dapat berubah karena layanan mendistribusikan lalu lintas antara Pod pada penyebaran yang sama) tetapi layanan tetap tersedia.

Gambar 16.48 Zona Host Oramagsearch.Com Gagal Ke Set Rekaman Sumber Daya Sekunder Dan Terus Melayani

Ketika set rekaman sumber daya utama menjadi tidak tersedia dan permintaan pengguna diarahkan ke set rekaman sekunder, pada dasarnya layanan dilayani oleh satu set rekaman dan dengan demikian tidak lagi tersedia secara luas. Untuk membuat layanan tersedia secara luas, kita perlu membuat set rekaman utama menunjuk ke penyeimbang beban elastis yang berbeda atau menghapus dan membuat set rekaman baru. Dengan menggunakan pendekatan kedua, pilih set rekaman sumber daya utama dan klik Hapus Set Rekaman seperti yang ditunjukkan pada Gambar 16.49.

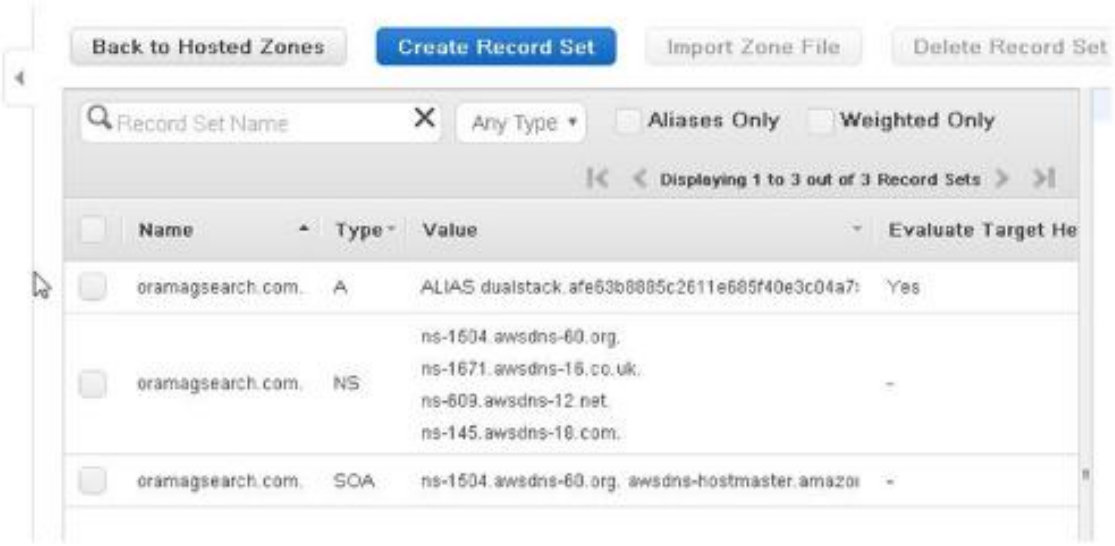


Gambar 16.49 Menghapus Set Rekaman Sumber Daya Utama

Klik Konfirmasi dalam dialog Konfirmasi seperti yang ditunjukkan pada Gambar 16.50.

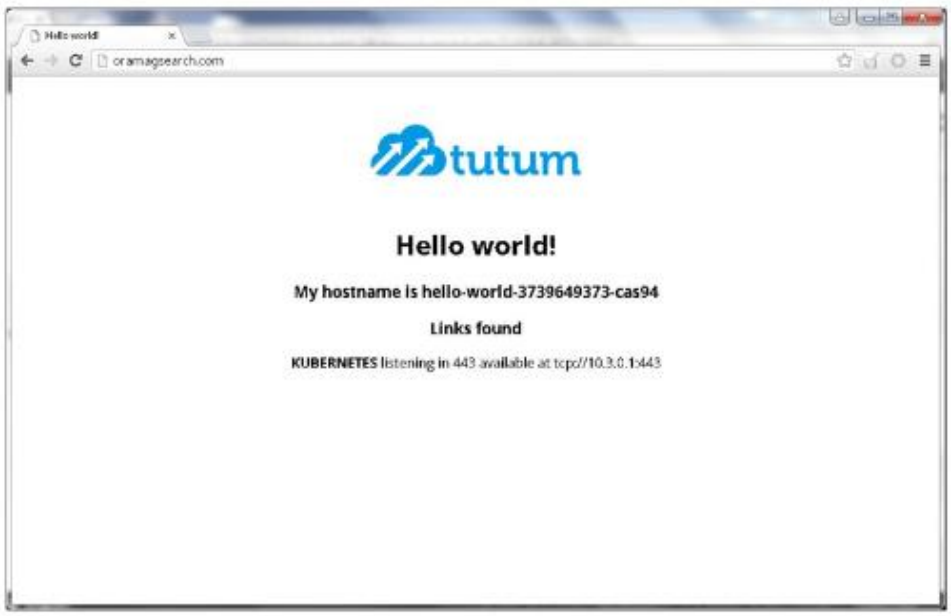
Gambar 16.50 Dialog Konfirmasi

Hanya set rekaman sekunder yang tersedia untuk mengarahkan permintaan pengguna, seperti yang ditunjukkan pada Gambar 16.51.



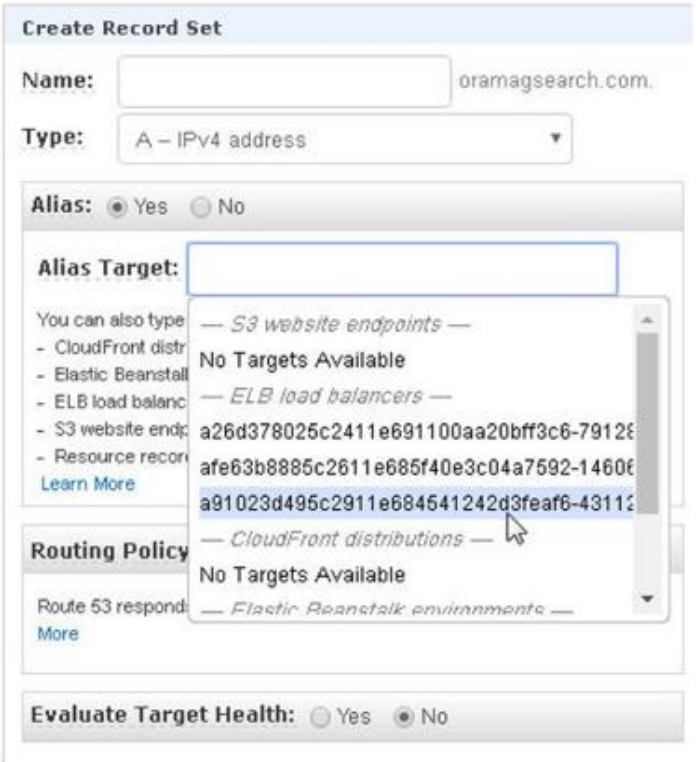
Gambar 16.51 Hanya Set Rekaman Sekunder Yang Tersedia

Layanan terus dilayani di oramagsearch.com seperti yang ditunjukkan pada Gambar 16.52. Nama host mungkin telah berubah, karena penyeimbang beban juga menyeimbangkan beban antara dua replika dalam penyebaran.



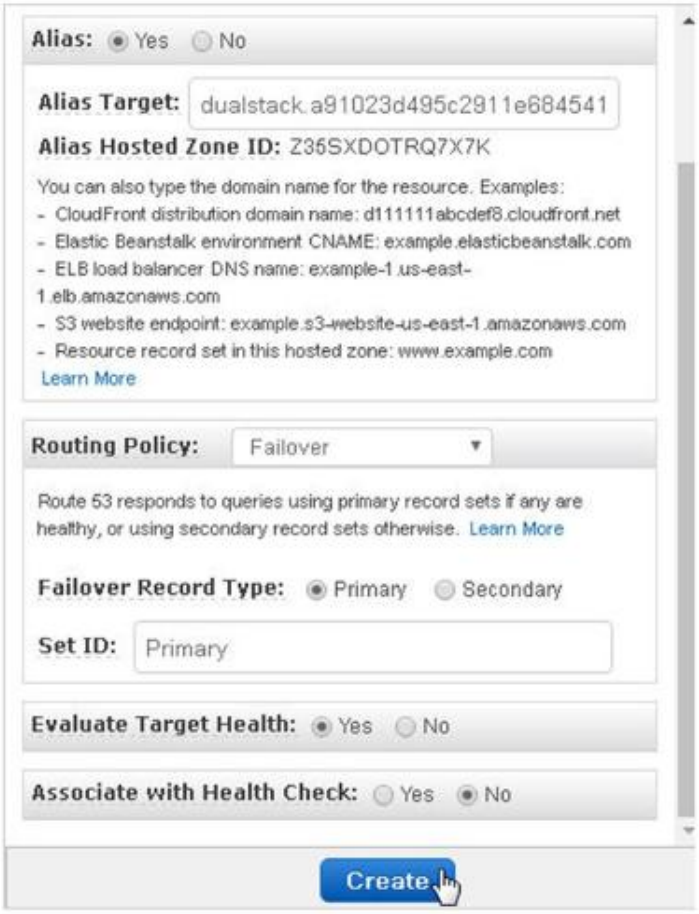
Gambar 16-52. Zona Yang Dihosting Dilayani Oleh Set Rekaman Sekunder

Untuk menambahkan set rekaman sumber daya utama, klik Buat Set Rekaman seperti yang ditunjukkan pada Gambar 16.53. Dalam dialog Buat Set Rekaman, tetapkan Jenis sebagai A - alamat IPv4. Tetapkan Alias sebagai Ya dan pilih penyeimbang beban elastis ketiga di Target Alias.



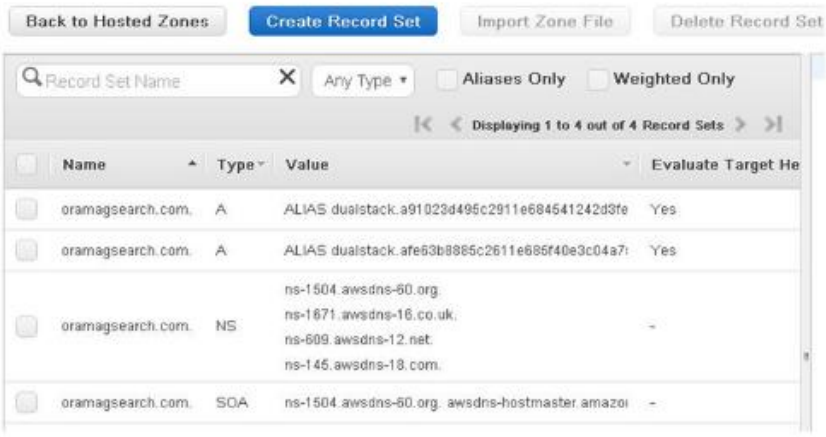
Gambar 16.53 Menambahkan Kembali Set Rekaman Sumber Daya Utama

Tetapkan Kebijakan Perutean sebagai Failover dan Jenis Rekaman Failover sebagai Utama. Dengan pengaturan lain yang sama seperti saat set rekaman Utama/Sekunder dibuat, klik Buat seperti yang ditunjukkan pada Gambar 16.54.



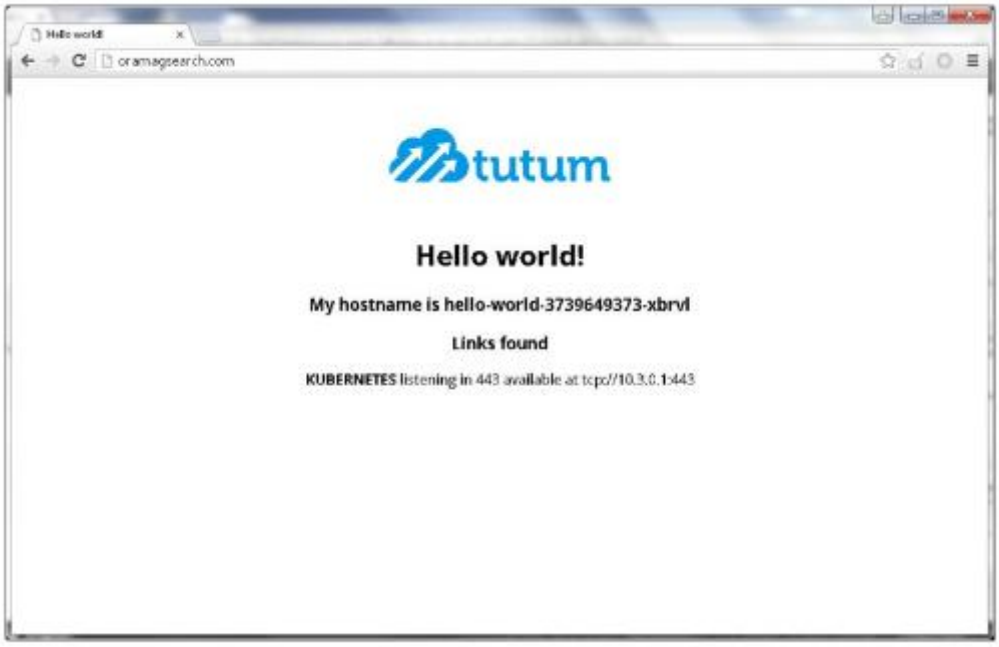
Gambar 16.54 Membuat Set Rekaman Sumber Daya Utama

Set Rekaman Utama ditambahkan seperti yang ditunjukkan pada Gambar 16.55.

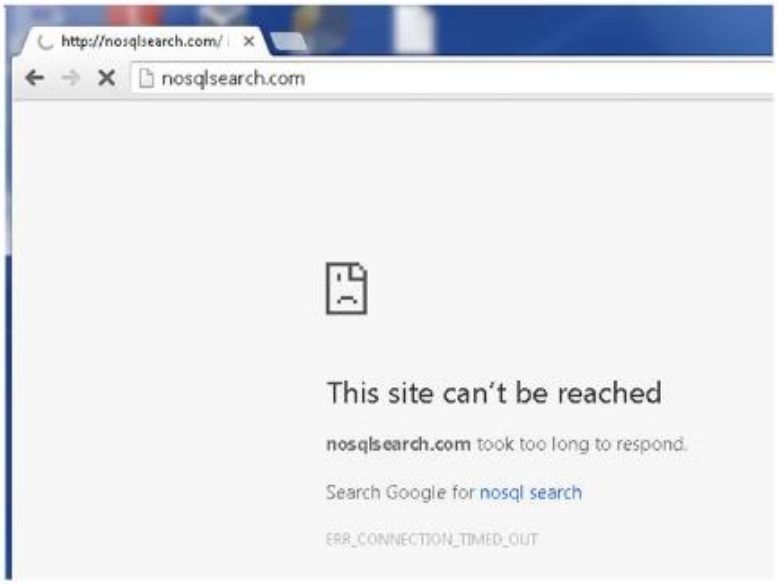


Gambar 16.55 Kumpulan Rekaman Sumber Daya Utama Yang Baru

Permintaan peramban web diarahkan ke kumpulan rekaman sumber daya utama, seperti yang ditunjukkan pada Gambar 16.56.



Gambar 16.56. Zona Yang Dihosting Menyajikan Kumpulan Rekaman Sumber Daya Utama Yang Baru



Gambar 16.57 Zona Yang Dihosting Menjadi Tidak Dapat Dijangkau Jika Semua Kumpulan Rekaman Sumber Daya Dihapus

AWS CloudFormations untuk kluster Kubernetes memiliki konfigurasi peluncuran dan grup penskalaan yang terkait dengannya. Jika instans pengontrol dimatikan secara langsung,

pada awalnya instans pengontrol dan pekerja akan dimatikan; tetapi karena konfigurasi peluncuran dikaitkan dengan CloudFormation, instans pengontrol dan pekerja lain untuk CloudFormation dimulai. Jika CloudFormation dihapus, kluster dihapus dan tidak diluncurkan kembali. Jika kumpulan rekaman sumber daya utama dan sekunder dibuat tidak tersedia, layanan Kubernetes yang dihosting di oramagsearch.com menjadi tidak tersedia, seperti yang ditunjukkan pada Gambar 16.57.

Ringkasan

Dalam bab ini kami membuat situs web dengan ketersediaan tinggi. Ketersediaan tinggi dapat terwujud dengan membuat beberapa formasi awan dan kemudian membuat layanan AWS Route 53 dengan failover DNS yang dikonfigurasi. Bab ini menyimpulkan buku Pola Desain Manajemen Kubernetes. Seiring dengan pengembangan versi Kubernetes berikutnya, fitur-fitur lain akan ditambahkan. Pada saat buku ini ditulis, Kubernetes 1.3 telah menambahkan federasi lintas-kluster, yang dapat digunakan untuk mengembangkan layanan terfederasi yang menjangkau beberapa kluster, sehingga menyediakan bentuk ketersediaan tinggi lainnya.

DAFTAR PUSTAKA

Anderson, G., & Kaur, N. (2019). *Kubernetes Patterns and Practices*. O'Reilly Media.

Bach, C. (2018). *Deploying and Managing Containers with Docker and Kubernetes*. Packt Publishing.

Batra, A., & Chopra, R. (2020). *Kubernetes for Developers: Develop and Deploy Cloud-Native Applications*. Packt Publishing.

Bell, L., & Patterson, J. (2018). *Optimizing Kubernetes Performance for Cloud-Native Applications*. Wiley.

Betts, M. (2019). *Kubernetes Cookbook: Practical solutions to container management and orchestration*. Packt Publishing.

Burn, S. (2019). *Building Modern Cloud Applications with Docker, Kubernetes, and CoreOS*. Wiley.

Burns, B., & Oppenheimer, D. (2017). *Design Patterns for Container-based Distributed Systems*. USENIX Association.

Chandra, S., & Gupta, P. (2020). *CoreOS and Docker: Automating Deployment in Cloud Environments*. Apress.

Chia, S., & Tan, P. (2018). *Docker and Kubernetes in Action*. Manning Publications.

CoreOS. (2020). *CoreOS Documentation*. https://coreos.com

Darnell, P., & Cooper, B. (2020). *Kubernetes and Docker for Continuous Integration and Deployment*. Wiley.

Dillon, T. (2020). *Designing Containers with Docker and Kubernetes: Using CoreOS for Optimal Deployment*. Apress.

Docker, Inc. (2017). *Docker Best Practices*. https://docs.docker.com/best-practices/

Docker, Inc. (2018). *Docker Documentation*. https://www.docker.com

Docker, Inc. (2020). *Kubernetes: From Docker to Containers Orchestration*. Docker Documentation. https://www.docker.com

Ellsworth, C. (2020). *Hands-On Kubernetes on AWS: Create, Deploy, and Manage Cloud-Native Applications*. Packt Publishing.

Finkelstein, S., & Garrett, D. (2018). *Kubernetes and Docker in Action for System Administrators*. Packt Publishing.

Hansen, D., & Johnson, M. (2020). *Managing Cloud-Native Applications with Kubernetes and Docker*. Wiley.

Healy, R. (2018). *Implementing Docker with Kubernetes in Production Environments*. O'Reilly Media.

Hightower, K. (2020). *Kubernetes in Action*. Manning Publications.

Hightower, K., Burns, B., & Beda, J. (2017). *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications*. O'Reilly Media.

Hutter, G., & Stein, L. (2019). *Scaling Docker on Kubernetes: Managing Large Containerized Systems*. Wiley.

Jenkins, C. (2019). *Kubernetes and Docker for Microservices*. O'Reilly Media.

Jenkins, H. (2019). *Containerized Application Development: Building Scalable Solutions with Docker and Kubernetes*. Wiley.

Jones, J., & Walters, S. (2019). *Managing Containers and Microservices: Building Cloud-Native Applications with Docker and Kubernetes*. Springer.

Kelsey, H. (2016). *CoreOS: An Introduction*. O'Reilly Media.

Kelsey, H., & McKie, P. (2017). *CoreOS: A container-centric operating system for distributed applications*. CoreOS Inc.

Khan, R., & Singh, A. (2018). *Managing Kubernetes in Production: Best Practices*. O'Reilly Media.

King, R., & Patel, M. (2017). *Practical Kubernetes and Docker for Beginners*. O'Reilly Media.

Kubernetes Authors. (2020). *Kubernetes: Up & Running*. O'Reilly Media.

Kumar, R. (2020). *Docker Containers and Kubernetes: A Developer's Guide*. Apress.

Kumar, S., & Mistry, R. (2019). *Container Orchestration with Kubernetes*. Wiley.

Kumar, V., & Rathi, R. (2021). *Cloud-Native Development with Kubernetes and Docker*. Springer.

Lee, A. (2021). *Docker and Kubernetes in Cloud-Native Applications*. Springer.

Lucas, A. (2018). *Managing Microservices with Kubernetes and Docker*. Packt Publishing.

Nguyen, P., & Pham, L. (2019). *Exploring Kubernetes and Docker for Efficient Cloud-Based Solutions*. Springer.

Nicholson, T., & Phillips, C. (2021). *Containerization with Docker and Kubernetes: A Practical Guide*. Apress.

Pahl, C. (2018). *Containerized Applications: Managing Kubernetes and Docker*. Morgan & Claypool Publishers.

Raj, S., & Sharma, A. (2019). *Docker Deep Dive*. Apress.

Richter, B., & Williams, R. (2019). *Building Cloud Applications with Kubernetes and Docker*. Packt Publishing.

Robbins, K. (2021). *Mastering Kubernetes and Docker: Building Scalable and Resilient Applications*. Packt Publishing.

Roberts, T., & Harrison, L. (2021). *Design Patterns for Kubernetes and Cloud-Native Development*. Packt Publishing.

Smith, M., & Roberts, E. (2020). *Continuous Deployment with Kubernetes: Automation and DevOps Best Practices*. Apress.

Tan, H., & Liu, Z. (2020). *Practical Guide to Kubernetes and Docker for System Administrators*. Springer.

Tan, W., & Al-Mashari, M. (2020). *Distributed Systems with Kubernetes: Building Cloud-Native Applications with Docker and Kubernetes*. Springer.

Tirth, S., & Bajaj, A. (2019). *Docker for DevOps: Building Applications with Docker and Kubernetes*. Packt Publishing.

Turner, D., & Harris, K. (2020). *Mastering Docker and Kubernetes for Enterprise Cloud Applications*. Wiley.

Verma, M., & Shah, A. (2020). *Kubernetes for Cloud-Native Microservices*. Apress.

Yadav, S., & Jaiswal, P. (2018). *Cloud-Native Application Development with Docker and Kubernetes*. Packt Publishing.

Zhang, Y., & Lee, J. (2021). *Advanced Kubernetes Design Patterns for Production Workloads*. Wiley.

Desain Manajemen Pola dengan Kubernetes lewat Docker, CoreOS Linux dan Platform Lainnya

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

BIODATA PENULIS

Dr. Joseph Teguh Santoso, M.Kom adalah pemimpin yang visioner dan praktisi industri berpengalaman, yang menjabat sebagai Rektor Universitas Sains dan Teknologi Komputer (Universitas STEKOM), salah satu universitas terkemuka di Jawa Tengah, Indonesia. Dengan pengalaman lebih dari 13 tahun di dunia bisnis dan praktisi industri di China, beliau membawa perspektif global dan inovasi yang signifikan ke dalam dunia akademis.Sebagai seorang entrepreneur, penulis adalah pencipta TopLoker.com, sebuah platform inovatif yang merevolusi cara mencari dan menawarkan pekerjaan. TopLoker.com adalah portal lowongan bursa kerja terbesar di Indonesia, khusus untuk pendidikan SMA/SMK sederajat. TopLoker.com telah mendapatkan penghargaan sebagai juara 1 Startup4industry 2022 oleh Kementerian Perindustrian Republik Indonesia. Kontribusi Dr. Joseph dalam menyediakan akses pekerjaan yang luas bagi lulusan SMA/SMK telah membantu banyak individu menemukan peluang kerja yang sesuai dengan keahlian mereka. Selain itu, Dr. Joseph Teguh Santoso, M.Kom adalah pendiri dari dua organisasi yaitu (1) organisasi guru/pendidik PTIC (Perkumpulan Teacherpreneur Indonesia Cerdas) yang bertujuan untuk meningkatkan kualitas pendidikan dan kesejahteraan guru/pendidik dengan wawasan entrepreneurship, serta (2) organisasi industri PERKIVI (Perkumpulan Komunitas Industri dan Vokasi Indonesia) yang berfokus pada pengembangan link and match antara industri dan dunia pendidikan. Sebagai Rektor, Dr. Joseph Teguh Santoso, M.Kom memiliki kepemimpinan yang berorientasi pada hasil, dan berkomitmen untuk mendorong kemajuan Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Saat ini Universitas STEKOM telah mengalami transformasi positif dalam peningkatan kualitas pendidikan, perluasan fasilitas, serta penguatan kemitraan Perguruan Tinggi Nasional dan Internasional. Beliau memprioritaskan pengembangan sumber daya manusia dan penelitian, serta memastikan bahwa universitas berada di garis depan dalam inovasi dan teknologi untuk mencapai tujuan akhir, yaitu lulusan yang mampu bekerja dan sukses setelah lulus.Dr. Joseph Teguh Santoso, M.Kom sering diundang sebagai pembicara di berbagai konferensi nasional maupun internasional dan telah menerima berbagai penghargaan atas dedikasinya dalam bidang pendidikan, industri, dan kewirausahaan.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8642-74-8 (PDF)
9 786238 642748