YAYASAN PRIMA AGUS TEKNIK

KOMPUTASI TANPA SERVER

(Serverless Computing)

Dr. Agus Wibowo, M.Kom, M.Si, MM.

KOMPUTASI TANPA SERVER

(Serverless Computing)

Dr. Agus Wibowo, M.Kom, M.Si, MM.

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 978-623-8642-62-5 (PDF)
9 786238 642625

Komputasi Tanpa Server (Serverless Computing)

Penulis :
Dr. Agus Wibowo, M.Kom, M.Si, MM.

ISBN : 978-623-8642-62-5

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

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

Puji syukur kita panjatkan ke hadirat Tuhan Yang Maha Esa, atas rahmat dan karunia-Nya, sehingga buku yang berjudul ***"Komputasi Tanpa Server (Serverless Computing)"*** ini dapat diselesaikan dengan baik. Buku ini disusun sebagai upaya untuk memberikan wawasan dan pemahaman yang lebih mendalam mengenai konsep komputasi tanpa server, yang semakin berkembang pesat dalam dunia teknologi informasi. Komputasi tanpa server merupakan salah satu inovasi penting dalam pengembangan aplikasi dan infrastruktur TI, yang memberikan kemudahan dan efisiensi dalam pengelolaan sumber daya komputasi.

Perkembangan teknologi informasi yang semakin pesat memberikan dampak besar terhadap cara kita mengelola infrastruktur komputasi. Di tengah tantangan mengelola server fisik yang memerlukan banyak sumber daya dan pemeliharaan, muncul sebuah solusi yang inovatif, yaitu komputasi tanpa server (serverless computing). Konsep ini memungkinkan pengembang untuk fokus pada penulisan kode tanpa harus khawatir mengenai pengelolaan server yang menjalankan aplikasi tersebut.

Memulai Komputasi Tanpa Server merupakan topik yang sangat relevan di era cloud computing saat ini. Dengan komputasi tanpa server, pengguna tidak lagi perlu membeli, mengonfigurasi, atau mengelola server fisik atau virtual. Sebagai gantinya, penyedia layanan cloud, seperti AWS, Google Cloud, dan Microsoft Azure, menyediakan infrastruktur yang sepenuhnya dikelola dan otomatis. Pengguna hanya perlu membayar berdasarkan penggunaan yang sesungguhnya, bukan untuk kapasitas yang telah dipesan sebelumnya.

Buku ini, kami akan mengupas tuntas mengenai komputasi tanpa server, dimulai dari pengertian dasar, kelebihan dan kekurangan, hingga bagaimana memulai dan mengimplementasikan model ini dalam pengembangan aplikasi. Pembahasan ini bertujuan untuk memberikan pemahaman yang komprehensif mengenai komputasi tanpa server, serta menginspirasi para pengembang dan profesional teknologi informasi untuk memanfaatkan teknologi ini guna meningkatkan efisiensi dan fleksibilitas dalam proyek mereka.

Penulis berharap pembaca dapat memperoleh wawasan yang mendalam dan praktis terkait komputasi tanpa server, serta dapat memanfaatkannya untuk menyelesaikan berbagai masalah teknologi dengan cara yang lebih efisien dan ekonomis. Semoga tulisan ini dapat menjadi referensi yang berguna untuk setiap individu yang tertarik mempelajari dunia komputasi modern.

Semarang, Desember 2024
Penulis

Dr. Agus Wibowo, M.Kom, M.Si, MM.

DAFTAR ISI

BAB 1
MEMAHAMI KOMPUTASI TANPA SERVER

Arsitektur tanpa server mencakup banyak hal, dan sebelum mulai membuat aplikasi tanpa server, penting untuk memahami secara pasti apa itu komputasi tanpa server, cara kerjanya, serta manfaat dan kasus penggunaan komputasi tanpa server. Secara umum, ketika orang berpikir tentang komputasi tanpa server, mereka cenderung berpikir tentang aplikasi dengan back-end yang berjalan pada layanan pihak ketiga, yang juga digambarkan sebagai kode yang berjalan pada kontainer sementara. Menurut pengalaman saya, banyak bisnis dan orang yang baru mengenal komputasi tanpa server akan menganggap aplikasi tanpa server hanya "ada di cloud." Meskipun sebagian besar aplikasi tanpa server dihosting di cloud, merupakan persepsi yang salah bahwa aplikasi ini sepenuhnya tanpa server. Aplikasi tersebut masih berjalan di server yang dikelola oleh pihak lain. Dua contoh paling populer dari hal ini adalah fungsi AWS Lambda dan Azure. Kita akan membahasnya nanti dengan contoh langsung dan juga akan melihat fungsi Cloud Google.

Apa Itu Komputasi Tanpa Server? Komputasi tanpa server adalah teknologi, yang juga dikenal sebagai fungsi sebagai layanan (FaaS), yang memberikan manajemen lengkap kepada penyedia cloud atas kontainer tempat fungsi berjalan sebagaimana diperlukan untuk melayani permintaan. Dengan demikian, arsitektur ini menghilangkan kebutuhan akan sistem yang terus berjalan dan berfungsi sebagai komputasi berbasis peristiwa. Kelayakan untuk membuat aplikasi yang dapat diskalakan dalam arsitektur ini sangat besar.

Bayangkan memiliki kemampuan untuk sekadar menulis kode, mengunggahnya, dan menjalankannya, tanpa harus khawatir tentang infrastruktur, pengaturan, atau pemeliharaan lingkungan yang mendasarinya... Kemungkinannya tidak terbatas, dan kecepatan pengembangan meningkat pesat. Dengan memanfaatkan arsitektur tanpa server, Anda dapat meluncurkan aplikasi yang berfungsi penuh dan dapat diskalakan dalam waktu setengah dari waktu yang dibutuhkan untuk membangunnya dari awal.

1.1 TANPA SERVER SEBAGAI KOMPUTASI BERBASIS PERISTIWA

Komputasi berbasis peristiwa adalah pola arsitektur yang menekankan tindakan sebagai respons terhadap atau berdasarkan penerimaan peristiwa. Pola ini mempromosikan layanan yang digabungkan secara longgar dan memastikan bahwa suatu fungsi hanya dijalankan saat dipicu. Hal ini juga mendorong pengembang untuk memikirkan jenis kejadian dan respons yang dibutuhkan suatu fungsi untuk menangani kejadian tersebut sebelum memprogram fungsi tersebut.

Dalam arsitektur berbasis peristiwa ini, fungsi-fungsi tersebut merupakan konsumen peristiwa karena mereka diharapkan untuk aktif ketika suatu peristiwa terjadi dan bertanggung jawab untuk memprosesnya. Beberapa contoh peristiwa yang memicu fungsi tanpa server meliputi:

- ✓ Permintaan API
- ✓ Penempatan dan pengambilan objek dalam penyimpanan objek

- ✓ Perubahan pada item basis data
- ✓ Peristiwa terjadwal
- ✓ Perintah suara (misalnya, Amazon Alexa)
- ✓ Bot (seperti AWS Lex dan Azure LUIS, keduanya merupakan mesin pemrosesan bahasa alami)

Pada Gambar 1.1 ditunjukan sebuah ilustrasi contoh eksekusi fungsi berbasis peristiwa menggunakan AWS Lambda dan permintaan metode ke API Gateway.

Gambar 1.1. Permintaan dibuat ke API Gateway, yang kemudian memicu fungsi Lambda untuk memberikan respons

Dalam contoh ini, permintaan ke API Gateway dibuat dari aplikasi seluler atau web. API Gateway adalah layanan API Amazon yang memungkinkan Anda membuat permintaan HTTP RESTful dengan cepat dan mudah. API Gateway memiliki fungsi Lambda khusus yang dibuat untuk menangani metode ini yang ditetapkan sebagai titik integrasi. Fungsi Lambda dikonfigurasi untuk menerima peristiwa dari API Gateway. Saat permintaan dibuat, fungsi Amazon Lambda dipicu dan dijalankan.

Contoh kasus penggunaan ini bisa berupa basis data film. Pengguna mengklik nama aktor dalam aplikasi. Klik ini membuat permintaan GET di API Gateway, yang telah ditetapkan sebelumnya untuk memicu fungsi Lambda guna mengambil daftar film yang terkait dengan aktor/aktris tertentu. Fungsi Lambda mengambil daftar ini dari DynamoDB dan mengembalikannya ke aplikasi.

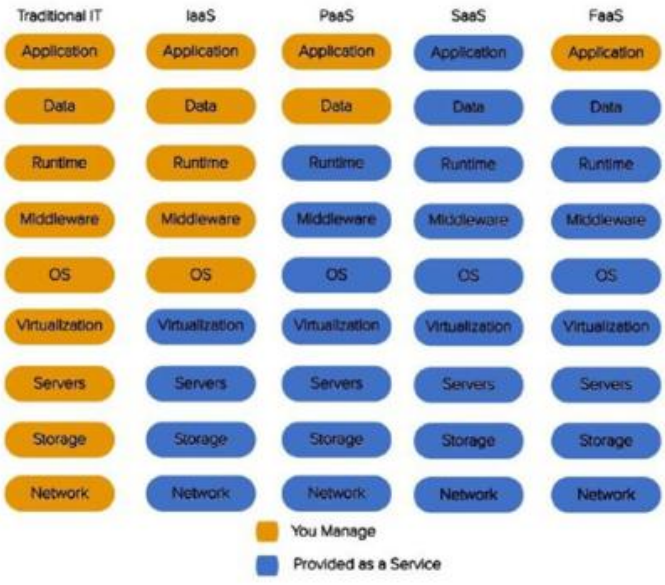
Hal penting lain yang dapat Anda lihat dari contoh ini adalah bahwa fungsi Lambda dibuat untuk menangani satu bagian dari keseluruhan aplikasi. Katakanlah aplikasi tersebut juga memungkinkan pengguna untuk memperbarui basis data dengan informasi baru. Dalam arsitektur tanpa server, Anda ingin membuat fungsi Lambda terpisah untuk menangani hal ini. Tujuan di balik pemisahan ini adalah untuk menjaga fungsi tetap spesifik untuk satu peristiwa. Hal ini membuat fungsi tersebut tetap ringan, dapat diskalakan, dan mudah direfaktor. Kami akan membahas hal ini secara lebih rinci di bagian selanjutnya.

1.2 FUNGSI SEBAGAI LAYANAN (FAAS)

Seperti yang disebutkan sebelumnya, komputasi tanpa server adalah model komputasi awan di mana kode dijalankan sebagai layanan tanpa perlu pengguna untuk memelihara atau membuat infrastruktur yang mendasarinya. Ini tidak berarti bahwa arsitektur tanpa server tidak memerlukan server, tetapi pihak ketiga mengelola server ini sehingga server tersebut dipisahkan dari pengguna. Cara yang baik untuk menganggapnya sebagai "Fungsi sebagai Layanan" (FaaS). Kode berbasis peristiwa khusus dibuat oleh pengembang dan dijalankan pada kontainer sementara tanpa status yang dibuat dan dipelihara oleh pihak ketiga.

FaaS sering kali merupakan istilah untuk menggambarkan teknologi tanpa server, jadi ada baiknya untuk mempelajari konsep tersebut secara lebih mendetail. Anda mungkin juga pernah mendengar tentang IaaS (infrastruktur sebagai layanan), PaaS (platform sebagai layanan), dan SaaS (perangkat lunak sebagai layanan) sebagai model layanan komputasi awan. IaaS menyediakan infrastruktur komputasi, mesin fisik atau virtual, dan sumber daya lain seperti pustaka citra disk mesin virtual, penyimpanan berbasis blok dan file, firewall, penyeimbang beban, alamat IP, dan jaringan area lokal virtual. Contohnya adalah instans Amazon Elastic Compute Cloud (EC2).

PaaS menyediakan platform komputasi yang biasanya mencakup sistem operasi, lingkungan eksekusi bahasa pemrograman, basis data, dan server web. Beberapa contohnya termasuk AWS Elastic Beanstalk, Azure Web Apps, dan Heroku. SaaS menyediakan akses ke perangkat lunak aplikasi. Instalasi dan penyiapan dihapus dari proses dan Anda tinggal menggunakan aplikasi. Beberapa contohnya termasuk Salesforce dan Workday. Secara unik, FaaS mengharuskan menjalankan kode back-end tanpa tugas mengembangkan dan menerapkan aplikasi server dan sistem server Anda sendiri. Semua ini ditangani oleh penyedia pihak ketiga. Kita akan membahasnya nanti di bagian ini. Gambar 1.2 menunjukan ilustrasi perbedaan utama antara tren arsitektur yang telah kita bahas.



Gambar 1.2 Apa yang dikelola pengembang dibandingkan dengan apa yang dikelola penyedia dalam sistem arsitektur yang berbeda

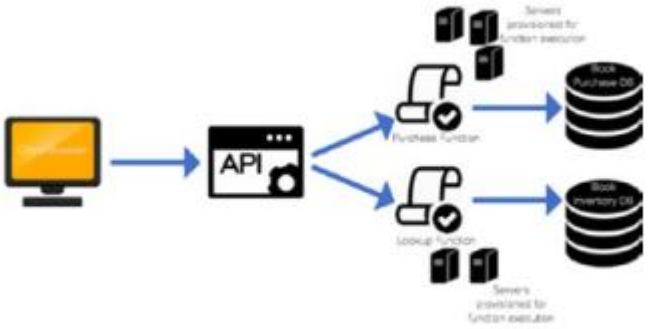
1.3 BAGAIMANA KOMPUTASI TANPA SERVER BEKERJA?

Kita tahu bahwa komputasi tanpa server adalah FaaS yang digerakkan oleh peristiwa, tetapi bagaimana cara kerjanya dari sudut pandang penyedia cloud? Bagaimana server disediakan, diskalakan secara otomatis, dan ditempatkan agar FaaS berfungsi? Salah satu kesalahpahaman adalah menganggap bahwa komputasi tanpa server tidak memerlukan server. Ini sebenarnya tidak benar. Fungsi tanpa server tetap berjalan di server; perbedaannya adalah pihak ketiga yang mengelolanya, bukan pengembang. Untuk menjelaskan hal ini, kita akan menggunakan contoh sistem tiga tingkat tradisional dengan logika sisi server dan menunjukkan perbedaannya menggunakan arsitektur tanpa server.

Misalnya kita memiliki situs web tempat kita dapat mencari dan membeli buku teks. Dalam arsitektur tradisional, Anda mungkin memiliki klien, server dengan beban seimbang, dan basis data untuk buku teks. Dalam Gambar 1.3 mengilustrasikan arsitektur tradisional ini untuk toko buku teks daring.

Gambar 1.3 Konfigurasi arsitektur tradisional tempat server disediakan dan dikelola oleh pengembang

Dalam arsitektur tanpa server, beberapa hal dapat berubah, termasuk server dan basis data. Contoh perubahan ini adalah membuat API yang disediakan cloud dan memetakan permintaan metode tertentu ke berbagai fungsi. Alih-alih memiliki satu server, aplikasi kita kini memiliki fungsi untuk setiap bagian fungsionalitas dan server yang disediakan cloud yang dibuat berdasarkan permintaan. Kita dapat memiliki fungsi untuk mencari buku, dan juga fungsi untuk membeli buku. Kita juga dapat memilih untuk membagi basis data kita menjadi dua basis data terpisah yang sesuai dengan kedua fungsi tersebut. Gambar 1.4 menunjukan sebuah ilustrasi arsitektur tanpa server untuk toko buku teks daring.



Gambar 1.4 Konfigurasi arsitektur tanpa server tempat server diaktifkan dan dinonaktifkan berdasarkan permintaan

Ada beberapa perbedaan antara dua diagram arsitektur untuk toko buku daring. Salah satunya adalah bahwa dalam contoh lokal, Anda memiliki satu server yang perlu diseimbangkan bebannya dan diskalakan secara otomatis oleh pengembang. Dalam solusi cloud, aplikasi dijalankan dalam kontainer komputasi stateless yang diaktifkan dan dinonaktifkan oleh fungsi yang dipicu. Perbedaan lainnya adalah pemisahan layanan dalam contoh tanpa server.

1.4 APA BEDANYA?

Apa bedanya komputasi tanpa server dengan mengaktifkan server dan membangun infrastruktur dari awal? Kita tahu bahwa perbedaan utamanya adalah mengandalkan vendor pihak ketiga untuk memelihara server Anda, tetapi bagaimana hal itu membuat perbedaan dalam keseluruhan aplikasi dan proses pengembangan Anda? Dua perbedaan utama yang mungkin Anda lihat adalah dalam pengembangan aplikasi dan proses independen yang digunakan untuk membuatnya.

Pengembangan

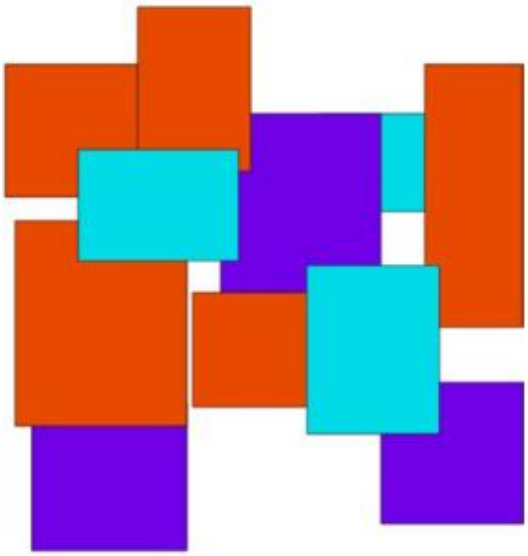
Proses pengembangan untuk aplikasi tanpa server sedikit berbeda dari cara seseorang mengembangkan sistem di tempat. Aspek lingkungan pengembangan termasuk IDE, kontrol sumber, pembuatan versi, dan opsi penyebaran semuanya dapat ditetapkan oleh pengembang baik di tempat atau dengan penyedia cloud. Metode pengembangan berkelanjutan yang disukai meliputi penulisan fungsi tanpa server menggunakan IDE, seperti Visual Studio, Eclipse, dan IntelliJ, dan menyebarkannya dalam potongan-potongan kecil ke penyedia cloud menggunakan antarmuka baris perintah penyedia cloud. Jika fungsinya cukup kecil, fungsinya dapat dikembangkan dalam portal penyedia cloud yang sebenarnya. Kami akan membahas proses pengunggahan di bab-bab selanjutnya untuk memberi Anda gambaran tentang perbedaan antara lingkungan pengembangan serta perbedaan penyedia. Namun, sebagian besar memiliki batasan ukuran fungsi sebelum mengharuskan pengunggahan zip proyek.

Antarmuka baris perintah (CLI) adalah alat pengembangan yang hebat karena membuat fungsi tanpa server dan layanan yang diperlukannya dapat disebarkan dengan mudah dan memungkinkan Anda untuk terus menggunakan alat pengembangan yang ingin Anda gunakan untuk menulis dan menghasilkan kode Anda. Alat Serverless Framework adalah opsi pengembangan lain yang dapat diinstal menggunakan NPM, seperti yang akan Anda lihat lebih detail nanti di bab ini.

Proses Independen

Cara lain untuk menganggap fungsi serverless adalah sebagai layanan mikro serverless. Setiap fungsi memiliki tujuannya sendiri dan menyelesaikan proses secara independen dari fungsi lainnya. Komputasi serverless bersifat stateless dan berbasis peristiwa, jadi beginilah cara fungsi tersebut dikembangkan. Misalnya, dalam arsitektur tradisional dengan operasi API CRUD dasar (GET, POST, PUT, DELETE), Anda mungkin memiliki model berbasis objek dengan metode ini yang ditetapkan pada setiap objek. Gagasan untuk mempertahankan modularitas masih berlaku di level serverless. Setiap fungsi dapat mewakili satu metode API dan

melakukan satu proses. Serverless Framework membantu dalam hal ini, karena ia memberlakukan fungsi yang lebih kecil yang akan membantu memfokuskan kode Anda dan membuatnya tetap modular.



Gambar 1.5. Gambar ini menunjukkan ketergantungan setiap aspek sistem yang berbeda secara fungsional terhadap aspek lainnya

Fungsi harus ringan, dapat diskalakan, dan harus memiliki satu tujuan. Untuk membantu menjelaskan mengapa gagasan proses independen lebih disukai, kita akan melihat berbagai gaya arsitektur dan perubahan yang telah dilakukan terhadapnya dari waktu ke waktu. Gambar 1.5 mengilustrasikan desain arsitektur monolitik.

Aplikasi monolitik dibangun sebagai satu unit yang saling terkait dengan aplikasi sisi server yang menangani semua permintaan dan logika yang terkait dengan aplikasi tersebut. Ada beberapa masalah dengan model arsitektur ini. Masalah selama periode pengembangan mungkin adalah tidak ada pengembang yang memiliki pemahaman lengkap tentang sistem tersebut, karena semua fungsionalitas dikemas dalam satu unit. Beberapa masalah lainnya termasuk ketidakmampuan untuk diskalakan, penggunaan ulang yang terbatas, dan kesulitan dalam penerapan berulang.

Pendekatan layanan mikro melepaskan diri dari pola arsitektur monolitik dengan memisahkan layanan menjadi komponen independen yang dibuat, diterapkan, dan dipelihara secara terpisah satu sama lain. Gambar 1.6 mengilustrasikan arsitektur layanan mikro.

Gambar 1.6 Gambar ini menunjukkan layanan independen yang membentuk arsitektur layanan mikro

Banyak masalah yang kami lihat pada pendekatan monolitik diatasi melalui solusi ini. Layanan dibangun sebagai komponen individual dengan satu tujuan. Hal ini memungkinkan aplikasi untuk dikonsumsi dan digunakan oleh layanan lain dengan lebih mudah dan efisien. Hal ini juga memungkinkan skalabilitas yang lebih baik karena Anda dapat memilih layanan mana yang akan ditingkatkan atau diturunkan skalanya tanpa harus menskalakan seluruh sistem. Selain itu, menyebarkan fungsionalitas di berbagai layanan mengurangi kemungkinan terjadinya satu titik kegagalan dalam kode Anda. Layanan mikro ini juga lebih cepat dibangun dan disebarkan karena Anda dapat melakukannya secara independen tanpa membangun seluruh aplikasi. Hal ini membuat waktu pengembangan lebih cepat dan lebih efisien, dan juga memungkinkan pengembangan dan pengujian yang lebih cepat dan mudah.

1.5 MANFAAT DAN KASUS PENGGUNAAN

Satu hal yang menjadi kendala banyak pengembang dan bisnis besar tentang arsitektur tanpa server adalah memberikan penyedia cloud kendali penuh atas platform layanan Anda. Namun, ada banyak alasan dan kasus penggunaan yang menjadikan ini keputusan yang baik yang dapat menguntungkan hasil keseluruhan dari suatu solusi. Beberapa manfaatnya antara lain:

- Pengembangan dan penerapan yang cepat
- Kemudahan penggunaan
- Biaya yang lebih rendah
- Skalabilitas yang ditingkatkan
- Tidak memerlukan pemeliharaan infrastruktur

Pengembangan dan Penerapan yang Cepat

Karena semua infrastruktur, pemeliharaan, dan penskalaan otomatis ditangani oleh penyedia cloud, waktu pengembangan menjadi jauh lebih cepat dan penerapan menjadi lebih mudah dari sebelumnya. Pengembang hanya bertanggung jawab atas aplikasi itu sendiri, sehingga tidak perlu lagi merencanakan waktu yang akan dihabiskan untuk penyiapan server. AWS, Azure, dan Google juga menyediakan templat fungsi yang dapat digunakan untuk membuat fungsi yang dapat dieksekusi dengan segera.

Penerapan juga menjadi jauh lebih sederhana, sehingga prosesnya menjadi lebih cepat. Penyedia cloud ini memiliki versi dan aliasing bawaan bagi pengembang untuk digunakan dalam bekerja dan menerapkan di lingkungan yang berbeda.

Kemudahan Penggunaan

Salah satu manfaat terbesar dalam menerapkan solusi tanpa server adalah kemudahan penggunaannya. Hanya diperlukan sedikit waktu persiapan untuk memulai pemrograman aplikasi tanpa server. Sebagian besar kesederhanaan ini berkat layanan yang disediakan oleh penyedia cloud yang memudahkan penerapan solusi lengkap. Pemicu yang diperlukan untuk menjalankan fungsi Anda mudah dibuat dan disediakan dalam lingkungan cloud, dan hanya memerlukan sedikit pemeliharaan. Melihat contoh berbasis peristiwa dari sebelumnya, gateway API sepenuhnya dikelola oleh AWS tetapi mudah dibuat dan ditetapkan sebagai pemicu untuk fungsi Lambda dalam waktu singkat. Pengujian, pencatatan, dan pembuatan versi adalah semua kemungkinan dengan teknologi tanpa server dan semuanya dikelola oleh penyedia cloud. Fitur dan layanan bawaan ini memungkinkan pengembang untuk fokus pada kode dan hasil aplikasi.

Biaya Lebih Rendah

Untuk solusi tanpa server, Anda dikenai biaya per eksekusi, bukan keberadaan seluruh aplikasi. Ini berarti Anda membayar untuk apa yang Anda gunakan. Selain itu, karena server aplikasi dikelola dan diskalakan otomatis oleh penyedia cloud, harganya juga lebih murah daripada yang akan Anda bayarkan di rumah. Dalam tabel 1.1 memberi Anda rincian biaya solusi tanpa server di berbagai penyedia.

Tabel 1.1 Harga untuk Eksekusi Fungsi oleh Penyedia Cloud pada Publikasi

AWS Lambda	Fungsi Azure	Fungsi Google Cloud
Gratis satu juta permintaan pertama dalam sebulan	Gratis satu juta permintaan pertama per bulan	Gratis 2 juta permintaan pertama per bulan
Rp.3.209 per juta permintaan setelahnya	$ 0,20 per juta permintaan setelahnya	$ 0,40 per juta permintaan setelahnya
Rp.26.730 untuk setiap GB-detik yang digunakan	$ 0,000016 untuk setiap GB-detik yang digunakan	$ 0,000025 untuk setiap GB-detik yang digunakan

Skalabilitas yang Ditingkatkan

Dengan solusi tanpa server, skalabilitas secara otomatis sudah terpasang karena server dikelola oleh penyedia pihak ketiga. Ini berarti waktu, uang, dan analisis yang biasanya

digunakan untuk menyiapkan penskalaan dan penyeimbangan otomatis akan terbuang sia-sia.

Gambar 1.7 Gambar ini, dari blog.fugue.co, menunjukkan ketersediaan fungsi serverless yang luas di seluruh penyedia cloud

Selain skalabilitas, ketersediaan juga meningkat karena penyedia cloud mempertahankan kapasitas komputasi di seluruh zona dan wilayah ketersediaan. Hal ini membuat aplikasi tanpa server Anda aman dan tersedia karena melindungi kode dari kegagalan regional. Gambar 1.7 mengilustrasikan wilayah dan zona untuk penyedia cloud.

Penyedia cloud menangani administrasi yang dibutuhkan untuk sumber daya komputasi. Ini termasuk server, sistem operasi, patching, logging, pemantauan, serta penskalaan dan penyediaan otomatis.

Studi Kasus Netflix Dengan Aws

Netflix, pemimpin dalam layanan streaming video dengan teknologi baru, menggunakan arsitektur tanpa server untuk mengotomatiskan proses pengodean file media, validasi penyelesaian pencadangan dan penerapan instans dalam skala besar, serta pemantauan sumber daya AWS yang digunakan oleh organisasi.

Untuk menerapkan ini, Netflix membuat peristiwa pemicu pada fungsi Lambda mereka yang menyinkronkan tindakan dalam produksi ke situs pemulihan bencana. Mereka juga melakukan peningkatan dalam otomatisasi dengan dasbor dan pemantauan produksi mereka. Netflix mencapai ini dengan menggunakan peristiwa pemicu untuk membuktikan bahwa konfigurasi tersebut benar-benar berlaku.

1.6 BATASAN KOMPUTASI TANPA SERVER

Seperti kebanyakan hal, arsitektur tanpa server memiliki batasannya sendiri. Sama pentingnya dengan mengenali kapan harus menggunakan komputasi tanpa server dan cara mengimplementasikannya, sama pentingnya untuk menyadari kekurangan dalam mengimplementasikan solusi tanpa server dan mampu mengatasi masalah ini sebelumnya. Beberapa batasan ini meliputi:

- Anda ingin mengendalikan infrastruktur Anda.
- Anda merancang aplikasi server yang berjalan lama.
- Anda ingin menghindari ketergantungan pada vendor.
- Anda khawatir tentang efek "cold start".
- Anda ingin mengimplementasikan infrastruktur bersama.
- Ada sejumlah alat siap pakai yang terbatas untuk diuji dan diterapkan secara lokal.

Kita akan melihat opsi untuk mengatasi semua masalah ini segera. Secara unik, FaaS memerlukan menjalankan kode back-end tanpa tugas mengembangkan dan menerapkan aplikasi server dan sistem server Anda sendiri. Semua ini ditangani oleh penyedia pihak ketiga.

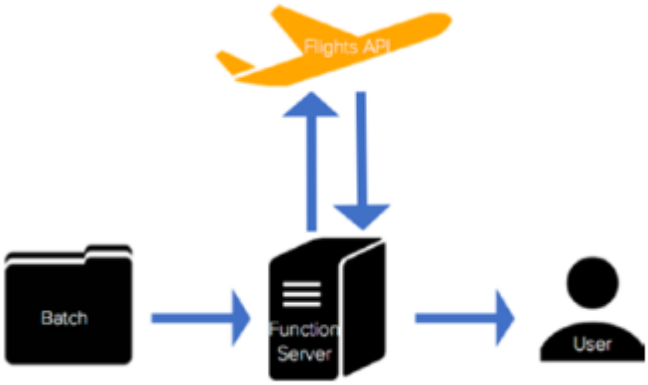
Kontrol Infrastruktur

Keterbatasan potensial untuk menggunakan arsitektur tanpa server adalah kebutuhan untuk mengontrol infrastruktur. Meskipun penyedia cloud tetap memiliki kontrol dan penyediaan infrastruktur dan OS, ini tidak berarti pengembang kehilangan kemampuan untuk menentukan bagian-bagian infrastruktur. Di dalam portal fungsi masing-masing penyedia cloud, pengguna memiliki kemampuan untuk memilih waktu proses, memori, izin, dan batas waktu. Dengan cara ini, pengembang tetap memiliki kontrol tanpa pemeliharaan.

Aplikasi Server yang Berjalan Lama

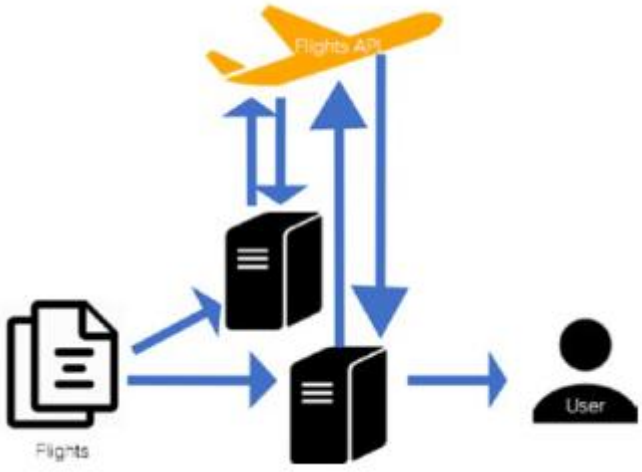
Salah satu manfaat arsitektur tanpa server adalah arsitektur tersebut dibuat agar menjadi fungsi yang cepat, dapat diskalakan, dan digerakkan oleh peristiwa. Oleh karena itu, operasi batch yang berjalan lama tidak cocok untuk arsitektur ini. Sebagian besar penyedia cloud memiliki batas waktu lima menit, jadi proses apa pun yang membutuhkan waktu lebih lama dari waktu yang dialokasikan ini akan dihentikan. Idenya adalah untuk beralih dari pemrosesan batch dan beralih ke fungsionalitas yang cepat, responsif, dan real-time.

Jika ada kebutuhan untuk beralih dari pemrosesan batch dan keinginan untuk melakukannya, arsitektur tanpa server adalah cara yang baik untuk mencapainya. Mari kita lihat sebuah contoh. Misalnya, kita bekerja di perusahaan asuransi perjalanan dan kita memiliki sistem yang mengirimkan sejumlah penerbangan untuk hari itu ke aplikasi yang memantaunya dan memberi tahu perusahaan tersebut jika ada penerbangan yang ditunda atau dibatalkan. Gambar 1.8 menunjukan sebuah ilustrasi aplikasi ini.



Gambar 1.8 Konfigurasi aplikasi pemantauan penerbangan yang mengandalkan pekerjaan batch

Untuk memodifikasinya guna memproses dan memantau penerbangan secara real time, kita dapat menerapkan solusi tanpa server. Gambar 1.9 mengilustrasikan arsitektur solusi ini dan bagaimana kita dapat mengubah aplikasi server yang berjalan lama ini menjadi aplikasi real-time yang digerakkan oleh peristiwa.



Gambar 1.9 Konfigurasi aplikasi pemantauan penerbangan yang menggunakan fungsi dan pemicu API untuk memantau dan memperbarui penerbangan

Solusi waktu nyata ini lebih disukai karena beberapa alasan. Pertama, bayangkan Anda menerima penerbangan yang ingin Anda pantau setelah pekerjaan batch hari itu telah dijalankan. Penerbangan ini akan diabaikan dalam sistem pemantauan. Alasan lain Anda mungkin ingin membuat perubahan ini adalah agar dapat memproses penerbangan ini lebih cepat.

Pada jam berapa pun di siang hari saat proses batch dapat terjadi, penerbangan dapat lepas landas, oleh karena itu juga diabaikan dari sistem pemantauan. Meskipun dalam kasus

ini masuk akal untuk beralih dari batch ke tanpa server, ada situasi lain di mana pemrosesan batch lebih disukai.

Vendor Lock-In

Salah satu ketakutan terbesar saat beralih ke teknologi tanpa server adalah vendor lock-in. Ini adalah ketakutan umum saat beralih ke teknologi cloud. Perusahaan khawatir bahwa dengan berkomitmen menggunakan Lambda, mereka berkomitmen pada AWS dan tidak akan dapat beralih ke penyedia cloud lain atau tidak akan mampu melakukan transisi lain ke penyedia cloud.

Meskipun ini dapat dimengerti, ada banyak cara untuk mengembangkan aplikasi agar vendor dapat beralih menggunakan fungsi dengan lebih mudah. Strategi yang populer dan disukai adalah menarik logika penyedia cloud dari file pengendali sehingga dapat dengan mudah dialihkan ke penyedia lain. Daftar 1-1 menggambarkan contoh yang buruk dari pengabstrakan logika penyedia cloud, yang disediakan oleh serverlessframework.com.

Daftar 1.1. File pengendali untuk fungsi yang mencakup semua logika basis data yang terikat ke penyedia FaaS (AWS dalam kasus ini)

```
const db = require('db').connect();
const mailer = require('mailer');

module.exports.saveUser = (event, context, callback) => {
  const user = {
   email: event.email,
created_at: Date.now()
}
db.saveUser(user, function (err) {
 if (err) {
 callback(err);
} else {
  mailer.sendWelcomeEmail(event.email);
  callback();
  }
 });
};
```

Kode dalam Daftar 1.2 mengilustrasikan contoh yang lebih baik tentang pengabstraksian logika penyedia cloud, yang juga disediakan oleh serverlessframework.com.
Daftar 1.2. File pengendali yang diabstraksikan dari logika penyedia FaaS dengan membuat kelas Pengguna yang terpisah

```
class Users {
 constructor(db, mailer) {
  this.db = db;
```

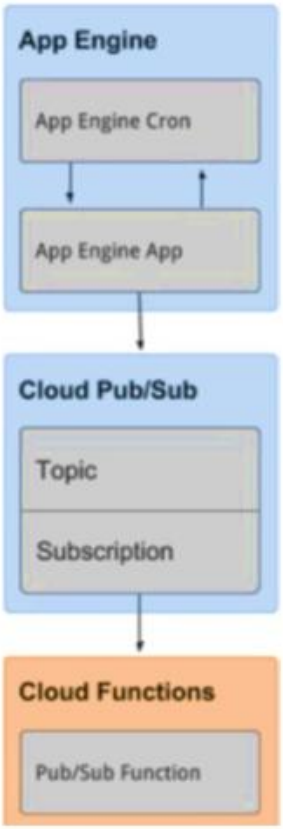
```
  this.mailer = mailer;
 }
 save(email, callback) {
  const user = {
   email: email,
   created_at: Date.now()
 }
this.db.saveUser(user, function (err) {
 if (err) {
 callback(err);
 } else {
 this.mailer.sendWelcomeEmail(email);
 callback();
 }
 });
 }
}
module.exports = Users;

const db = require('db').connect();
const mailer = require('mailer');
const Users = require('users');

let users = new Users(db, mailer);

module.exports.saveUser = (event, context, callback) => {
 users.save(event.email, callback);
};
```

Metode kedua lebih baik untuk menghindari ketergantungan pada vendor dan untuk pengujian. Menghapus logika penyedia cloud dari event handler membuat aplikasi lebih fleksibel dan dapat diterapkan ke banyak penyedia. Ini juga memudahkan pengujian dengan memungkinkan Anda menulis pengujian unit tradisional untuk memastikannya berfungsi dengan baik. Anda juga dapat menulis pengujian integrasi untuk memverifikasi bahwa integrasi dengan layanan lain berfungsi dengan baik.

“Cold Start”

Kekhawatiran tentang “cold start” adalah bahwa suatu fungsi membutuhkan waktu sedikit lebih lama untuk merespons suatu peristiwa setelah periode tidak aktif. Ini memang cenderung terjadi, tetapi ada cara untuk mengatasi cold start jika Anda memerlukan fungsi yang segera responsif. Jika Anda tahu fungsi Anda hanya akan dipicu secara berkala, pendekatan untuk mengatasi cold start adalah dengan membuat penjadwal yang memanggil fungsi Anda untuk membangunkannya sesekali.

Di AWS, opsi ini adalah CloudWatch. Anda dapat mengatur peristiwa terjadwal agar terjadi sesekali sehingga fungsi Anda tidak mengalami cold start. Azure dan Google juga memiliki kemampuan ini dengan pemicu pengatur waktu. Google tidak memiliki penjadwal langsung untuk fungsi Cloud, tetapi memungkinkan untuk membuatnya menggunakan App Engine Cron, yang memicu topik dengan langganan fungsi. Gambar 1.10 mengilustrasikan solusi Google untuk menjadwalkan peristiwa pemicu.



Gambar 1.10. Diagram dari Google Cloud ini menunjukkan konfigurasi peristiwa pemicu terjadwal menggunakan fungsi Cron, Topic, dan Cloud App Engine

Hal penting yang perlu diperhatikan tentang masalah cold start adalah bahwa hal itu sebenarnya dipengaruhi oleh runtime dan ukuran memori. C# dan Java memiliki latensi cold start yang jauh lebih besar daripada runtime seperti Python dan Node.js. Selain itu, ukuran memori meningkatkan cold start secara linear (semakin banyak memori yang Anda gunakan, semakin lama waktu yang dibutuhkan untuk memulai). Hal ini penting untuk diingat saat Anda menyiapkan dan mengonfigurasi fungsi tanpa server.

Infrastruktur Bersama

Karena manfaat arsitektur tanpa server bergantung pada kemampuan penyedia untuk menghosting dan memelihara infrastruktur dan perangkat keras, sebagian biaya aplikasi tanpa server juga ada dalam layanan ini. Ini juga dapat menjadi perhatian dari perspektif bisnis, karena fungsi tanpa server dapat berjalan berdampingan tanpa memandang kepemilikan bisnis (Netflix dapat dihosting di server yang sama dengan layanan streaming Disney

mendatang). Meskipun ini tidak memengaruhi kode, ini berarti ketersediaan dan skalabilitas yang sama akan disediakan di antara para pesaing.

Jumlah Alat Pengujian yang Terbatas

Salah satu keterbatasan pertumbuhan arsitektur tanpa server adalah jumlah alat pengujian dan penerapan yang terbatas. Ini diantisipasi akan berubah seiring dengan pertumbuhan bidang tanpa server, dan sudah ada beberapa alat yang akan datang yang telah membantu penerapan. Saya mengantisipasi bahwa penyedia cloud akan mulai menawarkan cara untuk menguji aplikasi tanpa server secara lokal sebagai layanan. Azure telah melakukan beberapa langkah ke arah ini, dan AWS juga telah memperluasnya. NPM telah merilis beberapa alat pengujian sehingga Anda dapat menguji secara lokal tanpa menerapkannya ke penyedia Anda. Beberapa alat ini termasuk node-lambda dan aws-lambda-local. Salah satu alat penyebaran favorit saya saat ini adalah alat penyebaran Serverless Framework. Alat ini kompatibel dengan AWS, Azure, Google, dan IBM.

Saya menyukainya karena alat ini membuat konfigurasi dan penyebaran fungsi Anda ke penyedia yang Anda berikan menjadi sangat mudah, yang juga berkontribusi pada waktu pengembangan yang lebih cepat. Serverless Framework, jangan disamakan dengan arsitektur serverless, adalah kerangka kerja aplikasi sumber terbuka yang memungkinkan Anda membangun arsitektur serverless dengan mudah. Kerangka kerja ini memungkinkan Anda untuk menyebarkan fungsi yang digerakkan oleh peristiwa, pembayaran per eksekusi, dan penskalaan otomatis ke AWS, Azure, Google Cloud, dan OpenWhisk milik IBM. Manfaat menggunakan Serverless Framework untuk menyebarkan pekerjaan Anda meliputi:

- ✓ Penyebaran cepat: Anda dapat menyediakan dan menyebarkan dengan cepat menggunakan beberapa baris kode di terminal.
- ✓ Skalabilitas: Anda dapat bereaksi terhadap miliaran peristiwa di Serverless Framework; dan Anda dapat menyebarkan layanan cloud lain yang mungkin berinteraksi dengan fungsi Anda (ini termasuk peristiwa pemicu yang diperlukan untuk mengeksekusi fungsi Anda).
- ✓ Kesederhanaan: Arsitektur serverless yang mudah dikelola termuat dalam satu file yml yang disediakan oleh framework secara langsung.
- ✓ Kolaborasi: Kode dan proyek dapat dikelola lintas tim.

Dalam Tabel 1.2 mengilustrasikan perbedaan antara penerapan dengan Serverless Framework dan penerapan manual.

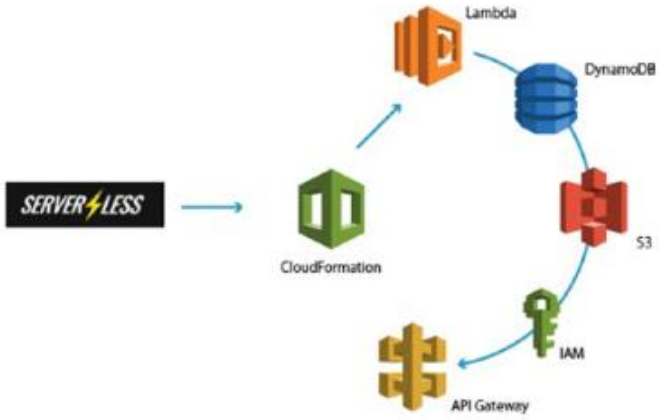
Tabel 1.2 Membandingkan konfigurasi peristiwa pemicu terjadwal menggunakan fungsi App engine Cron, Topic, dan Cloud dalam penerapan serverless dan manual

Fungsi	Penerapan tanpa server vs manual	
	Kerangka tanpa server	**Penerapan manual**
Cron	Keamanan di luar kotak	Keamanan dibangun secara mandiri
Topik	Pembuatan layanan otomatis	Layanan dibangun secara mandiri

Cloud	Sumber daya reproduksi dibuat skrip penerapan yang telah diformat sebelumnya	Sumber daya reproduksi harus dibuat secara terpisah tulis skrip khusus untuk menerapkan fungsi

Gambar tersebut memberikan gambaran umum yang baik tentang Serverless Framework dan manfaat menggunakannya. Kita akan mendapatkan beberapa pengalaman langsung dengan Serverless nanti, jadi mari kita lihat cara kerjanya. Pertama, Serverless diinstal menggunakan NPM (node package manager) di direktori kerja Anda. NPM membongkar Serverless dan membuat file serverless.yml di folder proyek.

File ini adalah tempat Anda menentukan berbagai layanan (fungsi), pemicunya, konfigurasi, dan keamanannya. Untuk setiap penyedia cloud, saat proyek disebarkan, file terkompresi dari kode fungsi diunggah ke penyimpanan objek. Setiap sumber daya tambahan yang ditentukan ditambahkan ke templat khusus untuk penyedia (CloudFormation untuk AWS, Google Deployment Manager untuk Google, dan Azure Resource Manager untuk Azure). Setiap penyebaran menerbitkan versi baru untuk setiap fungsi di layanan Anda. Gambar 1.11 mengilustrasikan penyebaran tanpa server untuk fungsi AWS Lambda.



Gambar 1.11. Gambar ini menunjukkan cara Serverless menerapkan aplikasi menggunakan CloudFormation, yang kemudian membangun layanan lainnya dalam proyek yang dikonfigurasi

Platform Serverless adalah salah satu alat pengembangan dan pengujian terkemuka untuk arsitektur serverless. Seiring kemajuan teknologi serverless, lebih banyak alat akan muncul baik di dalam antarmuka penyedia cloud maupun di luarnya.

1.7 KESIMPULAN

Dalam bab ini, Anda mempelajari tentang aplikasi dan arsitektur serverless, manfaat dan kasus penggunaan, serta batasan penggunaan pendekatan serverless. Penting untuk memahami arsitektur serverless dan apa saja yang dicakupnya sebelum merancang aplikasi yang mengandalkannya. Komputasi serverless adalah teknologi berbasis peristiwa, fungsi sebagai layanan (FaaS) yang memanfaatkan teknologi dan server pihak ketiga untuk

menghilangkan masalah dalam membangun dan memelihara infrastruktur untuk membuat aplikasi. Bab berikutnya akan membahas perbedaan antara tiga penyedia yang sedang kita bahas (AWS, Azure, Google), opsi pengembangan, dan cara menyiapkan lingkungan Anda.

BAB 2
MEMULAI PENGEMBANGAN APLIKASI TANPA SERVER

Untuk mulai mengembangkan aplikasi tanpa server, kita perlu melihat penawaran dan lingkungan tanpa server untuk AWS, Azure, dan Google, pilihan kita untuk platform dan perangkat pengembangan, dan cara menyiapkan lingkungan kita untuk mereka. Seperti yang dibahas dalam bab sebelumnya, tanpa server tidak berarti tidak ada server yang terlibat, tetapi server dihosting oleh penyedia pihak ketiga yang berbeda. Beberapa penyedia yang paling umum untuk opsi tanpa server ini termasuk AWS, Azure, dan Google.

Kita akan memeriksa bagaimana opsi tanpa server berbeda dari satu penyedia ke penyedia lainnya. Kita juga akan membahas proses penyiapan lingkungan menggunakan Visual Studio Code, Node.js, dan Postman. Ada banyak alat pengembangan, lingkungan, dan SDK yang berbeda yang dapat digunakan untuk mengembangkan aplikasi tanpa server. Kita akan membahas beberapa opsi lain dalam bab ini dan kemudian membahas mengapa kita akan menggunakan yang khusus untuk tutorial ini.

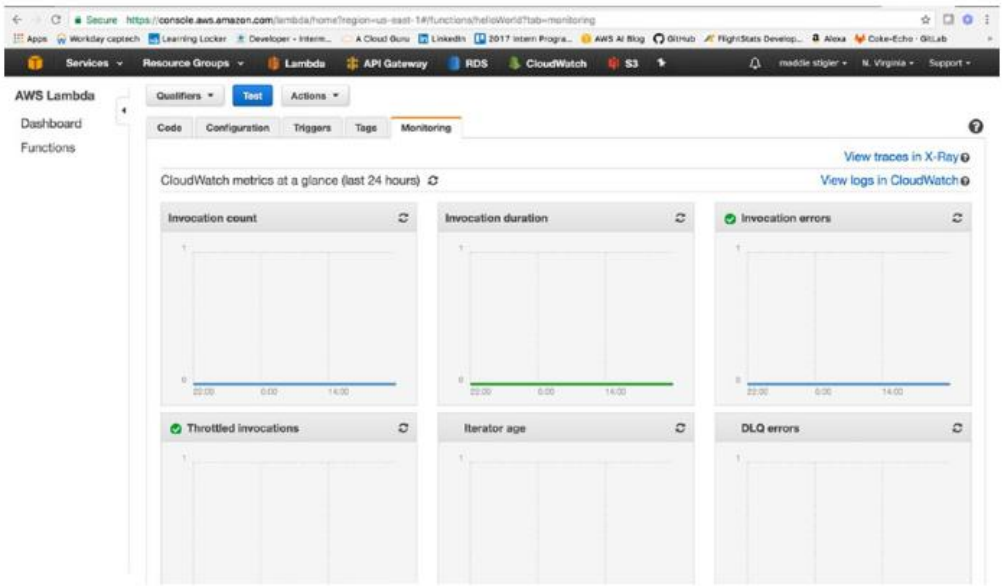
2.1 APA YANG DITAWARKAN SETIAP PENYEDIA

Amazon Web Services, Microsoft Azure, dan Google Cloud Platform adalah tiga penyedia pihak ketiga yang paling umum untuk teknologi tanpa server. Dalam bab ini, kita akan membahas opsi tanpa server untuk masing-masing dan perbedaannya satu sama lain. Ini akan memberi Anda pemahaman yang lebih baik tentang setiap penawaran untuk membantu Anda memilih di antara penyedia cloud saat Anda menulis aplikasi tanpa server Anda sendiri.

AWS Lambda

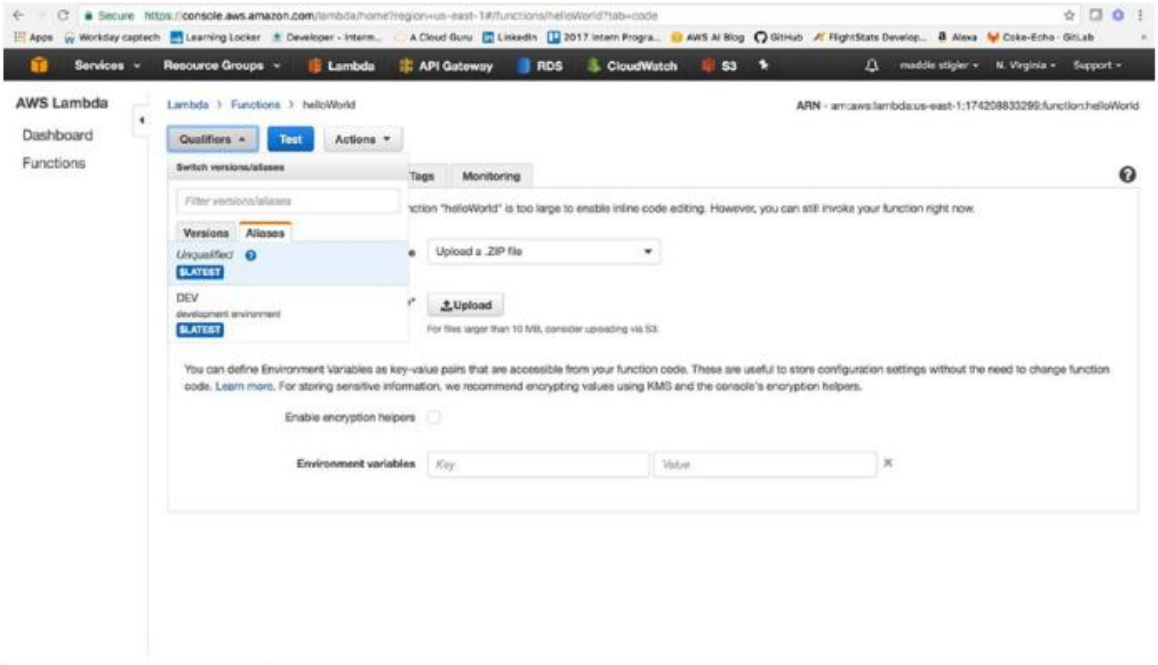
Penawaran tanpa server Amazon adalah AWS Lambda. AWS adalah penyedia cloud utama pertama yang menawarkan komputasi tanpa server, pada bulan November 2014. Lambda awalnya hanya tersedia dengan runtime Node.js, tetapi sekarang menawarkan C#, Java 8, dan Python. Fungsi Lambda dibuat secara independen dari sumber daya lain tetapi harus ditetapkan ke peran IAM (Identity and Access Management). Peran ini mencakup izin untuk CloudWatch, yang merupakan layanan pemantauan dan pencatatan cloud AWS. Dari konsol Lambda, Anda dapat melihat berbagai metrik pada fungsi Anda. Metrik ini disimpan dalam portal CloudWatch selama tiga puluh hari.

Gambar 2.1 mengilustrasikan metrik pencatatan CloudWatch yang tersedia. Fungsi AWS Lambda dapat ditulis di konsol AWS; namun, ini tidak direkomendasikan untuk proyek yang lebih besar. Saat ini, Anda tidak dapat melihat struktur proyek di dalam konsol. Anda hanya dapat melihat file index.js, atau fungsi yang menangani peristiwa tersebut. Hal ini menyulitkan pengembangan di dalam konsol. Meskipun Anda masih dapat mengekspor file dari konsol untuk melihat struktur file, Anda kemudian kembali dibatasi oleh proses penerapan dan pengujian.



Gambar 2.1 Log Pemantauan Yang Tersedia Di Cloudwatch

Seperti yang dapat Anda lihat, untuk fungsi Hello World ini, kami tidak memiliki pemanggilan apa pun dalam 24 jam terakhir. Bahkan ada lebih banyak metrik pencatatan yang dapat dilihat dari portal CloudWatch. Lambda memiliki alat Versioning dan Aliasing bawaan yang dapat digunakan langsung dari konsol juga. Alat-alat ini memungkinkan Anda membuat berbagai versi fungsi dan membuat alias versi tersebut ke berbagai tahap. Misalnya, jika Anda bekerja dengan lingkungan pengembangan, pengujian, dan produksi, Anda dapat membuat alias versi tertentu dari fungsi Lambda Anda ke masing-masing untuk menjaga lingkungan ini tetap terpisah. Gambar 2.2 mengilustrasikan contoh aliasing suatu versi fungsi Anda.



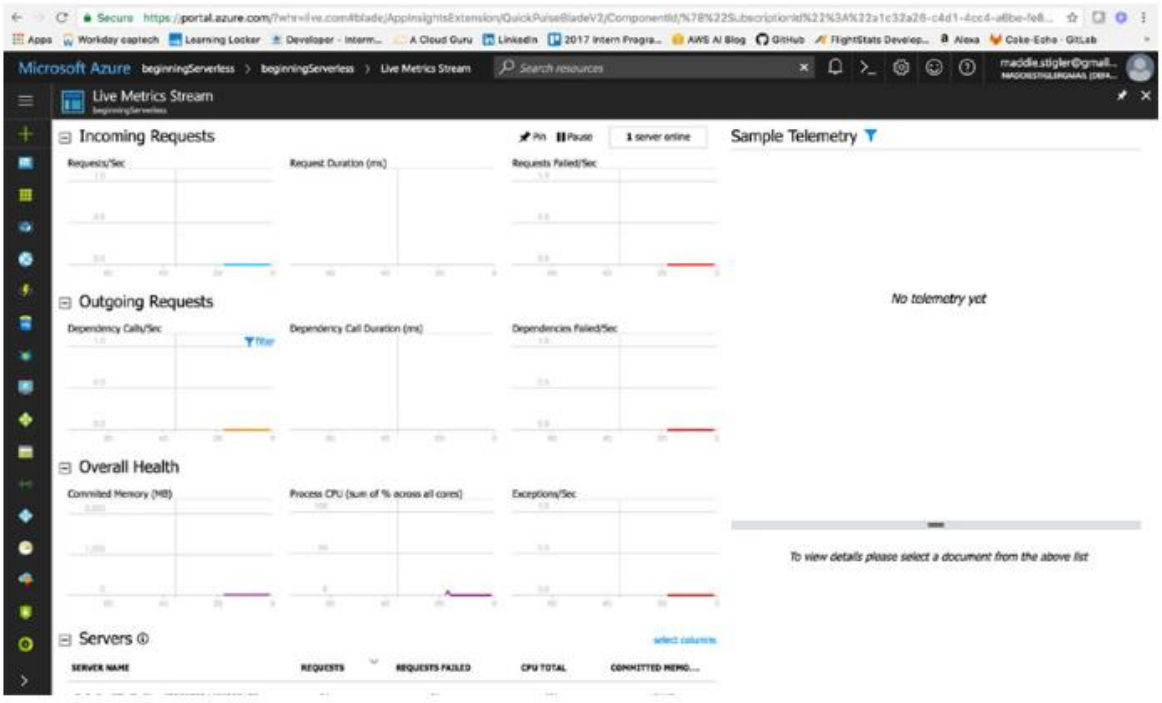
Gambar 2.2 Ilustrasikan Alias DEV Yang Selalu Menunjuk Ke Versi $Latest

Dari fungsi tersebut. $Latest hanya menunjukkan versi terbaru dari fungsi Lambda. AWS Lambda juga memudahkan penggabungan variabel lingkungan. Variabel ini dapat diatur menggunakan pasangan kunci/nilai, sehingga Anda dapat menggunakan variabel di seluruh fungsi untuk merujuk informasi yang dilindungi seperti kunci dan rahasia API, serta informasi basis data. Variabel ini juga memberi Anda cara yang lebih baik untuk meneruskan variabel ke fungsi tanpa harus mengubah kode di beberapa area. Misalnya, jika kunci berubah, Anda hanya perlu mengubahnya di satu tempat.

Azure Functions

Microsoft merilis penawaran tanpa servernya, Azure Functions, di konferensi Build pada tahun 2016. Meskipun dikembangkan hanya satu setengah tahun setelah AWS Lambda, Azure Functions tetap menjadi pesaing kuat di dunia tanpa server. Azure Functions mendukung JavaScript, C#, F#, Python, PHP, Bash, Batch, dan PowerShell. Salah satu kekuatan Azure adalah kemampuannya untuk mengintegrasikan Application Insights dengan fungsi Anda.

Meskipun AWS juga memiliki kemampuan ini, mengintegrasikan X-Ray dengan Lambda, penting untuk menunjukkan kekuatan Application Insights. Alat Manajemen Kinerja Aplikasi yang dapat diperluas ini untuk pengembang dapat digunakan di banyak platform. Alat ini menggunakan alat pemantauan yang canggih untuk membantu Anda memahami potensi kelemahan kinerja dalam aplikasi Anda. Gambar 2.3 mengilustrasikan Application Insights yang digunakan untuk pemantauan langsung suatu aplikasi.



Gambar 2.3 Live Metrics Streaming Memantau Permintaan Masuk, Permintaan Keluar, Kesehatan Keseluruhan, Dan Server Yang Digunakan Untuk Menangani Permintaan

Anda dapat melihat berapa lama permintaan berlangsung dan berapa banyak permintaan yang gagal. Anda dapat menggunakan statistik ini untuk menyesuaikan memori dan respons fungsi Anda. Aspek lain dari fungsi Azure adalah bahwa fungsi tersebut dibangun dalam grup sumber daya, kontainer yang digunakan untuk menampung semua sumber daya terkait untuk solusi Azure. Terserah pengembang untuk menentukan bagaimana sumber daya dikelompokkan dan dialokasikan, tetapi umumnya masuk akal untuk mengelompokkan sumber daya aplikasi yang berbagi siklus hidup yang sama sehingga dapat disebarkan, diperbarui, dan dihapus bersama-sama.

Fungsi Lambda diatur secara independen. Fungsi tersebut tidak diharuskan menjadi bagian dari grup sumber daya, tetapi dapat dikembangkan sepenuhnya secara terpisah dari sumber daya AWS lainnya. Salah satu keterbatasan potensial untuk fungsi tanpa server yang kita bahas di Bab 1 adalah ketakutan akan "cold start". Fungsi Azure berjalan di atas WebJobs, yang berarti file fungsi tidak hanya berada dalam file zip. Fungsi tersebut dibangun di atas WebJobs untuk lebih mudah menghosting proses back-end yang panjang atau pendek. Fungsi Azure juga terintegrasi dengan beberapa alat penerapan berkelanjutan, seperti Git, Visual Studio Team Services, OneDrive, Dropbox, dan editor bawaan Azure sendiri.

Visual Studio Team Services (sebelumnya Visual Studio Online) adalah alat yang hebat untuk integrasi berkelanjutan fungsi Anda dengan tim. Integrasi yang erat dengan Visual Studio Team Services berarti Anda dapat mengonfigurasi koneksi ke Azure dan menerapkannya dengan sangat mudah. Ini juga memberi Anda templat fungsi Azure gratis di luar kotak untuk mempercepat proses pengembangan lebih jauh. Saat ini, integrasi ini bukanlah sesuatu yang disediakan oleh AWS atau Google Cloud. Ini termasuk Git, repo pribadi gratis, alat pengembangan tangkas, manajemen rilis, dan integrasi berkelanjutan.

Google Cloud Functions

Google Cloud merilis penawaran tanpa servernya, Google Cloud Functions, pada bulan Februari 2016. Saat ini, Google Cloud hanya mendukung runtime JavaScript dengan hanya tiga pemicu. Penting untuk diingat bahwa, saat tulisan ini dibuat, Google Cloud Functions masih dalam versi Beta. Banyak fungsi dan lingkungannya yang dapat berubah dengan lebih banyak pengembangan pada penawaran layanannya yang diharapkan.

Google Cloud

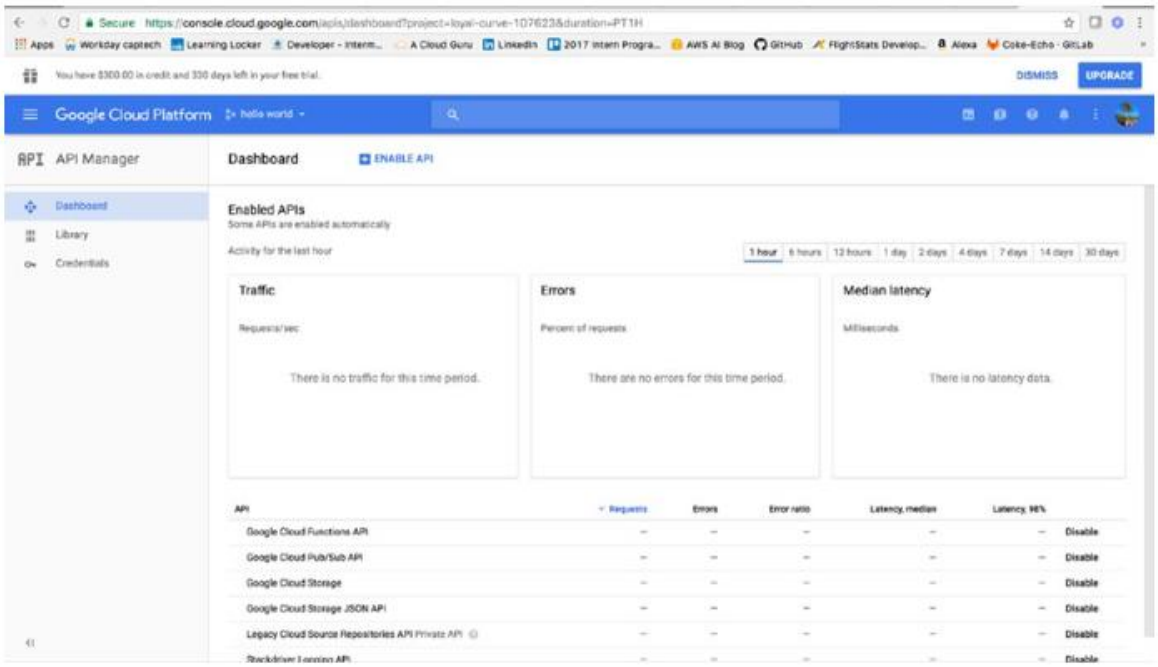
Functions telah mengaktifkan pencatatan otomatis dan menulisnya ke alat Pencatatan Stackdriver. Log tetap berada di Stackdriver hingga tiga puluh hari dan mencatat wawasan waktu nyata serta log khusus. Selain itu, kinerja dicatat dalam Stackdriver Monitoring dan Stackdriver Debugger memungkinkan Anda untuk men-debug perilaku kode Anda dalam produksi. Dengan Google Cloud Functions, Anda juga dapat menggunakan repositori Cloud Source untuk menyebarkan fungsi secara langsung dari repositori GitHub atau bitbucket.

Ini mengurangi waktu yang akan dihabiskan untuk mengompres dan mengunggah kode secara manual melalui konsol. Ini juga memungkinkan Anda untuk terus menggunakan bentuk kontrol versi seperti sebelumnya. Aspek unik dari Google Cloud Functions adalah integrasinya dengan Firebase. Pengembang seluler dapat mengintegrasikan platform Firebase

dengan fungsi mereka dengan lancar. Fungsi Anda dapat merespons peristiwa berikut yang dihasilkan oleh Firebase:

- Pemicu basis data waktu nyata
- Pemicu autentikasi Firebase
- Pemicu Google Analytics untuk Firebase
- Pemicu penyimpanan cloud
- Pemicu pub/sub cloud
- Pemicu HTTP

Cloud Functions meminimalkan kode boilerplate, sehingga Anda dapat dengan mudah mengintegrasikan Firebase dan Google Cloud dalam fungsi Anda. Firebase juga tidak memerlukan banyak atau tidak memerlukan pemeliharaan. Dengan menyebarkan kode Anda ke fungsi, pemeliharaan yang terkait dengan kredensial, konfigurasi server, dan penyediaan serta penyediaan server akan hilang. Anda juga dapat memanfaatkan Firebase CLI untuk menyebarkan kode Anda dan konsol Firebase untuk melihat dan mengurutkan log.



Gambar 2.4 Dasbor Pengelola API Menampilkan Semua API Yang Saat Ini Diaktifkan, Beserta Permintaan, Kesalahan, Latensi, Dan Lalu Lintas Yang Terkait Dengan API Tersebut

Statistik dasbor tersebut berlaku hingga tiga puluh hari yang lalu. Agar dapat menjalankan dan menguji kode Anda secara lokal, Google Cloud menyediakan emulator fungsi. Ini adalah repositori Git yang memungkinkan Anda menyebarkan, menguji, dan menjalankan fungsi Anda di komputer lokal sebelum menyebarkannya langsung ke Google Cloud. Perbedaan antara platform Google Cloud dan Azure atau AWS adalah ketergantungan yang tinggi pada API untuk setiap layanan.

Hal ini serupa dengan Kit Pengembangan Perangkat Lunak yang digunakan di AWS dan Azure; namun, tingkatnya lebih rendah. Google Cloud bergantung pada pustaka klien API untuk memperoleh fungsionalitas layanan. API ini memungkinkan Anda mengakses produk platform Google Cloud dari kode Anda dan mengotomatiskan alur kerja Anda. Anda dapat mengakses dan mengaktifkan API ini melalui Dasbor Pengelola API, yang ditunjukkan pada Gambar 2.4.

2.2 JELAJAHI PEMICU DAN PERISTIWA

Bab 1 memberikan gambaran umum tentang pemicu dan peristiwa serta bagaimana keduanya sesuai dengan gagasan yang lebih luas tentang arsitektur tanpa server. Di bagian ini, kita akan membahas apa itu pemicu, bagaimana keduanya bekerja dengan penyedia cloud yang berbeda dan dalam contoh dunia nyata, serta bagaimana peristiwa mendorong fungsi tanpa server.

Apa itu Pemicu?

Pemicu hanyalah peristiwa. Pemicu adalah layanan dan permintaan HTTP yang membuat peristiwa untuk mengaktifkan fungsi dan memulai respons. Pemicu biasanya ditetapkan dalam konsol fungsi atau antarmuka baris perintah dan biasanya dibuat dalam lingkungan penyedia cloud yang sama. Suatu fungsi harus memiliki tepat satu pemicu. Di AWS, pemicu dapat berupa permintaan HTTP atau pemanggilan layanan AWS lainnya. Fungsi Azure juga menggunakan pemicu layanan, tetapi pemicu juga menangkap gagasan tentang pengikatan.

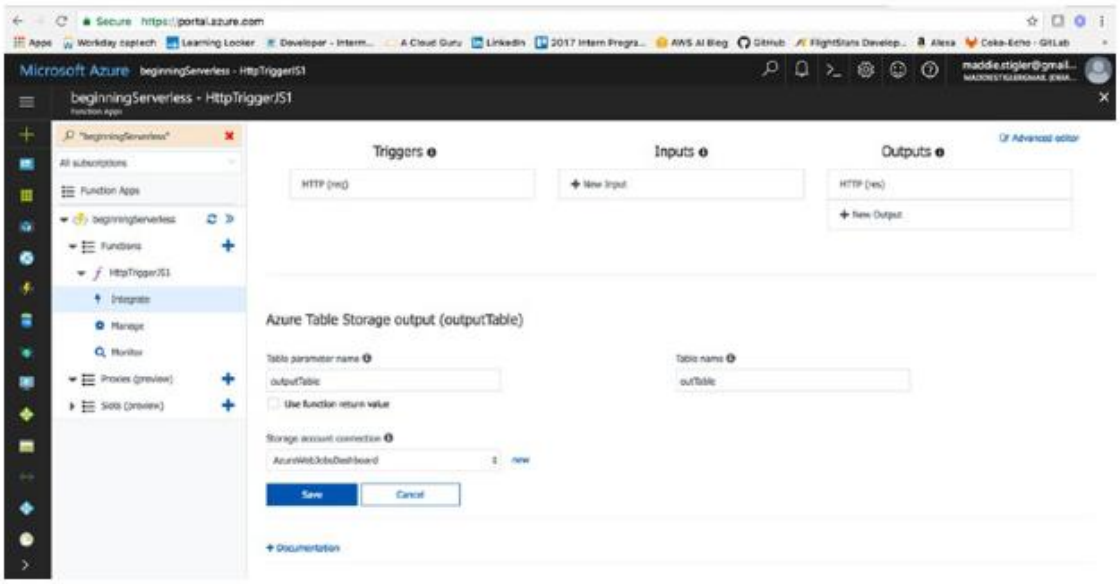
Pengikatan input dan output menawarkan cara deklaratif untuk terhubung ke data dari dalam kode Anda. Pengikatan tidak berbeda dengan pemicu karena Anda, sebagai pengembang, menentukan string koneksi dan properti lainnya dalam konfigurasi fungsi Anda. Tidak seperti pemicu, pengikatan bersifat opsional dan suatu fungsi dapat memiliki banyak pengikatan. Tabel 2.1 mengilustrasikan pengikatan input dan output yang didukung Azure untuk fungsinya.

Tabel 2.1 Pengikatan Input/Output untuk Fungsi Azure

Masukan	Pengeluaran
Penyimpanan blob	HTTP (rest atau webhook) Penyimpanan blob Events Antrian Antrian dan topik
Tabel penyimpanan	Tabel penyimpanan
Tabel SQL	Tabel SQL
NoSQL DB	NoSQL DB Notifikasi dorong Pesan teks sms Twillio SendGrid Email

Contoh aplikasi yang mengikat pemicu ke fungsi adalah menulis ke tabel dengan permintaan API. Misalnya kita memiliki tabel di Azure yang menyimpan informasi karyawan dan setiap kali permintaan POST masuk dengan informasi karyawan baru, kita ingin menambahkan baris lain ke tabel. Kita dapat melakukannya menggunakan pemicu HTTP, fungsi Azure, dan pengikatan keluaran Tabel. Dengan menggunakan pemicu dan pengikatan, kita dapat menulis kode yang lebih umum yang tidak membuat fungsi bergantung pada detail layanan yang berinteraksi dengannya. Data peristiwa yang masuk dari layanan menjadi nilai input untuk fungsi kita.

Mengeluarkan data ke layanan lain, seperti menambahkan baris ke tabel di Azure Table Storage, dapat dilakukan menggunakan nilai pengembalian fungsi kita. Pemicu dan pengikatan HTTP memiliki properti nama yang berfungsi sebagai pengenal yang akan digunakan dalam kode fungsi untuk mengakses pemicu dan pengikatan. Pemicu dan pengikatan dapat dikonfigurasi di tab integrasi di portal Azure Functions. Konfigurasi ini tercermin dalam file function.json di direktori fungsi. File ini juga dapat dikonfigurasi secara manual di Editor Lanjutan. Gambar 2.5 menunjukkan fungsionalitas integrasi dengan pengaturan input dan output yang dapat dikonfigurasi.



Gambar 2.5 Pemicu, Masukan, Dan Keluaran Yang Dapat Diatur Dan Dikonfigurasi Dalam Portal Azure

Kemampuan untuk mengonfigurasi keluaran menggunakan pengikatan dalam Azure adalah sesuatu yang tidak tersedia pada setiap penyedia cloud, tetapi memiliki keluaran tertentu berdasarkan penerimaan peristiwa pemicu adalah konsep yang dianut oleh penyedia cloud lain dan sesuai dengan gagasan untuk membuat fungsi tanpa server guna menjalankan operasi tunggal.

Pemicu dalam Penyedia Cloud

Penyedia cloud yang berbeda menawarkan pemicu yang berbeda untuk fungsi mereka. Meskipun banyak dari mereka pada dasarnya adalah layanan yang sama dengan nama yang berbeda berdasarkan penyedia, beberapa benar-benar unik. Tabel 2.2 menunjukkan pemicu untuk penyedia yang akan kita gunakan.

Tabel 2.2 Pemicu fungsi untuk AWS, Azure, dan Google

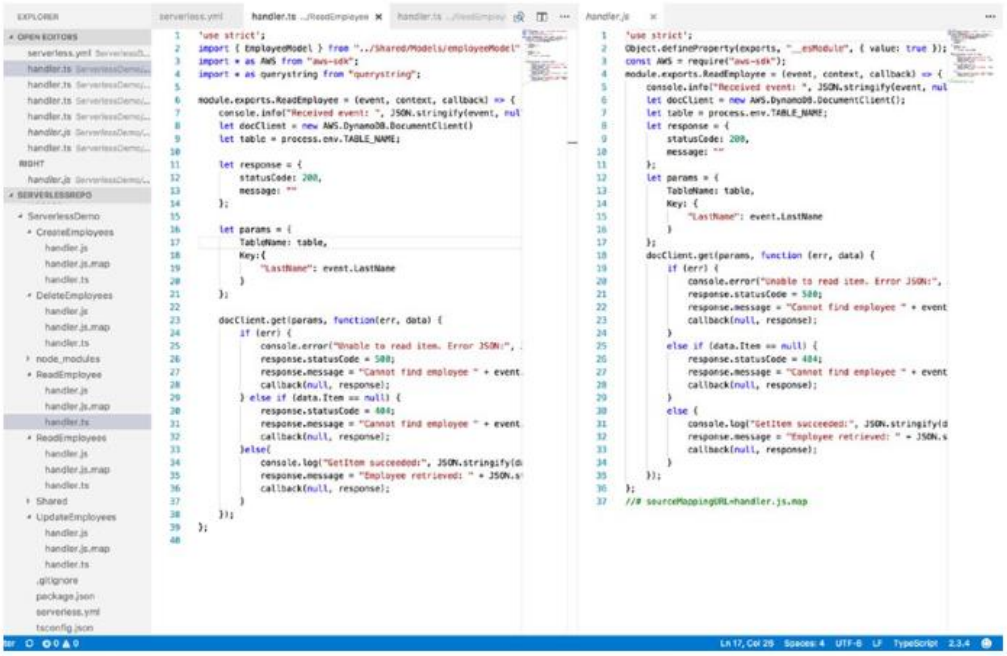
AWS Lambda	Fungsi Azure	Fungsi Google Cloud
Amazon S3	Penyimpanan Azure	
Amazon DynamoDB		
Amazon Kinesis Stream	Azure Event Hub (Pusat Peristiwa)	
Amazon Simple Notification Service	Antrian dan topik	Google Cloud Pub atau Sub triggers
Amazon Simple Email Service		
Amazon Cognito		
AWS CloudFormation		
Amazon CloudWatch Logs		
Amazon CloudWatch Events		
AWS CodeCommit		
Jadwal Event	Jadwal Azure	
AWS Config		
Amazon Alexa		
Amazon Lex		
Amazon API Gateway	HTTP (REST atau WebHook)	HTTP

2.3 OPSI PENGEMBANGAN, TOOLKIT, SDK

Di bagian ini, kita akan melihat berbagai opsi pengembangan, toolkit, dan SDK yang dapat digunakan untuk mengembangkan aplikasi tanpa server. Secara khusus, kita akan membahas TypeScript dengan Node.js, AWS SDK, Azure SDK, dan Cloud SDK untuk Google.

TypeScript dengan Node.JS

TypeScript adalah superset JavaScript yang dikembangkan oleh Microsoft untuk mengembangkan bahasa dengan tipe kuat yang dikompilasi ke JavaScript. Dimulai dengan sintaksis yang sama yang diketahui dan digunakan oleh pengembang yang bekerja dengan JavaScript saat ini. TypeScript dikompilasi langsung ke kode JavaScript yang berjalan di browser apa pun di banyak mesin JavaScript, termasuk Node.js. TypeScript memungkinkan pengembang untuk membangun aplikasi JavaScript yang juga akan mencakup pemeriksaan statis dan pemfaktoran ulang kode. Gambar 2.6 mengilustrasikan contoh kompilasi TypeScript ke JavaScript.



Gambar 2.6 Penggunaan Typescript Untuk Membuat Fungsi Create Employee Dan Cara Mengompilasinya Menjadi Kode Javascript Yang Dapat Digunakan Untuk Membangun Aplikasi Tanpa Server

TypeScript dapat diunduh menggunakan plug-in NPM atau Visual Studio. Dari terminal/command prompt, Anda dapat menginstal TypeScript dengan perintah npm install –g typescript. Ini memberi Anda akses ke alat TypeScript. Di Visual Studio, TypeScript disertakan secara default; jadi saya sarankan menginstal Visual Studio Code pada tahap ini jika diperlukan. Setelah TypeScript terinstal, Anda dapat mulai menulis file TypeScript dan mengompilasinya menggunakan baris perintah:

```
tsc helloWorld.ts
```

Atau dengan membangun proyek di Visual Studio. Setelah file TypeScript dikompilasi, Anda akan melihat file JavaScript yang dibuat dari file TypeScript.

AWS SDK

Kit pengembangan perangkat lunak (SDK) merupakan alat yang ampuh untuk mengembangkan aplikasi tanpa server. AWS, Azure, dan Google masing-masing memiliki SDK yang dapat digunakan pengembang untuk mengakses dan membuat layanan dengan mudah di setiap penyedia cloud. AWS menawarkan SDK untuk semua bahasa pemrograman dan platform berikut:

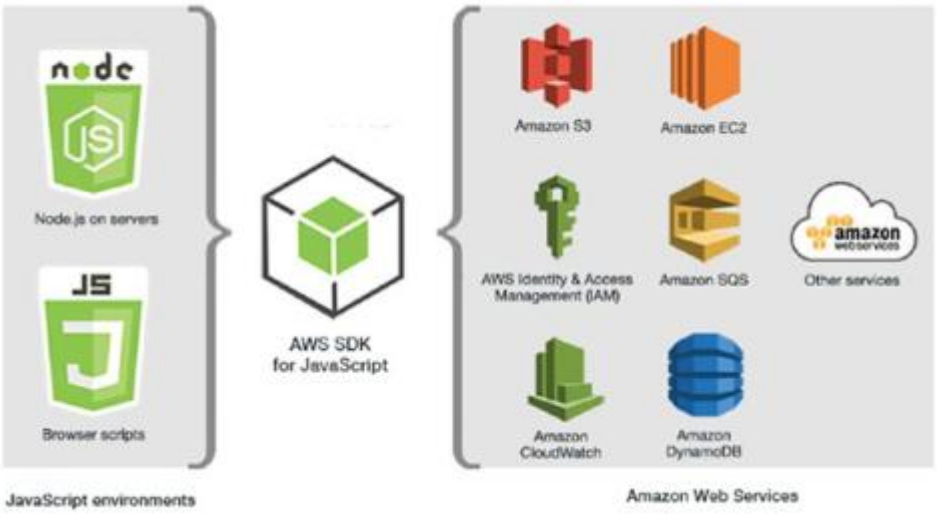
- Android
- Browser
- iOS
- Java

- .NET
- Node.js
- Ruby
- Python
- PHP
- Go
- C++
- AWS Mobile
- AWS IoT

Untuk menginstal AWS SDK, cukup ketik perintah ini di command prompt Anda:

```
npm install aws-sdk
```

Sebelum Anda dapat melakukannya, Anda harus menginstal NPM. Node Package Manager akan mengurus instalasi untuk Anda dan membuatnya dapat diakses di proyek Anda. Gambar 2.7 mengilustrasikan cara kerja AWS SDK dengan Node.js untuk memberikan layanan AWS yang dapat diakses ke kode Anda.



Gambar 2.7 Ilustrasi Bagaimana AWS SDK Untuk Javascript Memungkinkan Anda Membangun Aplikasi Skala Penuh Yang Memanfaatkan Layanan Yang Ditawarkan AWS Dengan Sedikit Usaha

Setelah menginstal SDK, Anda perlu melakukan beberapa konfigurasi dalam file Node.js untuk memuat paket AWS ke dalam aplikasi. Cara melakukannya adalah dengan menggunakan pernyataan require di bagian atas JavaScript Anda. Kode tersebut akan terlihat seperti ini:

```
var AWS = require('aws-sdk');
```

Anda kemudian dapat mengakses berbagai sumber daya AWS menggunakan variabel AWS yang Anda buat dan materi referensi API yang dapat ditemukan di sini:

```
https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/index.html
```

Dokumentasi ini akan menunjukkan kepada Anda cara membuat dan mengakses layanan tertentu. Contoh kode berikut menunjukkan cara membuat tabel di DynamoDB menggunakan AWS SDK.

```
'use strict';
Object.defineProperty(exports, "_esModule", { value: true });
var AWS = require("aws-sdk");
module.exports.CreateTable = (event, context, callback) => {
   var dynamodb = new AWS.DynamoDB();
   var docClient = new AWS.DynamoDB.DocumentClient();
   var params = {
     TableName: process.env.TABLE_NAME,
     KeySchema: [
     { AttributeName: "LastName", KeyType: "HASH" } //Partition
     key
],
AttributeDefinitions: [
 {AttributeName: "LastName", AttributeType: "S" }
],
ProvisionedThroughput: {
  ReadCapacityUnits: 10,
  WriteCapacityUnits: 10
 }
};
dynamodb.createTable(params, function (err, data) {
  if (err) {
   console.error("Unable to create table. Error JSON:",
   JSON.stringify(err, null, 2));
   }
   else {
     console.log("Created table. Table description JSON:",
     JSON.stringify(data, null, 2));};
   }
 });
};
```

AWS SDK akan digunakan dengan cara yang sama di beberapa demo kami di bab berikutnya. Sebaiknya Anda membaca dokumentasi API untuk mendapatkan pemahaman yang lebih baik tentang cara layanan ini dibuat dan cara penggunaannya di seluruh aplikasi Anda.

Azure SDK

Mirip dengan AWS SDK, Azure juga memiliki SDK yang dapat Anda gunakan saat membuat fungsi Azure. Daftar SDK yang tersedia untuk berbagai alat dan platform meliputi:

- .NET
- Java
- Node.js
- PHP
- Python
- Ruby
- Mobile
- Media
- Android
- iOS
- JavaScript
- Swift
- Windows

Karena kita akan menggunakan runtime Node.js untuk membuat aplikasi kita dalam demo berikut, kita akan terus melihat contoh penggunaan SDK dengan JavaScript. Anda bisa mendapatkan Azure SDK dengan menggunakan perintah npm install azure. Sama seperti AWS, Node Package Manager akan menginstal kit pengembangan Azure untuk Anda. Jika Anda hanya ingin menginstal modul individual untuk layanan tertentu, Anda juga dapat melakukannya melalui NPM. Kode berikut menunjukkan cara mudah membuat database di DocumentDB menggunakan Azure SDK:

```
var DocumentClient = require('documentdb').DocumentClient;
var host = 'host';
var key = 'key';
var dbClient = new DocumentClient(host, {masterKey: key});
var databaseDefinition = { id: 'myDatabase' };

//Create Database
Client.createDatabasedatabaseDefinition,function(err,databas){
    if(err) return console.log(err);
    console.log('Database Created');
});
```

JavaScript ini menggunakan klien DocumentDB untuk membuat dan membuat instance database DocumentDB baru di Azure. Pernyataan require mengumpulkan modul dari Azure dan memungkinkan Anda untuk melakukan beberapa operasi DocumentDB langsung dari fungsi Anda. Kami akan menggunakan ini secara lebih rinci dalam tutorial Azure.

Google Cloud SDK

SDK Google Cloud juga mendukung berbagai alat dan platform:

- Java
- Python
- Node.js
- Ruby
- GO
- .NET
- PHP

Namun, karena Google Cloud Functions saat ini hanya mendukung Node.js, Node.js SDK untuk Google Cloud adalah yang akan kami gunakan untuk mengimplementasikan aplikasi tanpa server. Cloud SDK memiliki banyak fitur yang perlu dijelaskan lebih lanjut. Alat gcloud mengelola autentikasi, konfigurasi lokal, alur kerja developer, dan interaksi dengan Cloud Platform API. Alat gsutil menyediakan akses baris perintah untuk mengelola bucket dan objek Cloud Storage.

Kubectl mengatur penyebaran dan pengelolaan kluster kontainer Kubernetes di gcloud. Bq memungkinkan Anda menjalankan kueri, memanipulasi set data, tabel, dan entitas di BigQuery melalui baris perintah. Anda dapat menggunakan alat ini untuk mengakses Google Compute Engine, Google Cloud Storage, Google BigQuery, dan layanan lainnya dari baris perintah. Dengan alat gcloud, Anda dapat memulai dan mengelola berbagai emulator Cloud SDK yang dibuat untuk Google Cloud Pub/Sub dan Google Cloud Datastore. Ini berarti Anda akan memiliki kemampuan untuk mensimulasikan layanan ini di lingkungan lokal Anda untuk pengujian dan validasi.

Anda juga memiliki kemampuan untuk menginstal pustaka klien khusus bahasa melalui Cloud SDK. Untuk menginstal Cloud SDK untuk Node.js, masukkan perintah berikut ke terminal Anda: npm install –save google- cloud. Google Cloud juga menyarankan Anda menginstal alat SDK baris perintah. Untuk melakukannya, Anda dapat menginstal SDK khusus untuk mesin Anda dari situs ini: https://cloud.google.com/sdk/docs/. Kode berikut menunjukkan cara menggunakan Google Cloud SDK untuk Node.js untuk mengunggah file ke penyimpanan cloud.

```
var googleCloud = require('google-cloud')({
  projectId: 'my-project-id',
  keyFilename: '/path/keyfile.json'
});

var googleStorage = googleCloud.storage();
var backups = googleStorage.bucket('backups');
```

```
backups.upload('file.zip', function(err, file) {
});
```

JavaScript memerlukan modul google-cloud, yang memungkinkan Anda memanfaatkan dan mengubah berbagai layanan Google Cloud dalam kode Anda. Meskipun SDK ini tidak terintegrasi seperti SDK AWS dan Azure, SDK ini terus berkembang dan dapat digunakan untuk membuat dan menyebarkan fungsi serta layanan lainnya.

2.4 MENGEMBANGKAN SECARA LOKAL VS. MENGGUNAKAN KONSOL

Bagaimana Anda harus mulai mengembangkan aplikasi tanpa server? Apakah Anda membangunnya secara lokal lalu menyebarkannya ke penyedia cloud, atau apakah Anda membangunnya di dalam konsol penyedia cloud? Campuran? Bagian ini membahas praktik terbaik dan opsi untuk mengembangkan secara lokal dan di dalam lingkungan penyedia.

Pengembangan Lokal

Pengembangan secara lokal sering kali lebih disukai karena berarti Anda dapat menggunakan alat, IDE, dan lingkungan yang biasa Anda gunakan. Namun, bagian yang sulit tentang pengembangan secara lokal adalah mengetahui cara mengemas dan menyebarkan fungsi Anda ke cloud sehingga Anda menghabiskan lebih sedikit waktu untuk mencari tahu hal ini dan lebih banyak waktu mengerjakan logika kode Anda. Mengetahui praktik terbaik untuk struktur dan pengujian proyek dapat membantu mempercepat proses pengembangan sambil tetap memungkinkan Anda mengembangkan menggunakan alat Anda sendiri.

Untuk fungsi AWS Lambda, penting untuk diingat bahwa fungsi pengendali harus berada di akar folder zip. Di sinilah AWS berupaya mengeksekusi fungsi Anda saat dipicu. Menstrukturkan proyek Anda dengan cara yang menegakkan aturan eksekusi ini diperlukan. Untuk pengujian lokal, paket NPM lambda-local memungkinkan Anda membuat dan menyimpan peristiwa pengujian yang dapat Anda jalankan pada fungsi Anda secara lokal sebelum meluangkan waktu untuk menerapkannya ke AWS.

Jika Anda tidak menggunakan kerangka kerja yang mengotomatiskan penerapan ini untuk Anda, sebaiknya gunakan paket seperti lambda-local. Azure juga menawarkan paket NPM yang dapat menguji fungsi Anda secara lokal. Azure Functions Core Tools adalah versi lokal dari runtime Azure Functions yang memungkinkan Anda membuat, menjalankan, men-debug, dan menerbitkan fungsi secara lokal. Paket Azure NPM saat ini hanya berfungsi di Windows.

Visual Studio menawarkan alat untuk fungsi Azure yang menyediakan templat, kemampuan untuk menjalankan dan menguji fungsi, dan cara untuk menerbitkan langsung ke Azure. Alat-alat ini cukup canggih dan memberi Anda banyak fungsi langsung dari kotaknya. Beberapa keterbatasan alat-alat ini termasuk IntelliSense yang terbatas, ketidakmampuan untuk menghapus file tambahan di tujuan, dan ketidakmampuan untuk menambahkan item baru di luar penjelajah file. Google Cloud memiliki rilis Alfa emulator lokal fungsi cloud. Saat ini, emulator memungkinkan Anda untuk menjalankan, men-debug, dan menyebarkan fungsi secara lokal sebelum menyebarkannya langsung ke cloud.

Penyebaran Fungsi dan Sumber Daya

Ada beberapa opsi untuk penyebaran dari lingkungan lokal ke lingkungan cloud. Menggunakan Serverless Framework merupakan metode yang lebih disukai karena membangun paket penyebaran ringkas yang khusus untuk penyedia sehingga Anda dapat menggunakannya untuk membangun aplikasi yang sama di akun mana pun. Metode ini juga lebih disukai karena memungkinkan Anda untuk membuat layanan dan keamanan dependen secara bersamaan. Opsi lain untuk menyebarkan dari lingkungan lokal ke cloud adalah menggunakan antarmuka baris perintah penyedia. AWS, Azure, dan Google Cloud semuanya menawarkan CLI yang dapat diinstal dan digunakan untuk membuat dan menyebarkan berbagai layanan. AWS CLI dapat diinstal jika Anda memiliki Python dan pip menggunakan perintah ini:

```
pip install --upgrade  --user awscli
```

Setelah CLI terinstal, Anda dapat mengonfigurasi akun AWS CLI Anda menggunakan perintah:

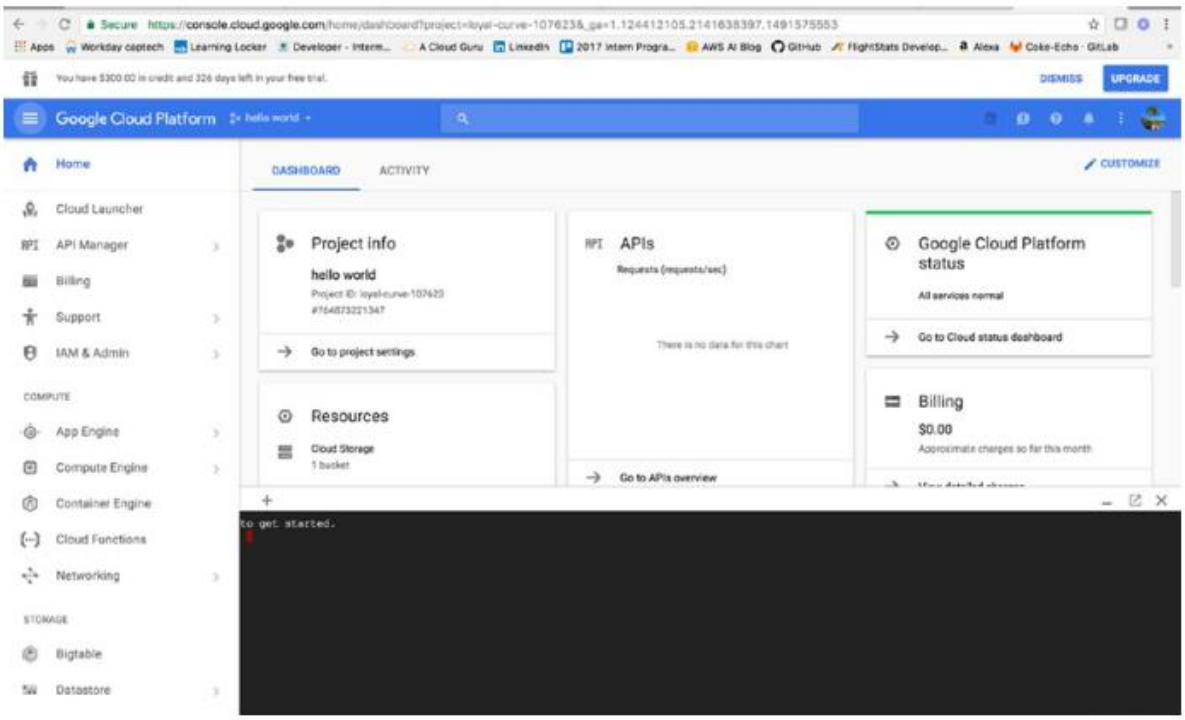
```
aws configure
```

Konfigurasi ini akan menanyakan AWS Access Key ID, AWS Secret Access Key, nama wilayah default, dan format output default. Setelah nilai-nilai ini dikonfigurasi, Anda dapat menggunakan CLI untuk membuat dan mengonfigurasi layanan AWS Anda. Azure juga menawarkan antarmuka baris perintah serta perintah PowerShell untuk mengelola dan menyebarkan sumber daya Azure Anda. Untuk menginstal Azure CLI dengan perintah bash, gunakan:

```
curl -L https://aka.ms/InstallAzureCli I bash
```

Azure juga telah merilis Cloud Shell, shell interaktif yang dapat diakses melalui browser untuk mengelola sumber daya Azure. Cloud Shell dapat diluncurkan dari portal Azure dan memungkinkan Anda memiliki pengalaman shell yang dapat diakses melalui browser tanpa harus mengelola atau menyediakan mesin sendiri. Ini memungkinkan Anda membuat dan mengelola skrip Azure untuk sumber daya dengan mudah. Untuk memulai dengan Cloud Shell, saya sarankan untuk mengikuti tutorial yang disediakan oleh Microsoft di https://docs.microsoft.com/en-us/azure/cloud-shell/quickstart.

Google Cloud juga memanfaatkan CLI dalam Cloud Shell yang memungkinkan Anda mengakses dan menyebarkan sumber daya lokal. Ini memungkinkan Anda mengelola proyek dan sumber daya tanpa harus menginstal Cloud SDK atau alat lain secara lokal. Untuk memanfaatkan Google Cloud Shell, yang harus Anda lakukan hanyalah mengaktifkannya dari konsol. Untuk memulai Cloud Shell, Anda cukup mengaktifkannya di konsol seperti yang Anda lakukan untuk Azure. Gambar 2.8 menunjukkan contoh pengaktifan Cloud Shell.



Gambar 2.8 Google Cloud Shell Diaktifkan Dengan Mengeklik Ikon Shell Di Kanan Atas; Lalu Berjalan Di Layar Shell Di Bagian Bawah

Anda dapat menggunakan salah satu alat ini untuk mengembangkan dan mengonfigurasi fungsi tanpa server dan sumber daya terkait. Agar konsisten, kami akan menggunakan Serverless Framework, yang dapat diakses oleh ketiga penyedia yang akan kami bahas.

Mengembangkan dan Menguji di Cloud Console

Mengembangkan fungsi di cloud console cenderung sedikit lebih rumit daripada mengembangkan secara lokal. Satu hal yang baik tentang pengembangan di konsol adalah Anda tidak perlu khawatir tentang penerapan fungsi; yang harus Anda lakukan hanyalah menguji dan menyimpannya. Azure memungkinkan Anda menavigasi melalui struktur proyek dan membuat perubahan di konsol. AWS memungkinkan Anda melihat file pengendali selama paket penerapan Anda tidak terlalu besar untuk memungkinkan pengeditan sebaris. Google Cloud juga memungkinkan pengeditan sebaris untuk paket penerapan yang lebih kecil.

Masing-masing penyedia cloud juga memungkinkan Anda menentukan kasus pengujian yang dapat Anda simpan dan gunakan untuk menguji fungsi saat Anda membuat perubahan pada fungsi tersebut. Mereka juga menyediakan pemantauan dan pencatatan yang tidak selalu Anda dapatkan secara lokal. Ini memberi pengembang riwayat wawasan tentang fungsi mereka dan bagaimana mereka menanggapi berbagai peristiwa. Menetapkan pemicu juga dapat lebih mudah di lingkungan penyedia cloud. AWS, Azure, dan Google memudahkan penetapan pemicu ke suatu fungsi di dalam konsol. Mereka juga menyediakan kasus pengujian templat yang dapat digunakan untuk menguji fungsi langsung dari kotaknya.

Seiring dengan semakin berkembangnya kemampuan dan fungsionalitas platform nirserver dari penyedia ini, saya dapat membayangkan pengembangan di konsol akan menjadi

jauh lebih umum. Untuk saat ini, metode yang lebih disukai adalah mengembangkan secara lokal menggunakan alat dan IDE Anda dan mengandalkan alat penerapan tingkat lanjut untuk menerapkan dan menguji di cloud. Hal ini juga dapat ditentukan oleh ukuran bisnis Anda. Misalnya, bisnis yang lebih besar mungkin lebih suka Anda menggunakan lingkungan pengembangan di tempat, sedangkan saat mengembangkan sendiri, Anda dapat membuat di dalam konsol tanpa kendala apa pun.

2.5 ALAT

Bagian ini akan membahas berbagai alat yang akan digunakan untuk mengembangkan dan menerapkan fungsi nirserver kita di AWS, Azure, dan Google Cloud. Alat-alat ini meliputi Visual Studio Code sebagai IDE kita, Node.js sebagai runtime kita, dan Postman untuk menguji API kita dan memicu fungsi kita.

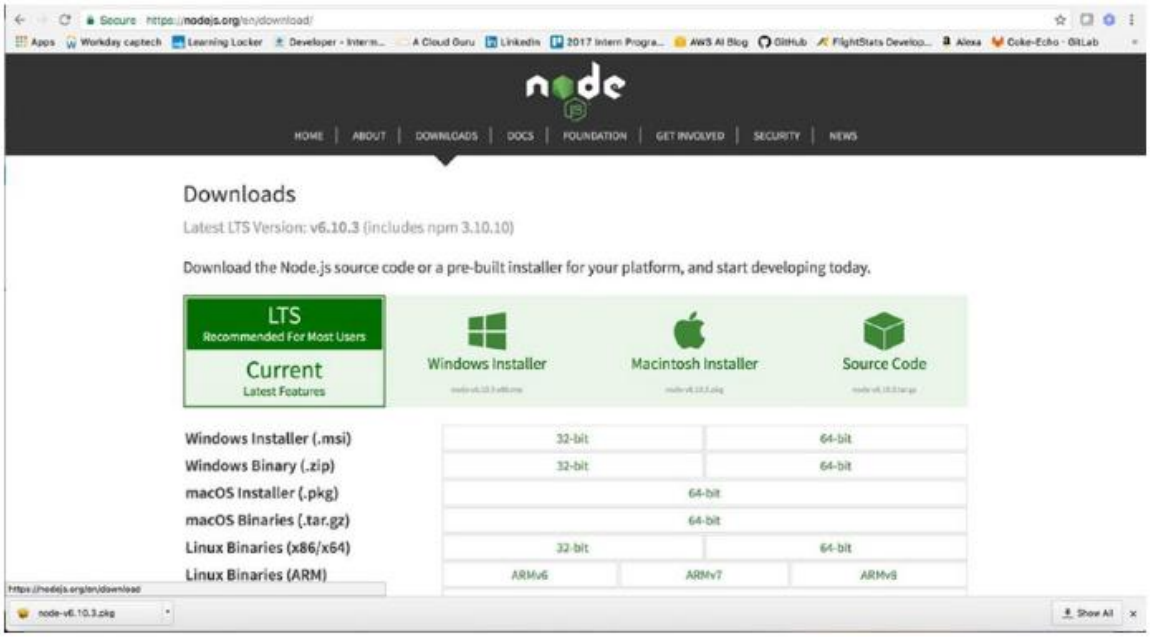
Memasang VS Code atau Memilih IDE Anda

Visual Studio Code adalah IDE pilihan saya karena kemudahan penggunaannya, wawasan bawaan, dan aksesibilitas lintas platform. Anda juga dapat memasang VS Code di Windows atau Mac, yang merupakan fitur yang hebat. Saya akan bekerja di dalam VS Code untuk tutorial berikut; Jadi, meskipun Anda dipersilakan menggunakan IDE Anda sendiri, mengikuti petunjuk mungkin lebih mudah di lingkungan yang sama. Untuk mengunduh VS Code, buka https://code.visualstudio.com/ dan unduh untuk komputer Anda.

Node.js

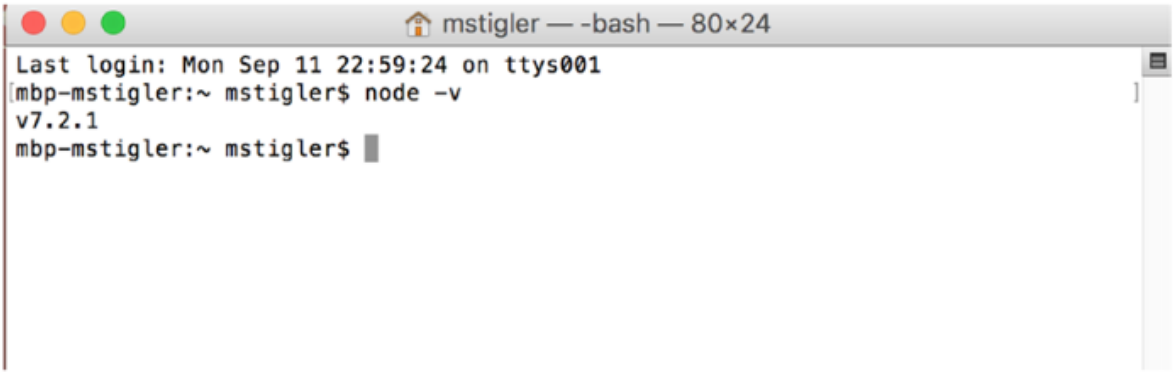
Node.js adalah satu-satunya runtime yang didukung oleh ketiga penyedia cloud, jadi kita juga akan membuat fungsi menggunakan Node. Selain menjadi satu-satunya runtime yang sepenuhnya didukung, Node adalah JavaScript yang digerakkan oleh peristiwa dan dapat diskalakan yang sesuai dengan kebutuhan untuk membangun fungsi yang ringan dan dapat diskalakan.

Untuk menginstal Node, navigasikan ke https://nodejs.org/en/download/ dan temukan penginstal yang kompatibel dengan komputer Anda. Hanya butuh sedikit waktu untuk mengunduh penginstal lalu ikuti langkah-langkahnya untuk menyelesaikan instalasi node. Gambar 2.9 menunjukkan opsi unduhan Node.js.



Gambar 2.9 Halaman Unduhan Nodejs Memungkinkan Anda Mengunduh Node Untuk Platform Windows Atau Mac. Halaman Ini Juga Memungkinkan Anda Memilih Metode Dan Paket Instalasi

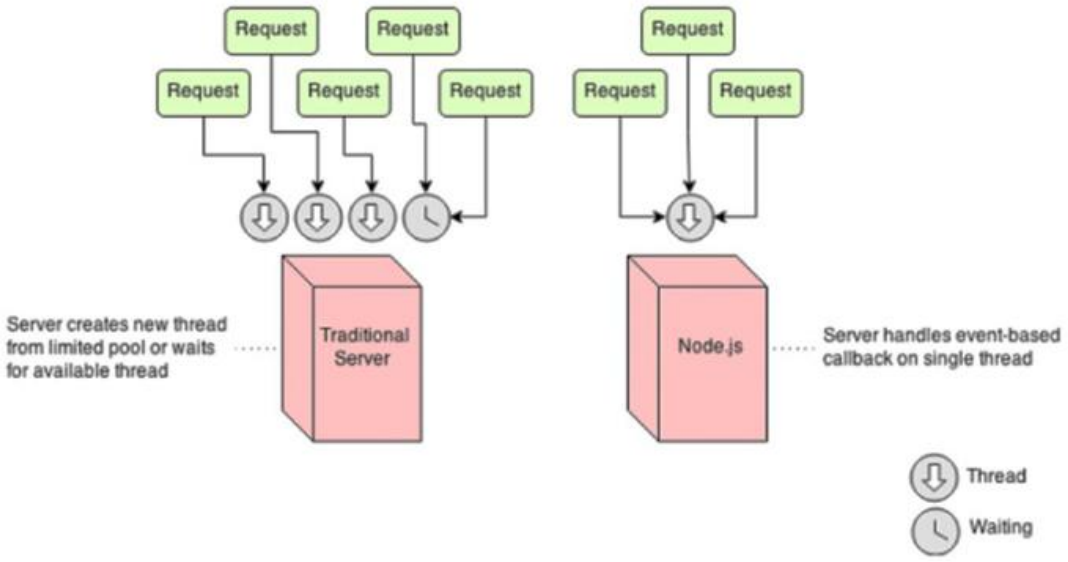
Setelah Anda menginstal Node.js, navigasikan ke terminal (untuk Mac) atau command prompt (untuk Windows) dan periksa untuk memastikan bahwa instalasi berhasil, dengan menjalankan perintah node –v. Gambar 2.10 menunjukkan hasilnya.



Gambar 2.10 Perintah Terminal Ini Mengonfirmasi Bahwa Saya Telah Menginstal Node. Tanda –V Menandakan Versi, Jadi Respons Yang Ditampilkan Adalah Versi Node Yang Telah Saya Instal.

Jika Node sudah terinstal dengan benar, Anda akan mendapatkan respons yang menunjukkan versinya. Pastikan versi Anda minimal v0.10.32. Jika belum, kembali ke halaman instalasi Node dan ambil versi yang lebih baru. Sekarang setelah Node terinstal, kita dapat mulai menggunakannya dalam demo tanpa server. Untuk menjelaskan lebih lanjut tentang

Node.js dan cara kerjanya, Gambar 2.11 menunjukkan cara kerjanya di balik layar untuk mendukung ribuan koneksi bersamaan.

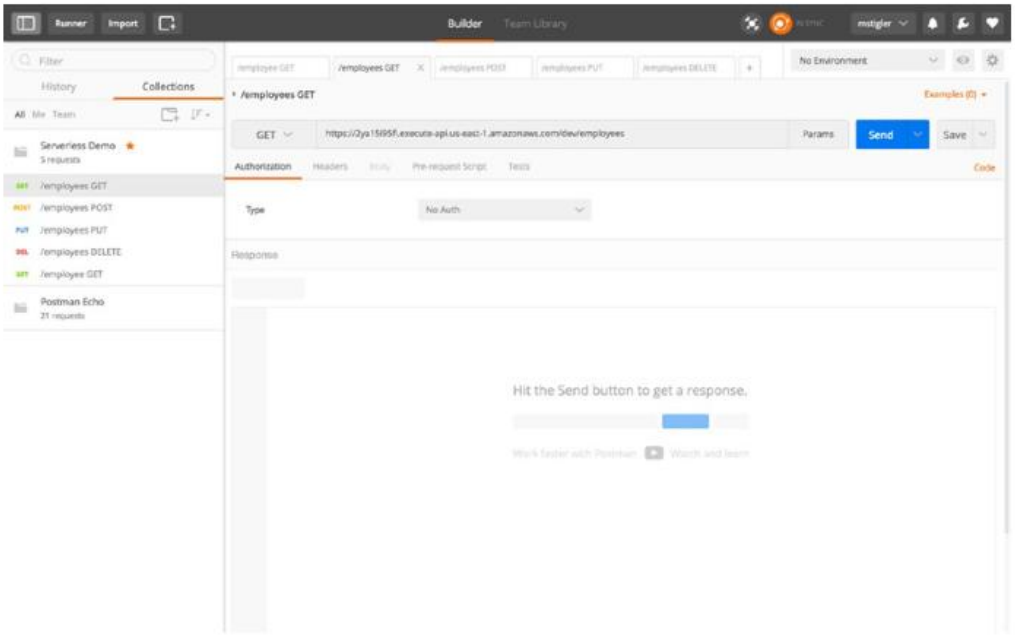


Gambar 2.11 Gambar Dari Havlena.Net Ini Mengilustrasikan Konkurensi Node.Js Yang Ditetapkan Oleh Pendekatan Single-Thread-Nya

Strukturnya yang digerakkan oleh peristiwa dan kemampuannya untuk memproses secara cepat menjadikan Node.js sebagai runtime yang ideal untuk aplikasi tanpa server, aplikasi obrolan, proksi, dan aplikasi waktu nyata. Mirip dengan fungsi tanpa server, Node tidak dimaksudkan untuk digunakan untuk operasi yang berjalan lama dan intensif CPU.

Postman

Postman adalah alat praktis lain yang akan kita gunakan di seluruh tutorial.



Gambar 2.12 Aplikasi Postman Dan Semua Yang Dapat Anda Lakukan Di Dalamnya

Postman adalah GUI lintas platform yang membuat pengujian API Anda menjadi sangat mudah. Untuk mengunduhnya, buka https://www.getpostman.com/ dan klik paket instalasi yang tepat untuk mesin Anda. Gambar 2.12 menunjukkan seperti apa tampilan aplikasi Postman setelah diinstal. Postman memungkinkan Anda menyimpan permintaan sehingga Anda dapat mengaksesnya dengan mudah saat Anda ingin mengujinya nanti.

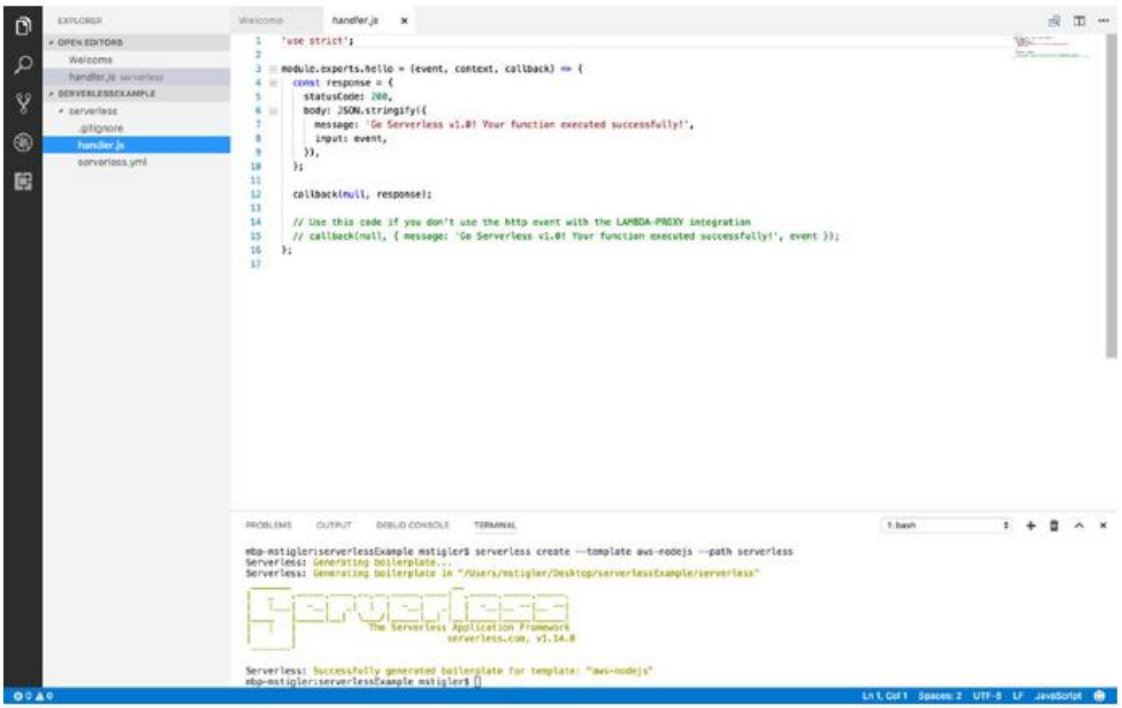
Untuk mengakses API, Anda menentukan metode, titik akhir, dan parameter apa pun, lalu klik Kirim. Ini akan memanggil API dan mengembalikan informasi apa pun yang diterimanya kembali ke Postman. Kita akan menggunakan alat ini di seluruh buku ini untuk memicu fungsi yang dijalankan oleh permintaan API. Postman akan memberi kita hasil permintaan di bagian Respons aplikasi.

2.6 PENYIAPAN LINGKUNGAN

Untuk sisa buku ini, kita akan membahas beberapa contoh fungsi menggunakan AWS, Azure, dan Google Cloud. Untuk mencapai hal ini, saya sarankan untuk menggunakan seperangkat alat agar lingkungan pengembangan sama di seluruh tutorial. Saya akan menggunakan Visual Studio Code sebagai IDE, runtime Node.js menggunakan Node Package Manager (NPM), dan Serverless Framework untuk menyebarkan fungsi kita.

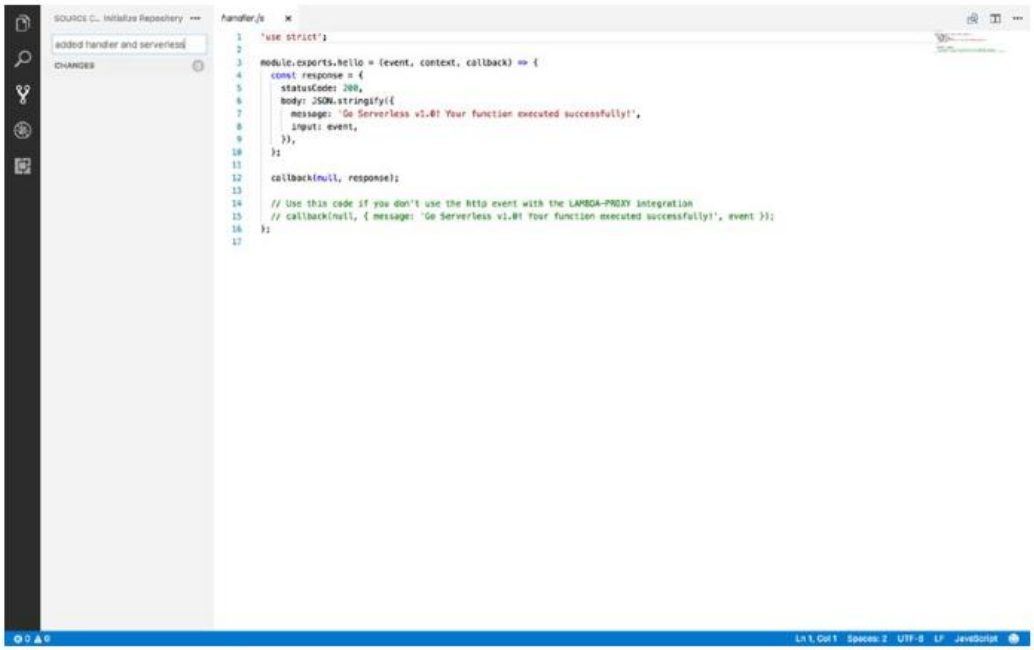
Menavigasi VS Code

Visual Studio Code adalah IDE ringan yang memiliki Git, Intellisense, Debugging, dan Extensions bawaan.



Gambar 2.13 Ilustrasikan Tampilan Explorer, Tempat Anda Dapat Menavigasi Melalui Proyek Dan Menulis Fungsi Di Ruang Kerja

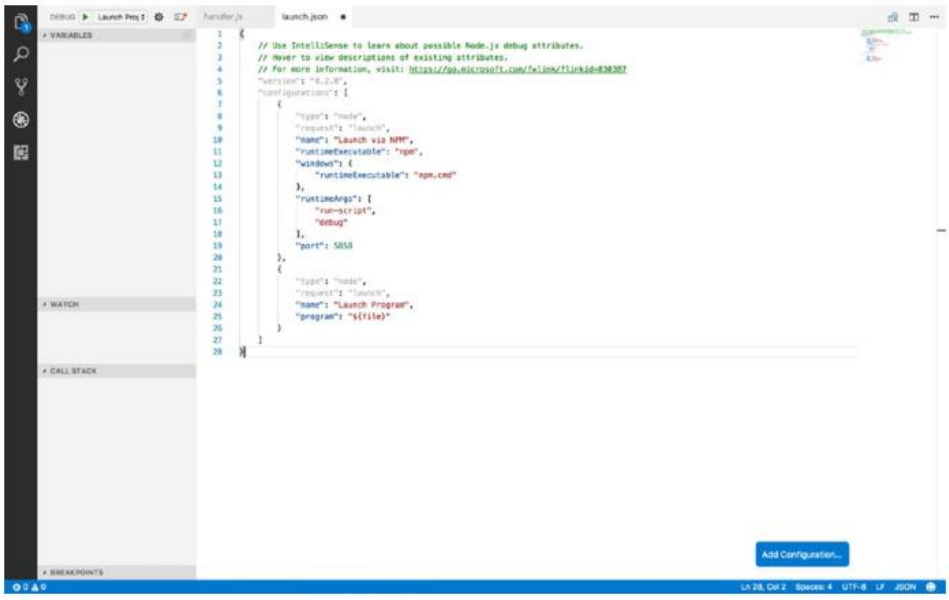
Intellisense memberi Anda lebih dari sekadar penyorotan sintaksis dan pelengkapan otomatis. Intellisense juga menyediakan pelengkapan cerdas berdasarkan jenis variabel, definisi fungsi, dan modul yang diimpor. Intellisense dapat diakses saat Anda menulis fungsi. Untuk melihat struktur proyek dan menavigasi antar-file, klik alat penjelajah di sudut kiri atas VS Code. Gambar 2.13 menunjukkan struktur proyek dasar dalam alat Penjelajah. Git bawaan memudahkan inisialisasi repo dan penggunaan semua perintah Git langsung dari Visual Studio Code. Gambar 2.14 memperlihatkan komponen Git bawaan.



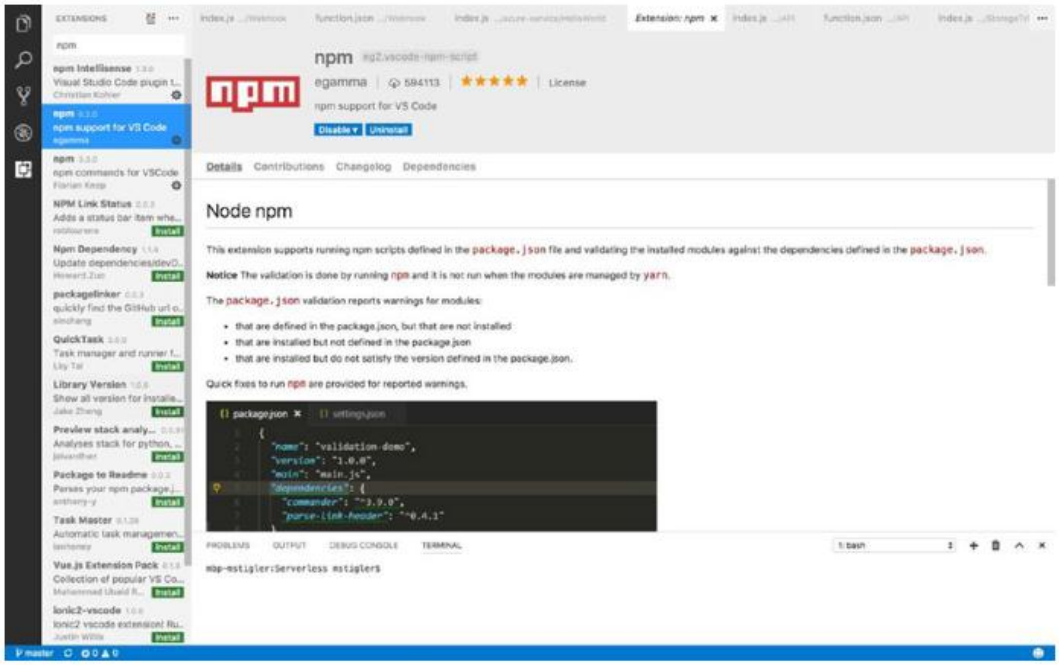
Gambar 2.14 Komponen Git Bawaan Memungkinkan Anda Untuk Menginisialisasi Repo, Melakukan Komitmen Dan Mengirim Perubahan, Serta Melacak Cabang

Bilah biru di bagian bawah VSCode memberi tahu Anda cabang mana yang Anda gunakan dan menampilkan peringatan atau kesalahan apa pun untuk cabang tersebut. Visual Studio Code juga menyediakan komponen debugging yang membantu Anda menambahkan konfigurasi ke proyek Anda untuk menelusuri dan men-debug kode Anda. Gambar 2.15 menunjukkan komponen debugging dan konfigurasi yang baru dibuat untuknya.

VSCode juga menyediakan bagian ekstensi yang memungkinkan Anda memasang dan menambahkan berbagai ekstensi ke proyek Anda. Salah satu ekstensi yang akan kita gunakan adalah ekstensi npm untuk dukungan Node Package Manager. Kita dapat melanjutkan dan memasang ekstensi ini sekarang dengan menavigasi ke komponen ekstensi dan mencari NPM di pasar. Gambar 2.16 menunjukkan cara menemukan ekstensi NPM.



Gambar 2.15 Alat Debugging Mencakup Breakpoint, Konfigurasi, Bagian Variabel Yang Dapat Anda Lacak, Tumpukan Panggilan, Dan Log Pengawasan



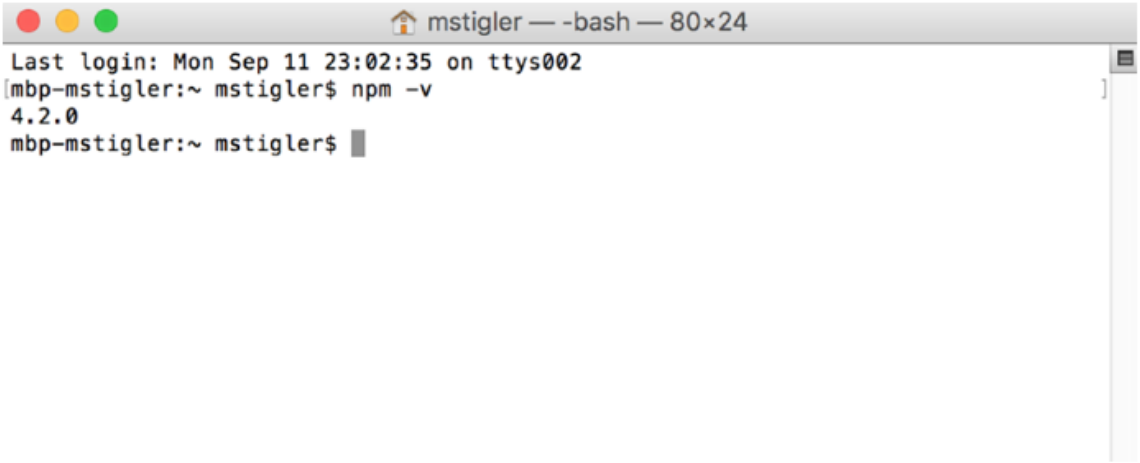
Gambar 2.16 Ekstensi NPM Yang Ingin Kita Instal Untuk Menyiapkan Lingkungan Kita Dapat Ditemukan Di Marketplace. Anda Dapat Menginstalnya Dengan Mengeklik Opsi Instal

Ekstensi akan memberi Anda detail, catatan perubahan, dan dependensi beserta contoh cara penggunaannya. Untuk tujuan tutorial berikut, kita akan melanjutkan dan menginstal ekstensi NPM yang memungkinkan Anda menjalankan perintah dalam kode serta ekstensi yang menyediakan dukungan untuk kode VS.

2.7 NODE PACKAGE MANAGER: APA FUNGSINYA DAN CARA MENGGUNAKANNYA

Node Package Manager adalah manajer paket untuk JavaScript, jadi kita akan menggunakannya dengan aplikasi Node.js kita untuk menginstal berbagai paket yang diperlukan untuk menyelesaikan proyek kita. NPM sangat kolaboratif, memungkinkan Anda untuk menginstal, berbagi, dan mendistribusikan kode. NPM mencakup paket-paket seperti gulp, grunt, bower, async, lodash, dan request.

NPM diinstal bersama dengan Node, jadi kita dapat melanjutkan dan menggunakan terminal/command prompt kita untuk melihat apakah kita memang telah menginstal NPM. Gunakan perintah npm –version untuk melihat apakah Anda telah menginstal npm. Gambar 2.17 menunjukkan respons dari terminal yang menunjukkan versi npm yang telah saya instal.



Gambar 2.17 Versi Npm Mengembalikan 4.2.0, Yang Menunjukkan Bahwa Saya Telah Menginstal NPM

Serverless Framework

Seperti yang dibahas dalam Bab 1, Serverless Framework adalah alat pengembangan dan penyebaran yang hebat. Dengan rilis baru integrasinya dengan Google Cloud Functions, ia menjadi bagian integral dari pengembangan fungsi cloud. Untuk menginstal Serverless, kita akan menggunakan NPM. Di terminal/command prompt Anda, masukkan perintah berikut:

```
npm install -g serverless
```

Ini akan menginstal Serverless secara global di komputer Anda. Jadi, ketika kita membuat berbagai proyek fungsi nanti, kita dapat dengan cepat dan mudah membuat layanan baru dan menyebarkannya menggunakan Serverless. Untuk informasi lebih lanjut sebelum itu, https://serverless.com/framework/docs/ menyediakan dokumentasi untuk Serverless untuk setiap penyedia cloud yang kami liput.

Mengatur Lingkungan Pengembangan Anda

Ada banyak cara untuk mengatur lingkungan pengembangan kita; Namun, karena kita akan mengembangkan aplikasi tanpa server dengan tiga penyedia cloud yang berbeda, maka

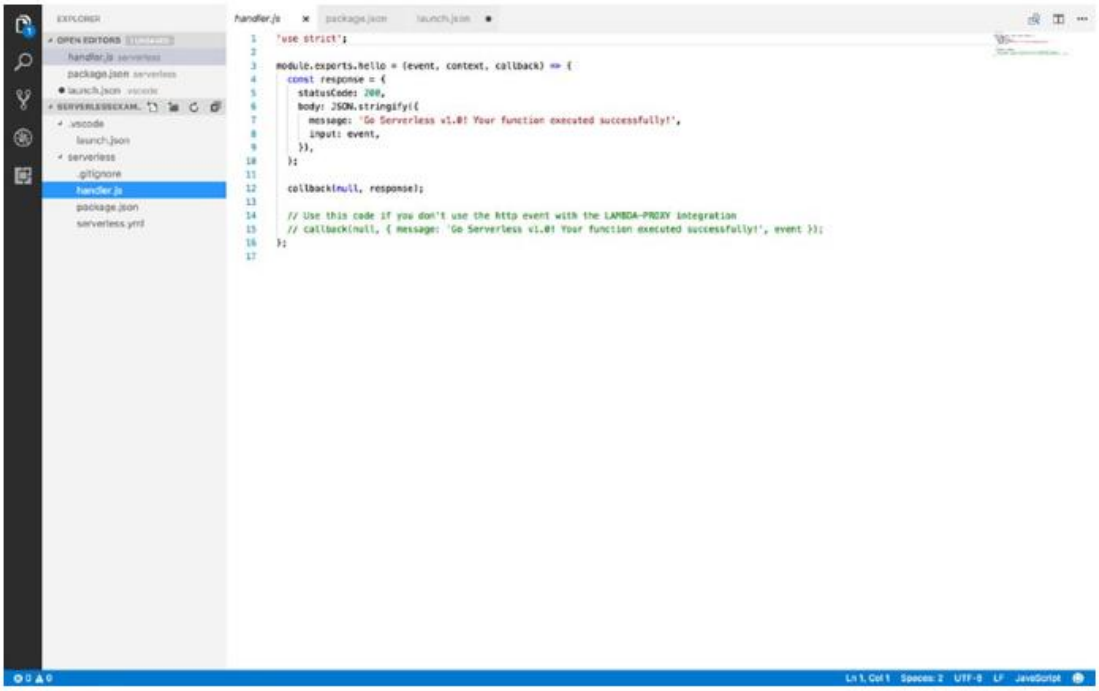
paling masuk akal untuk mengorganisasikannya berdasarkan penyedia dan kemudian menunjukkan cara mengembangkan proyek yang tidak bergantung pada penyedia. Untuk memulai, saya sarankan untuk menyiapkan repositori Git atau semacam kontrol versi.

Anda bisa mendapatkan akun gratis di GitHub untuk menyimpan kode Anda dengan membuka http://www.github.com. Saya membuat repositori bernama Serverless dan kemudian membuat tiga proyek di dalamnya (AWS, Azure, dan Google). Untuk setiap proyek, saya menginisialisasi proyek kerangka kerja tanpa server di dalamnya. Di dalam folder proyek AWS Anda, jalankan perintah:

```
serverless create --template aws-nodejs --path aws-service.
```

Template menentukan penyedia cloud yang Anda gunakan beserta runtime-nya. Jalur menentukan nama layanan yang Anda buat. Saya memilih untuk memberi nama layanan saya aws-service. Setelah membuat layanan, kita perlu menginstal dependensi proyek dalam layanan tersebut. Navigasi dalam layanan dan jalankan perintah berikut:

```
mpm install
```



Gambar 2.18 Serverless Framework Membuat File Package.Json, Contoh File Handler.Js, Dan File Serverless.Yml Saat Diinstal

Seperti yang telah kita lihat sebelumnya, Node Package Manager akan membaca file package.json yang disediakan oleh Serverless dan akan menginstal semua dependensi yang tercantum. Gambar 2.18 menunjukkan struktur proyek yang diberikan Serverless Framework

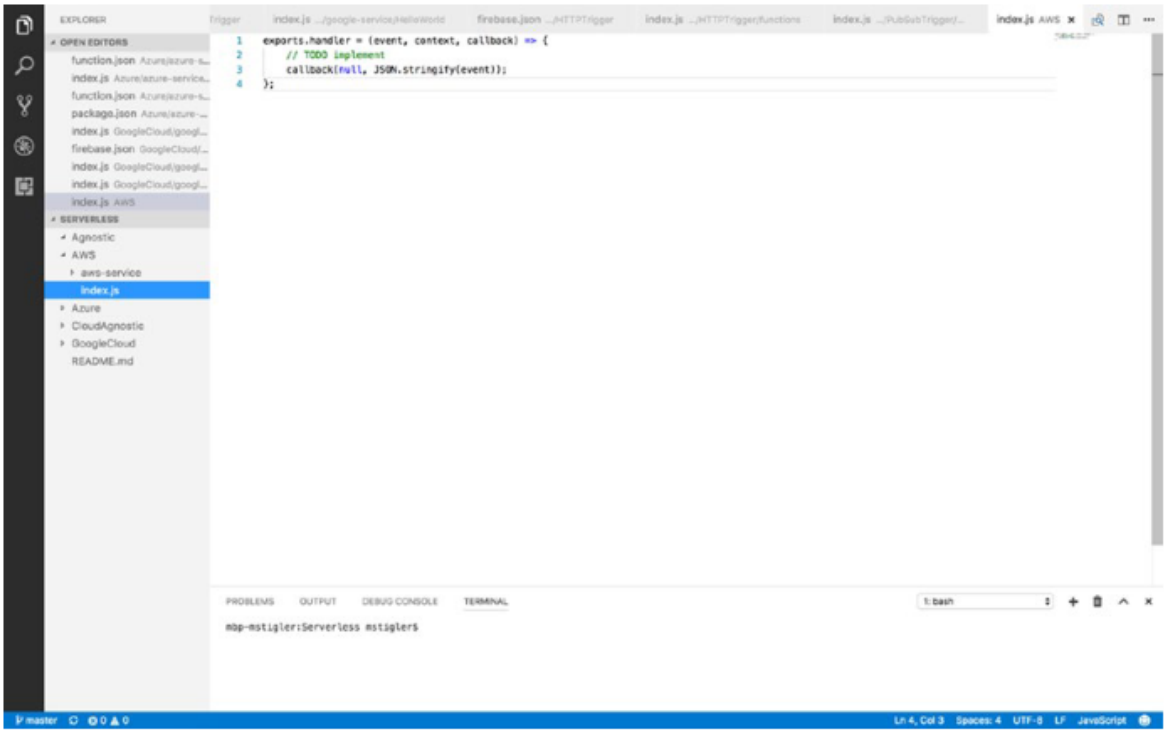
secara langsung. Di dalam repositori yang sama, kita juga akan menginstal Serverless di proyek Azure dan Google. Untuk Azure, kita masukkan perintah:

```
serverless create -- template azure-nodejs -- path azure-service.
npm install
```

Ini menghasilkan hal yang sama seperti yang kita lakukan dengan AWS. Jika Anda membuka proyek Azure di Visual Studio Code, Anda akan melihat struktur proyek yang sama (handler.js, serverless.yml, package.json, dan node_modules). Kita akan terus melakukan hal yang sama dengan proyek Google.

```
serverless create -- template google-nodejs -- path google-service.
npm install
```

Gambar 2.19 menunjukkan kerangka proyek yang telah selesai dengan masing-masing dari tiga penyedia cloud. Kami akan menggunakan file serverless.yml dalam tiga bab berikutnya untuk menerapkan fungsi serverless kami dalam lingkungan masing-masing penyedia.



Gambar 2.19 Gambar ini menunjukkan struktur proyek dasar dalam masing-masing dari tiga penyedia. Struktur proyek ini juga dapat dikloning dari https://github.com/mgstigler/Serverless.git.

2.8 KESIMPULAN

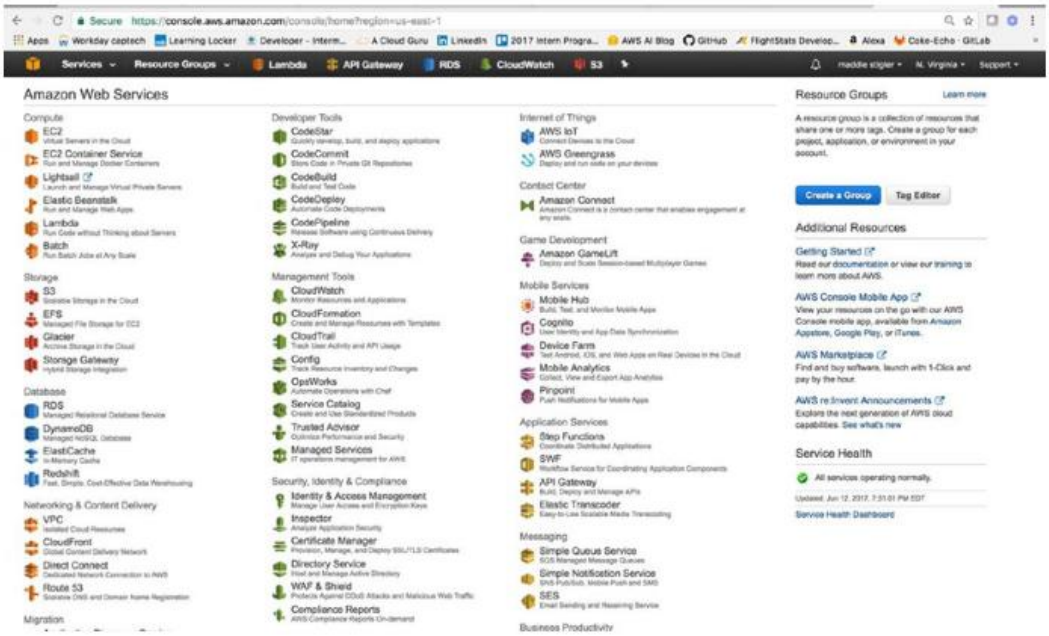
Dalam bab ini, kami membahas semua yang perlu Anda ketahui untuk mulai membangun fungsi tanpa server. Kami telah menyiapkan alat dan lingkungan, serta pengetahuan tentang bagaimana pemicu, peristiwa, dan fungsi bersatu untuk menghasilkan aplikasi berbasis peristiwa yang dapat diskalakan. Dalam bab berikutnya, kami akan membahas beberapa contoh membangun fungsi menggunakan AWS Lambda.

BAB 3
AMAZON WEB SERVICES

Dalam bab ini, kita akan memanfaatkan Amazon Web Services (AWS) untuk membuat beberapa aplikasi tanpa server. Kita akan menggunakan Lambda untuk membuat fungsi Hello World yang dipicu oleh kasus uji yang akan kita buat. Kita juga akan menggunakan sumber daya dan metode API Gateway untuk memicu fungsi Lambda dengan permintaan HTTP RESTful yang akan mengembalikan data dari DynamoDB. Terakhir, kita akan menjelajahi aplikasi penyimpanan yang dipicu, tempat kita dapat mengubah ukuran gambar yang diunggah ke layanan S3 Amazon menggunakan fungsi Lambda. Di akhir bab ini, kita akan memiliki tiga aplikasi tanpa server dan pengalaman dengan beberapa layanan AWS. DynamoDB adalah Layanan Basis Data Cloud NoSQL Amazon. Kita akan menggunakan ini dan S3 (Layanan Penyimpanan Sederhana, penyimpanan blob Amazon) untuk mendapatkan pengalaman dengan dua opsi penyimpanan yang berbeda dan tempatnya dalam aplikasi tanpa server.

3.1 JELAJAHI UI

Sebelum kita mulai menulis aplikasi kita, kita akan membahas UI AWS dan cara menavigasinya, berbagai opsi harga, dan portal Lambda. Setelah masuk ke konsol di http://www.console.aws.amazon.com, beranda Anda akan menampilkan daftar Layanan AWS. Gambar 3.1 memberi Anda gambaran tentang semua layanan yang disediakan AWS.



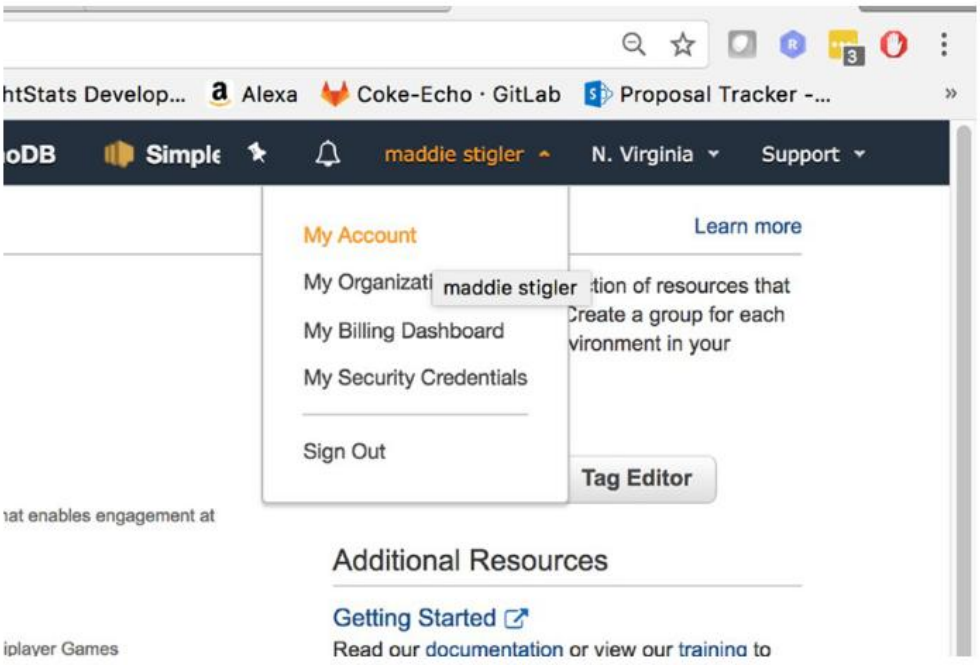
Gambar 3.1 AWS Menyediakan Beberapa Layanan Yang Mencakup Berbagai Topik Dan Kemampuan, Termasuk Layanan Komputasi, Penyimpanan, Basis Data, Dan Aplikasi

Kita dapat mengaksesnya di bawah tab Layanan di bagian atas. Kita juga dapat menyematkan layanan favorit dengan mengeklik pin di bilah navigasi dan menambahkan layanan ke dalamnya. Saya telah menyematkan layanan yang paling sering saya gunakan untuk memudahkan akses.

Navigasi

Selain mengakses Layanan AWS, kita juga memiliki akses ke grup sumber daya AWS dari portal. Grup sumber daya adalah kumpulan sumber daya yang berbagi satu atau beberapa tag. Grup sumber daya bagus untuk menjaga layanan proyek tetap terpisah dan teratur. Grup sumber daya dapat dengan mudah dibuat dari tab Grup Sumber Daya di bagian atas. Anda juga dapat membuat grup sumber daya dari panel di sebelah kanan. Tab Layanan akan memungkinkan Anda untuk mencari dan mengurutkan layanan yang Anda lihat di beranda. AWS juga menyediakan Sumber Daya Tambahan untuk membantu Anda memulai menggunakan AWS, dan pemeriksaan Kesehatan Layanan di panel kanan yang memungkinkan Anda melihat kesehatan layanan Anda saat masuk.

Lonceng di spanduk atas memberi Anda peringatan yang diberikan oleh AWS. Peringatan ini memberi Anda masalah yang belum terpecahkan, perubahan terjadwal di masa mendatang, dan pemberitahuan lainnya. Di bawah nama pengguna, Anda dapat mengakses akun, organisasi, dasbor penagihan, organisasi, dan keluar. Gambar 3.2 menunjukkan opsi ini dan tempat untuk mengaksesnya. Kita akan membahas dasbor penagihan lebih lanjut di bagian Harga bab ini. AWS Organizations memungkinkan Anda menerapkan kontrol berbasis kebijakan secara terpusat di beberapa akun di AWS Cloud. Anda dapat menggabungkan semua akun AWS ke dalam satu organisasi, dan mengatur semua akun AWS ke dalam unit organisasi yang berbeda.



Gambar 3.2 Nama Pengguna Dan Informasi Akun Semuanya Dapat Diakses Langsung Dari Beranda Portal

Di sinilah Anda dapat mengelola semua pengaturan akun dan informasi penagihan. Wilayah di sebelah kanan nama pengguna Anda memungkinkan Anda memilih Wilayah. Amazon tempat Anda bekerja. Nama Wilayah adalah sebagai berikut:

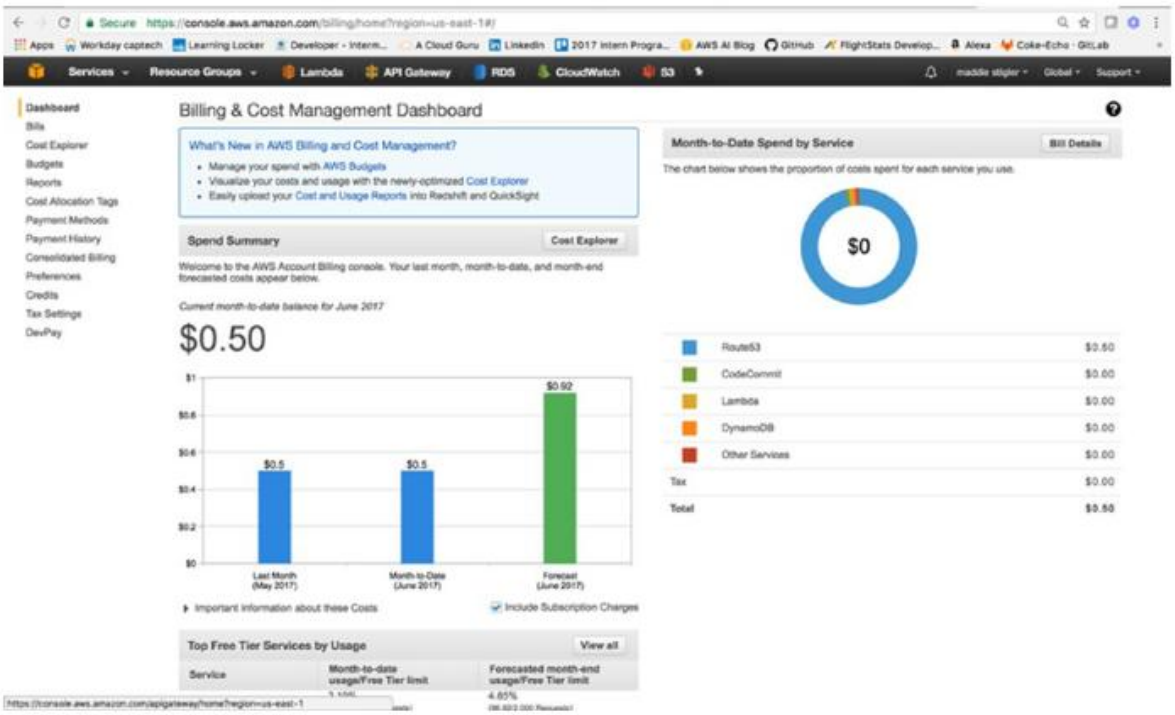
- AS Timur (Ohio)
- AS Timur (Virginia Utara)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Kanada (Tengah)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Tokyo)
- UE (Frankfort)
- UE (Irlandia)
- UE (London)
- Amerika Selatan (Sao Paulo)

Wilayah yang Anda pilih menjadi wilayah default Anda di konsol. Beberapa layanan tersedia di wilayah yang tidak tersedia di wilayah lain. Jadi, jika Anda menjelajahi konsol dan tidak melihat sumber daya yang Anda cari, coba pilih wilayah tempat sumber daya tersebut dibuat. AS-Timur adalah wilayah default, jadi sebaiknya periksa di sana terlebih dahulu.

AWS juga menyediakan layanan Dukungan yang menawarkan Pusat Dukungan, Forum, Dokumentasi, Pelatihan, dan Sumber Daya Lainnya. Ini bagus untuk mempelajari layanan baru dan mendapatkan pengalaman langsung dengan layanan baru. Dokumentasi menyediakan panduan Memulai dengan contoh, SDK dan Alat, sumber daya, dan contoh untuk layanan AWS. Ini sangat membantu saat memulai dengan layanan baru.

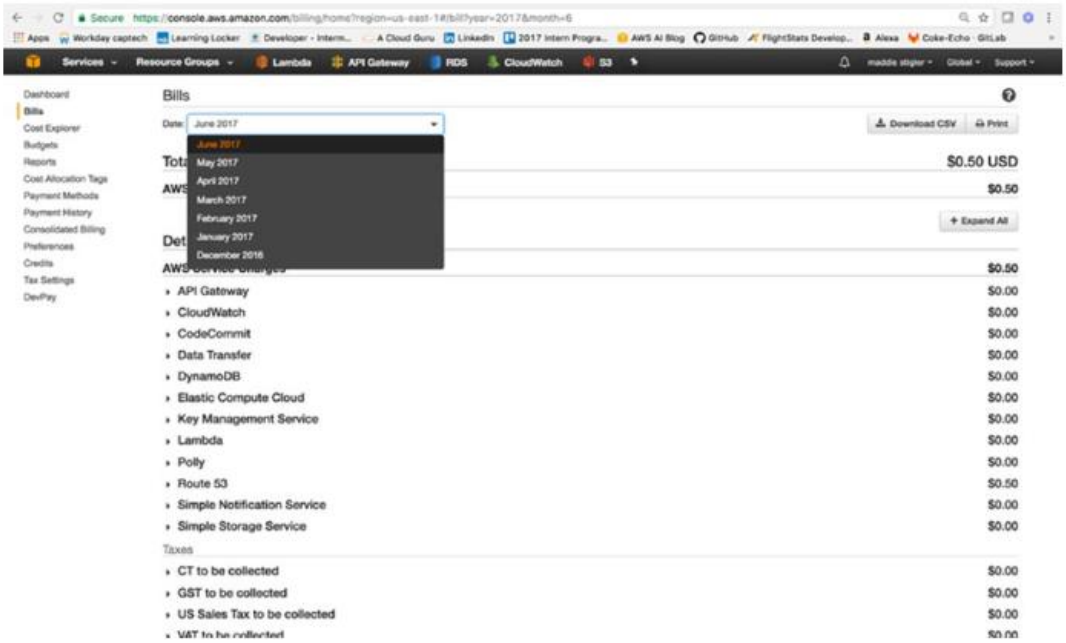
Harga

Pada Bab 1, Tabel 1.1 menunjukkan harga Lambda untuk AWS dibandingkan dengan Azure dan Google Cloud, tetapi sekarang Anda akan melihat cara menavigasi Konsol Manajemen Penagihan di bawah akun Anda untuk membantu mengelola biaya Anda. Di bawah nama pengguna Anda, pilih Dasbor Manajemen Penagihan dan Biaya. Ini akan membawa Anda ke halaman yang terlihat mirip dengan yang ditunjukkan pada Gambar 3.3. AWS memberi Anda banyak aksesibilitas dengan mengelola biaya layanan, metode pembayaran, membuat laporan, dan melihat tagihan Anda. Semua kemampuan ini tersedia melalui dasbor Penagihan di panel kiri.



Gambar 3.3 Dasbor Penagihan Dan Manajemen Biaya Memberi Anda Ikhtisar Biaya Per Bulan, Per Layanan, Dan Sebagai Perkiraan

Opsi Penagihan memungkinkan Anda menyortir tagihan AWS Anda berdasarkan bulan. Opsi ini menyediakan riwayat tagihan ini serta opsi ekspor CSV. Gambar 3.4 menunjukkan kemampuan ini.



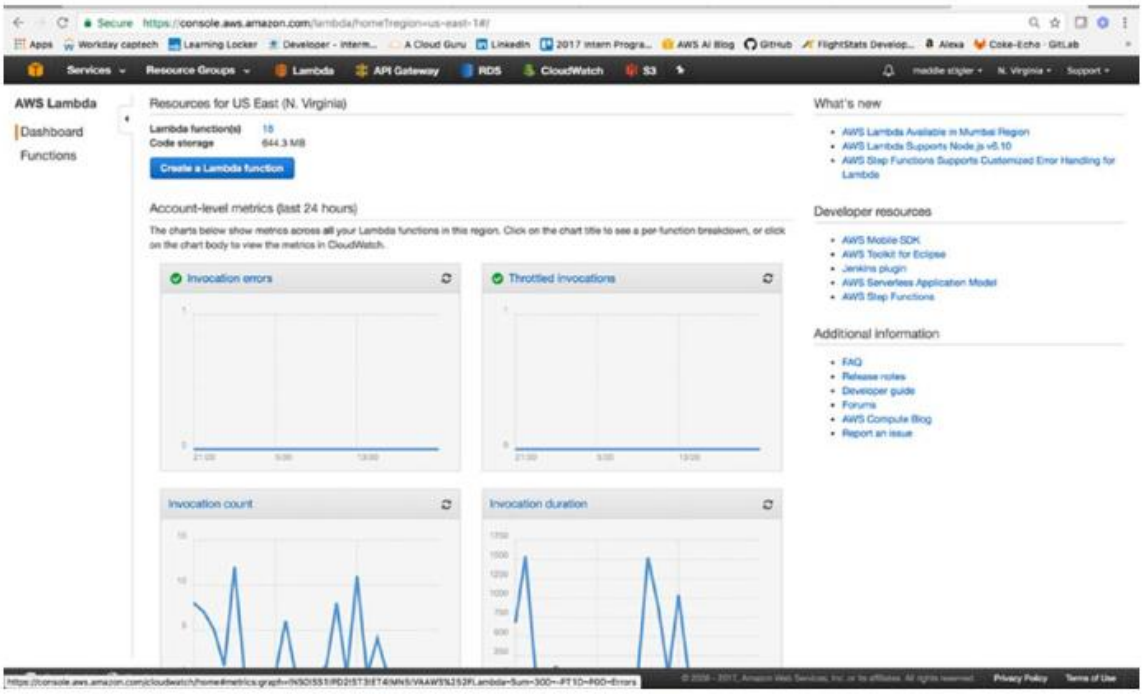
Gambar 3.4 Bagian Tagihan Memungkinkan Anda Menyortir Tagihan Dan Melihatnya Berdasarkan Tanggal

Kemampuan penagihan penting lainnya adalah Penagihan Terkonsolidasi, yang memungkinkan Anda menangani beberapa akun di bawah satu akun induk. Kemampuan ini bekerja dengan AWS Organizations untuk membuat beberapa organisasi, mengatur akun di organisasi tersebut, dan menerapkan kebijakan ke organisasi tersebut. Contoh kasus penggunaan yang baik untuk ini adalah perusahaan besar dengan beberapa proyek. Daripada menggunakan grup dan tag sumber daya, Anda dapat menjaga aplikasi dan sumber daya AWS Anda sepenuhnya terpisah satu sama lain dengan Organizations dan Penagihan Terkonsolidasi.

Solusi lainnya adalah menyiapkan peringatan penagihan dalam konsol penagihan. Peringatan penagihan akan mengirimkan pemberitahuan email kepada Anda saat akun Anda mencapai jumlah dolar atau jumlah sumber daya tertentu yang Anda tetapkan. Saya membuat kesalahan dengan berpikir semua layanan di tingkat gratis itu gratis dan dikenai tagihan yang cukup besar setelah menjalankan beberapa instans RDS dan instans EC2. Sejak saat itu, saya telah menyetel peringatan penagihan untuk memberi tahu saya saat saya melampaui $1.

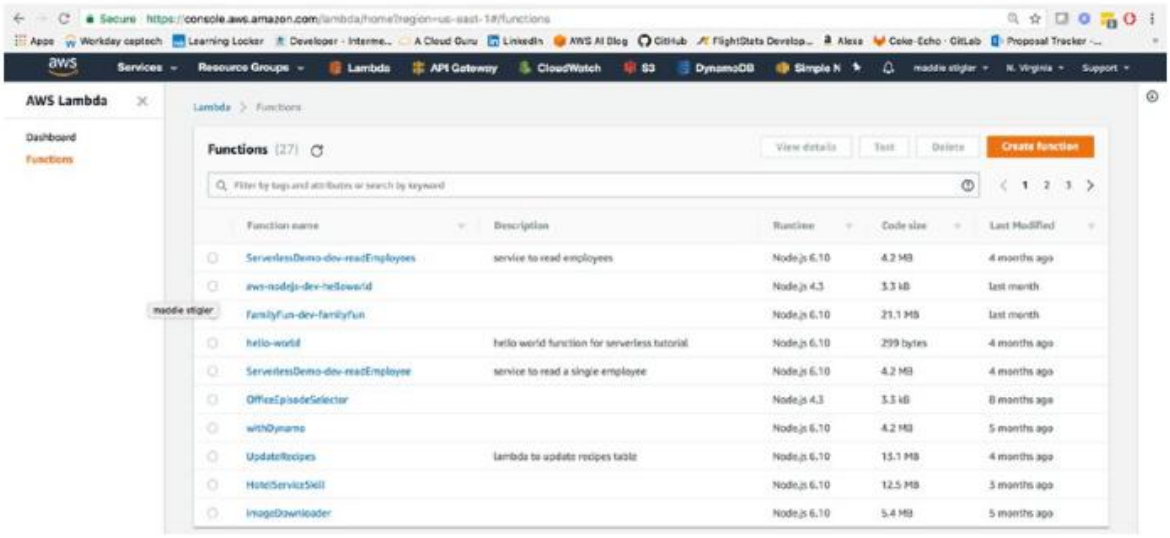
Lambda

Konsol Lambda dapat diakses dari tab Layanan di bagian atas, di bawah Komputasi. Dasbor Lambda memungkinkan Anda melihat jumlah fungsi, total penyimpanan kode, metrik tingkat akun selama 24 jam terakhir, dan apa yang baru dalam Lambda. Gambar 3.5 memberi Anda gambaran umum tentang dasbor Lambda.



Gambar 3.5 Dasbor Lambda Memberi Anda Ikhtisar Semua Fungsi Anda

Bagian Fungsi adalah tempat Anda mengakses dan membuat fungsi Lambda. Gambar 3.6 menunjukkan UI Fungsi.



Gambar 3.6 Layar Fungsi Lambda Menunjukkan Semua Fungsi Anda Secara Umum. Anda Dapat Mengeklik Suatu Fungsi Untuk Mengakses Kode, Peristiwa Pengujian, Dan Konfigurasi.

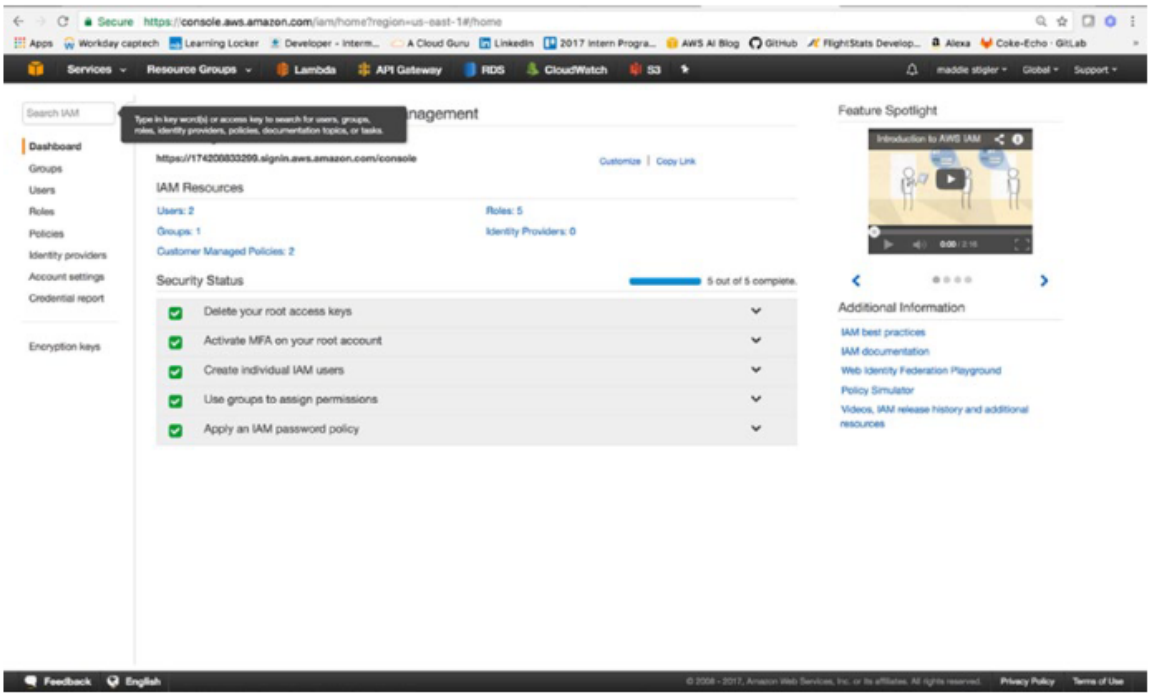
Kita akan menjelajahi portal Lambda dan semua kemampuannya secara lebih mendetail saat kita mulai membuat fungsi Lambda. Namun, sebelum kita dapat membuat fungsi, kita perlu mengonfigurasi IAM (Identity and Access Management) kita. Penting untuk dicatat bahwa menyiapkan layanan IAM tidak diperlukan untuk menggunakan fungsi, tetapi kita mengikuti praktik terbaik dan demi kepentingan terbaik Anda untuk melanjutkan dan mengikuti beberapa langkah berikutnya. Kita akan membahas contoh yang lebih canggih nanti yang akan memerlukan penggunaan IAM untuk menggunakan berbagai layanan.

3.2 KEAMANAN IAM

Layanan IAM adalah layanan yang sangat penting dan terintegrasi dalam AWS. IAM memungkinkan Anda menetapkan pengguna, peran, dan kebijakan untuk membantu mengamankan sumber daya Amazon Anda.

Konsol IAM

Konsol IAM ditemukan di bawah Layanan dan Keamanan, Identitas, dan Kepatuhan. Konsol memberi Anda dasbor, grup, pengguna, peran, kebijakan, penyedia identitas, pengaturan akun, laporan kredensial, dan kunci enkripsi. Dari dasbor (Gambar 3.7), Anda dapat mengakses tautan masuk pengguna. Di sinilah pengguna yang bukan pengguna admin diarahkan untuk masuk. Di sini juga diberikan ikhtisar tentang Sumber Daya IAM Anda, termasuk jumlah pengguna, grup, kebijakan yang dikelola pelanggan, peran, dan penyedia identitas.



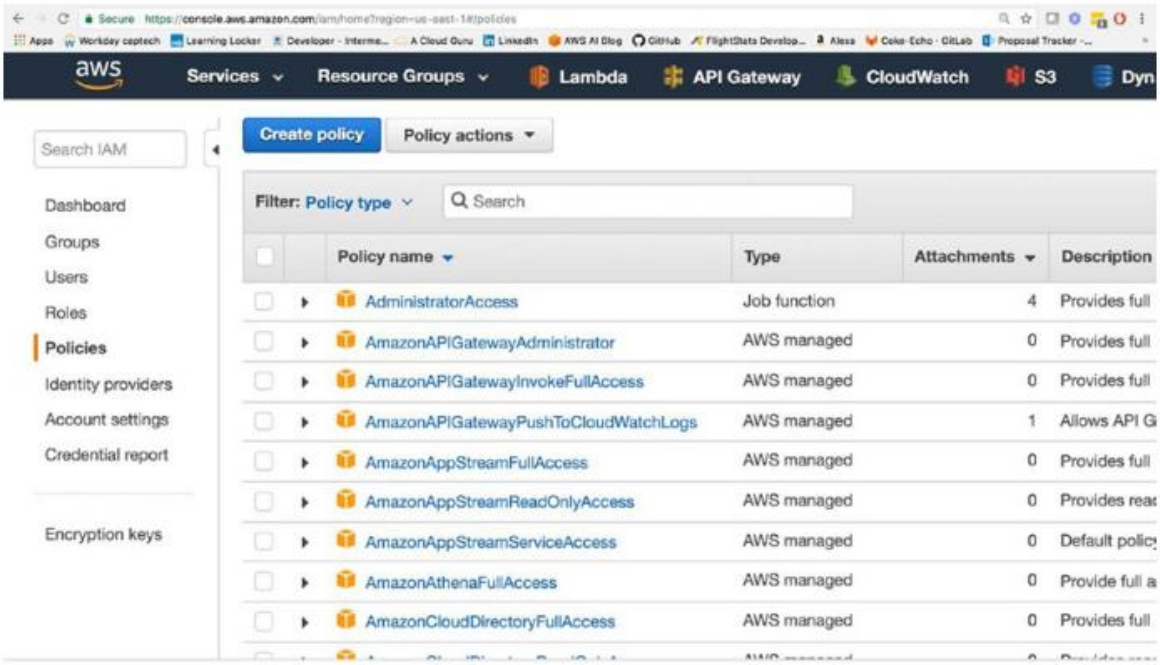
Gambar 3.7 Dasbor IAM Memberi Anda Gambaran Umum Tentang Semua Sumber Daya IAM Anda Beserta Lima Langkah Keamanan Yang Disarankan Untuk Diselesaikan

Untuk memulai, penting untuk menyelesaikan lima langkah yang tercantum di konsol Status Keamanan: menghapus kunci akses root, mengaktifkan MFA (autentikasi multifaktor) di akun root, membuat pengguna IAM individual, membuat grup untuk izin, dan menerapkan kebijakan kata sandi IAM. Dengan mengikuti langkah-langkah ini, Anda memastikan bahwa pengaturan IAM Anda diamankan dengan benar sehingga Anda dapat mulai membuat pengguna dan peran.

Peran, Kebijakan, dan Pengguna

Peran, Kebijakan, dan Pengguna adalah cara Anda untuk menetapkan izin bagi orang, layanan, dan sumber daya. Peran dibuat di bawah tab Peran dan memungkinkan Anda membuat peran dengan kebijakan yang ditetapkan. Peran ini dapat ditetapkan bagi pengguna dan layanan. Misalnya, jika saya memiliki sekelompok pengembang yang ingin saya edit dan akses Lambda dan layanannya, tetapi bukan informasi akun root, saya dapat membuat peran yang disebut Pengembang.

Setelah peran dibuat, saya dapat menetapkan kebijakan tertentu untuknya. Kebijakan menentukan jumlah akses yang dimiliki peran terhadap layanan. Gambar 3.8 menunjukkan konsol Kebijakan dengan semua kebijakan yang telah dikonfigurasi sebelumnya. Anda juga memiliki pilihan untuk membuatnya sendiri.



Gambar 3-8. Bagian Kebijakan Memungkinkan Anda Membuat Dan Menetapkan Kebijakan. Lampiran Menjelaskan Entitas (Pengguna Dan Layanan) Yang Terkait Dengan Kebijakan.

Kebijakan menjelaskan jumlah akses yang diizinkan ke layanan tertentu. Misalnya, kebijakan AdminstratorAccess memberi Anda akses penuh ke semua sumber daya untuk semua layanan. Jendela Pengguna memungkinkan Anda menambahkan pengguna ke AWS. Mereka diberi login mereka sendiri dan peran serta kebijakan apa pun yang Anda lampirkan kepada mereka. Untuk mengakses konsol, pengguna diberi Kunci Akses dan kata sandi yang diunduh melalui CSV atau dikirim langsung kepada mereka. Anda menentukan jumlah waktu yang mereka miliki dengan kata sandi default dan semua kebijakan kata sandi terkait login pengguna mereka.

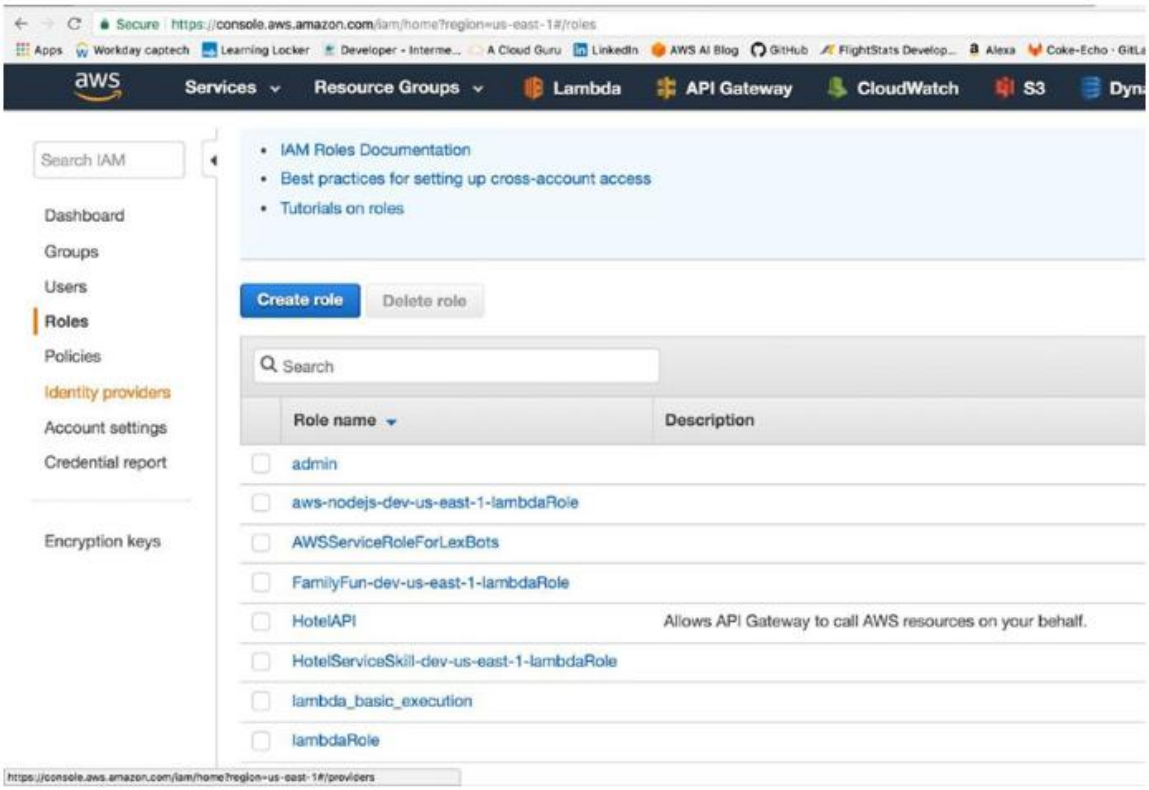
Anda juga memiliki kemampuan untuk menambahkan pengguna ke grup. Grup dapat digunakan untuk mempermudah pemberian izin. Jika Anda memiliki sekelompok pengguna yang ingin Anda berikan akses admin, Anda dapat menambahkan mereka ke grup sehingga semua kebijakan grup diterapkan secara menyeluruh. Untuk tujuan aplikasi tanpa server, kami tidak akan menetapkan pengguna atau grup, tetapi sebaiknya Anda mengingat peluang ini saat Anda membangun aplikasi yang lebih besar dengan grup yang lebih besar.

Peran untuk Lambda

AWS mengharuskan Anda menetapkan peran ke fungsi Lambda Anda. Peran ini dapat berbeda di seluruh fungsi Lambda karena memerlukan akses ke berbagai layanan AWS. Namun, untuk memulai fungsi Hello World, kami akan membuat peran AWS Lambda yang dapat ditetapkan ke fungsi kami. Di tab Peran, kami akan mengeklik opsi Buat Peran Baru. Kami akan memberi nama peran kami “lambda_basic_ execution”.

Di bawah izin, kami akan melampirkan kebijakan AWSLambdaExecute. Jika Anda melihat kebijakan ini, Anda dapat melihat izin persis yang dilampirkan padanya. Kebijakan ini

memungkinkan akses penuh ke CloudWatch untuk mencatat fungsi kami, dan menyediakan akses baca/tulis ke AWS S3. Gambar 3.9 menunjukkan seperti apa seharusnya peran tersebut setelah dibuat.



Gambar 3.9 Peran Tersebut Memiliki Kebijakan Awslambdaexecute Yang Dilampirkan Padanya

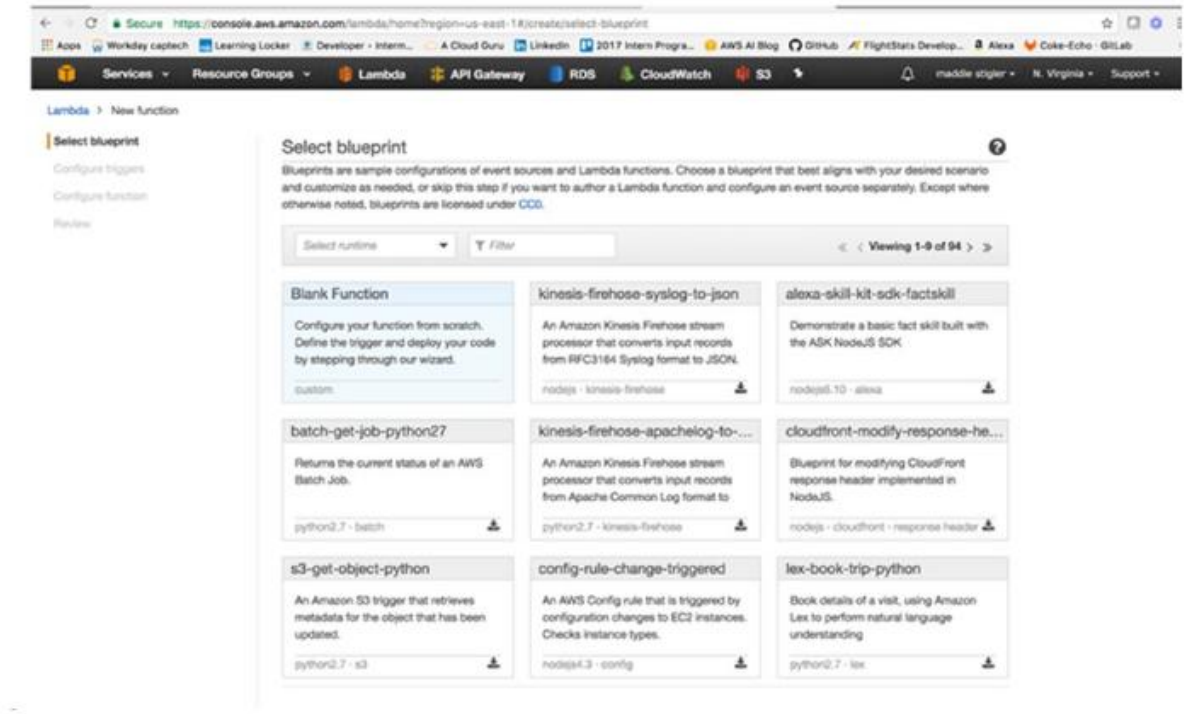
Anda dapat melihat kebijakan tersebut untuk melihat izin yang dilampirkan, dan Anda juga dapat melampirkan lebih banyak kebijakan jika diperlukan. ARN Peran di bagian atas konsol adalah Nama Sumber Daya Amazon. Inilah yang secara unik mengidentifikasi peran yang baru saja kita buat. Saat kita membuat fungsi pertama kita, kita akan menetapkan Lambda kita ke peran ini, memberinya semua izin yang ditentukan dalam satu kebijakan yang dilampirkan.

Kode Pertama Anda

Sekarang setelah peran IAM kita ditetapkan dan kita sudah memahami cara menavigasi konsol AWS, kita dapat mulai menulis kode pertama kita. Fungsi Hello World Lambda ini akan memberi kita pengalaman membuat fungsi Lambda, menetapkan peran padanya, membuat peristiwa pengujian, menjalankannya, dan kemudian melihat log di CloudWatch.

Hello World

Kita akan mulai dengan membuat fungsi baru di konsol Lambda. Setelah mengklik Buat Fungsi Lambda, Anda akan melihat daftar opsi cetak biru (Gambar 3.10). Cetak biru ini memberi Anda kerangka Lambda yang dapat Anda edit untuk melengkapi fungsionalitas yang Anda cari. Untuk memulai, kita akan memilih fungsi kosong.

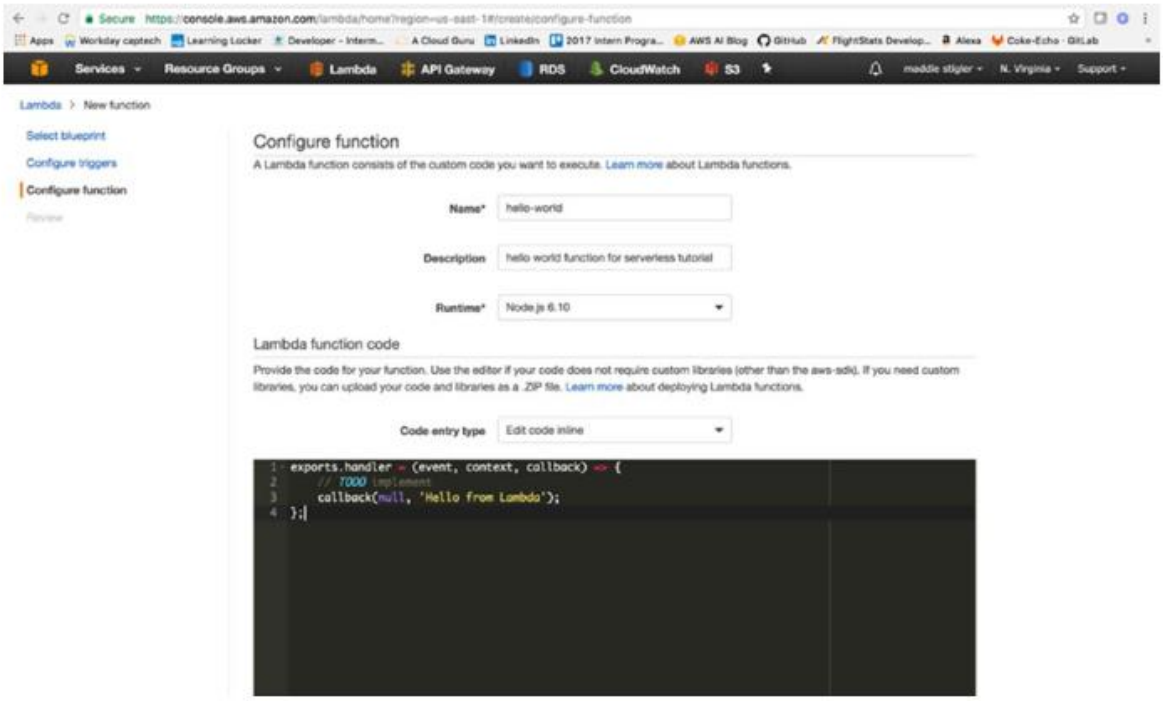


Gambar 3.10 AWS Menyediakan Banyak Opsi Cetak Biru Untuk Berbagai Runtime Dan Pemicu. Opsi Ini Bagus Untuk Dijelajahi Jika Anda Baru Mengenal Runtime Atau Layanan Tertentu.

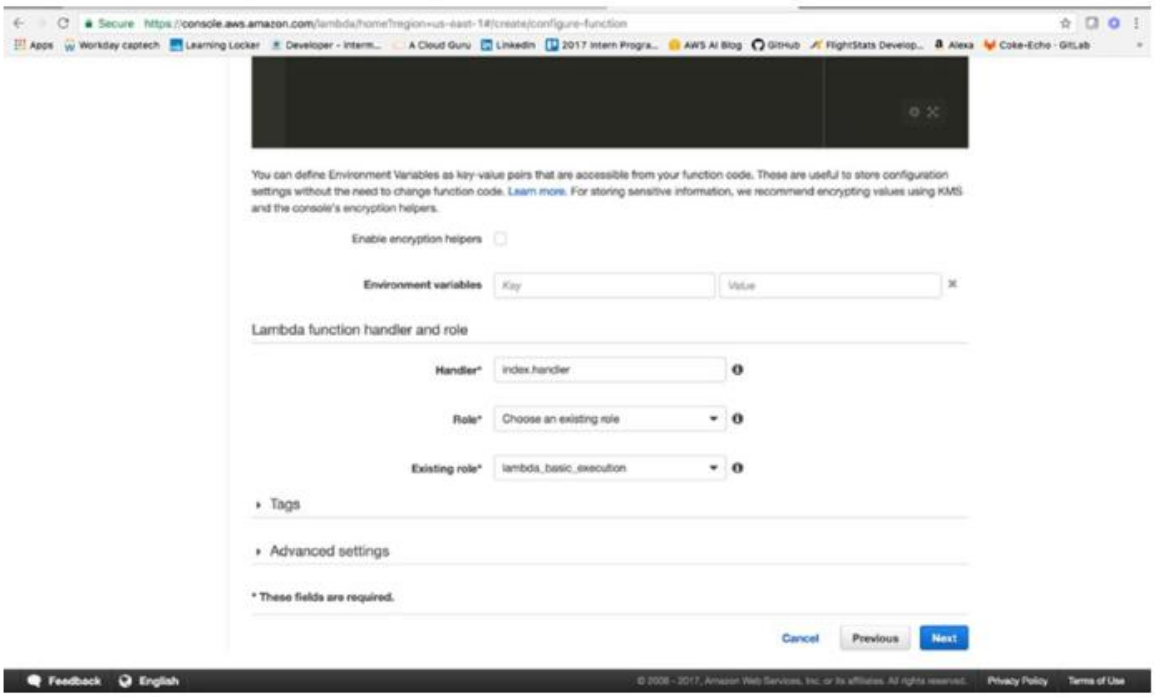
Cetak biru akan berubah berdasarkan bahasa yang Anda pilih. Misalnya, apa yang Anda lihat untuk fungsi Node.js akan berbeda dari apa yang Anda lihat untuk fungsi C#. Setelah memilih fungsi kosong, selanjutnya kita harus mengonfigurasi fungsi kita. Ini memerlukan pemberian nama, deskripsi, runtime, dan handler serta role. Nama tidak harus unik secara universal, cukup dalam fungsi Anda. Saya menamainya hello-world dan memberinya deskripsi dan runtime Node.js 6.10. AWS juga memungkinkan Anda untuk mengedit kode Anda secara inline atau mengunggah zip.

Karena fungsi ini akan sederhana dan kecil, kita dapat mengeditnya secara inline. Gambar 3.11 menunjukkan seperti apa konfigurasi Anda nantinya. Exports.handler mendefinisikan ini sebagai pengendali fungsi. Ia mengambil peristiwa (pemicu), konteks, dan panggilan balik, yang akan kita gunakan untuk menandakan bahwa fungsi telah selesai dijalankan. Panggilan balik kita saat ini merespons “Halo dari Lambda.” Selanjutnya kita perlu mengonfigurasi pengendali dan peran kita.

Kita akan membiarkan Variabel Lingkungan, Tag, dan Pengaturan Lanjutan kosong untuk saat ini, tetapi jangan ragu untuk memeriksanya. Pengendali kita adalah index.handler, dan peran kita adalah peran lambda_basic_execution yang kita buat sebelumnya. Setelah ini dikonfigurasi (Gambar 3.12), kita dapat melanjutkan dan membuat fungsi kita.



Gambar 3.11 Untuk Tujuan Fungsi Ini, Kita Hanya Akan Membuatnya Merespons "Halo Dari Lambda"



Gambar 3.12 Sebelum Melanjutkan, Pastikan Pengendali Fungsi Dan Peran Lambda Anda Terlihat Seperti Ini

Selanjutnya, kita akan melihat pengujian dan eksekusi fungsi kita. Untuk saat ini, kita tidak akan menyiapkan pemicu, karena itu memerlukan konfigurasi pemicu. Kita hanya ingin

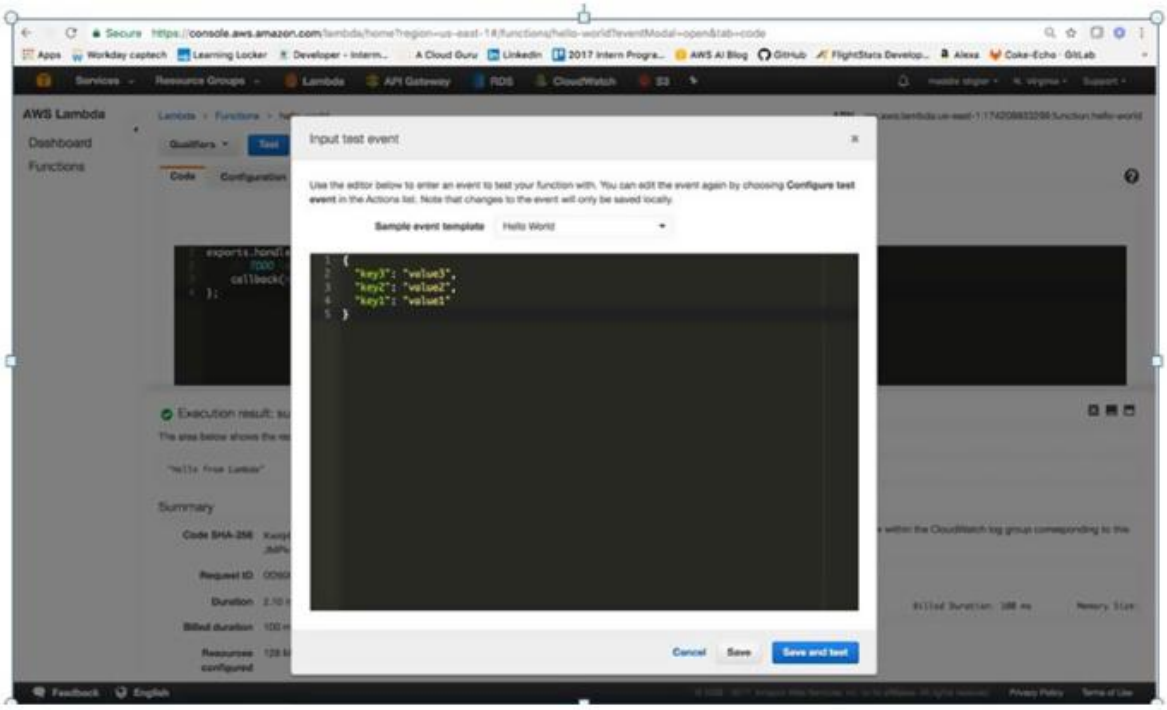
melihat seperti apa Lambda yang sedang dieksekusi, bagaimana kita dapat membuatnya, dan bagaimana kita dapat mengakses log.

Pengujian

Untuk menguji fungsi Lambda kita di konsol, kita dapat menggunakan tindakan Konfigurasikan Peristiwa Uji. Di dalam fungsi, jika Anda mengeklik Tindakan, Anda akan melihat daftar tindakan yang dapat Anda lakukan pada Lambda Anda:

- Konfigurasikan Peristiwa Uji
- Publikasikan Versi Baru
- Buat Alias
- Hapus Fungsi
- Ekspor Fungsi

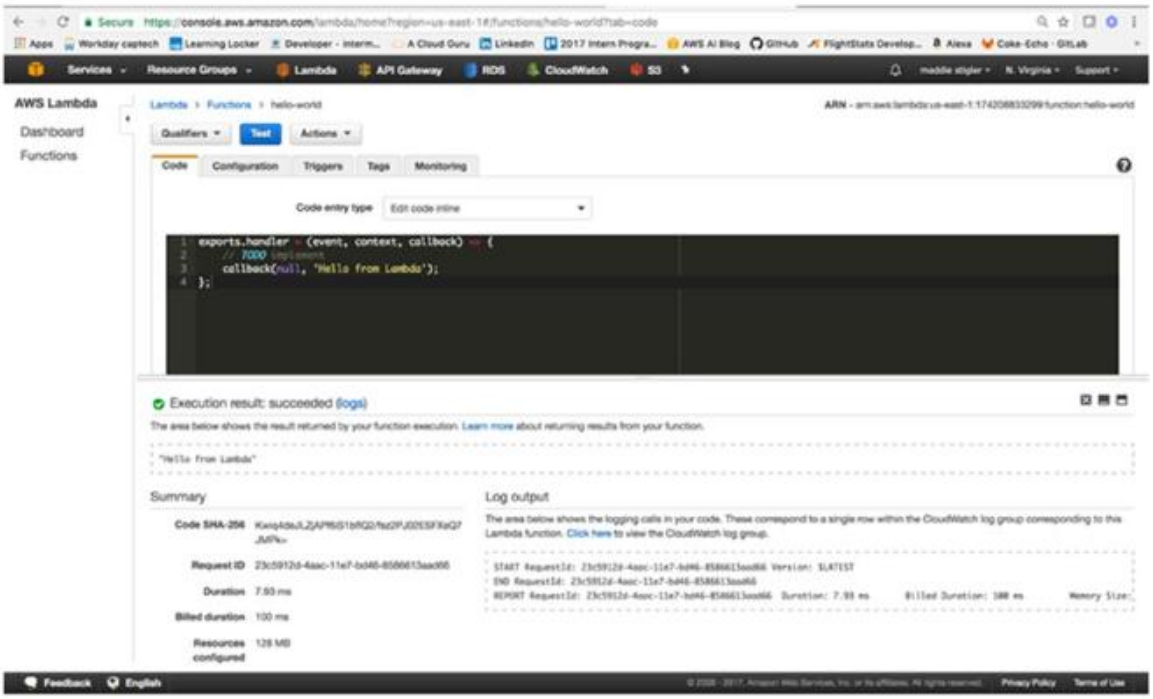
Kita akan mengonfigurasi peristiwa uji. Dalam peristiwa uji input, AWS menyediakan beberapa templat peristiwa. Jangan ragu untuk menjelajahinya untuk melihat seperti apa berbagai peristiwa masuk. Kita akan menggunakan peristiwa Hello World, seperti yang ditunjukkan pada Gambar 3.13. Peristiwa ini hanya menawarkan JSON dari berbagai kunci dan variabel.



Gambar 3.13 Contoh Templat Peristiwa Pengujian Yang Disediakan Oleh AWS. Kami Akan Menyimpan Dan Menguji Peristiwa Ini.

Karena Lambda kami tidak dikonfigurasi untuk melakukan hal tertentu dengan peristiwa tersebut, kami seharusnya dapat memperoleh respons Hello World dari peristiwa pengujian kami hanya dengan memicunya. Peristiwa pengujian bekerja dengan cara yang sama seperti pemicu, yang menyebabkan fungsi Lambda dijalankan dan merespons peristiwa yang masuk. Anda diberikan opsi untuk Menyimpan dan Menyimpan dan Menguji. Dengan tombol Simpan, fungsi tersebut tidak dijalankan. Simpan dan Uji menyimpan fungsi Anda dan

mengujinya menggunakan kasus pengujian yang disediakan. Ini adalah arsitektur berbasis peristiwa yang sedang beraksi. Gambar 3.14 menunjukkan hasil Eksekusi, Ringkasan, dan Keluaran Log.



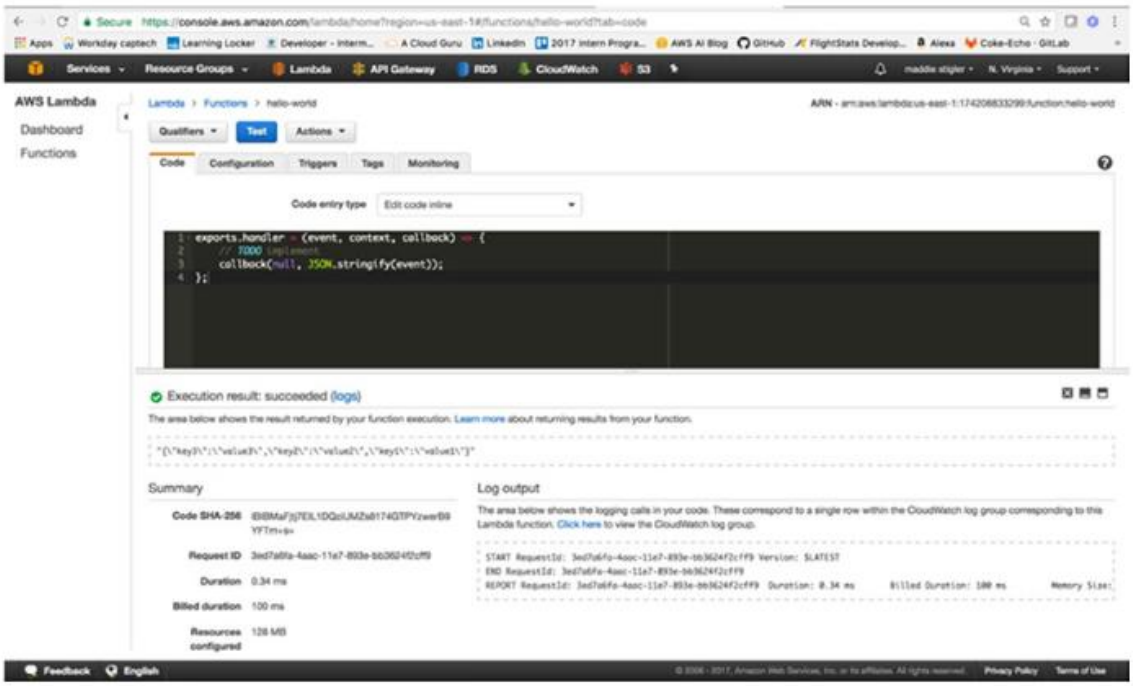
Gambar 3.14 Eksekusi Yang Berhasil Dengan Pencatatan

Hasil eksekusi menunjukkan apa yang kami tentukan untuk dilakukan Lambda kami. Karena kami hanya menentukan panggilan balik dengan string, itulah yang kami terima. Ringkasan menunjukkan durasi eksekusi, durasi yang ditagih, dan jumlah sumber daya yang dikonfigurasi. Ini penting untuk konfigurasi lebih lanjut fungsi Anda.

Jika Lambda Anda hanya menggunakan sedikit memori, sebaiknya sesuaikan konfigurasinya untuk membatasi ruang dan biaya yang tidak perlu. Kami juga melihat output log. Bahkan tanpa pencatatan eksplisit, kami diberi Awal eksekusi dengan ID permintaan dan versi yang dijalankan, Akhir permintaan, dan laporan akhir. Berikut ini menunjukkan kode yang kami jalankan untuk fungsi Hello World kami:

```
exports.handler = (event, context, callback)=>{
   callback(null, JSON.stringify(event));
};
```

Sekarang setelah kita menunjukkan fungsi yang sedang dijalankan, mari kita ubah untuk mengulang kembali peristiwa yang masuk.

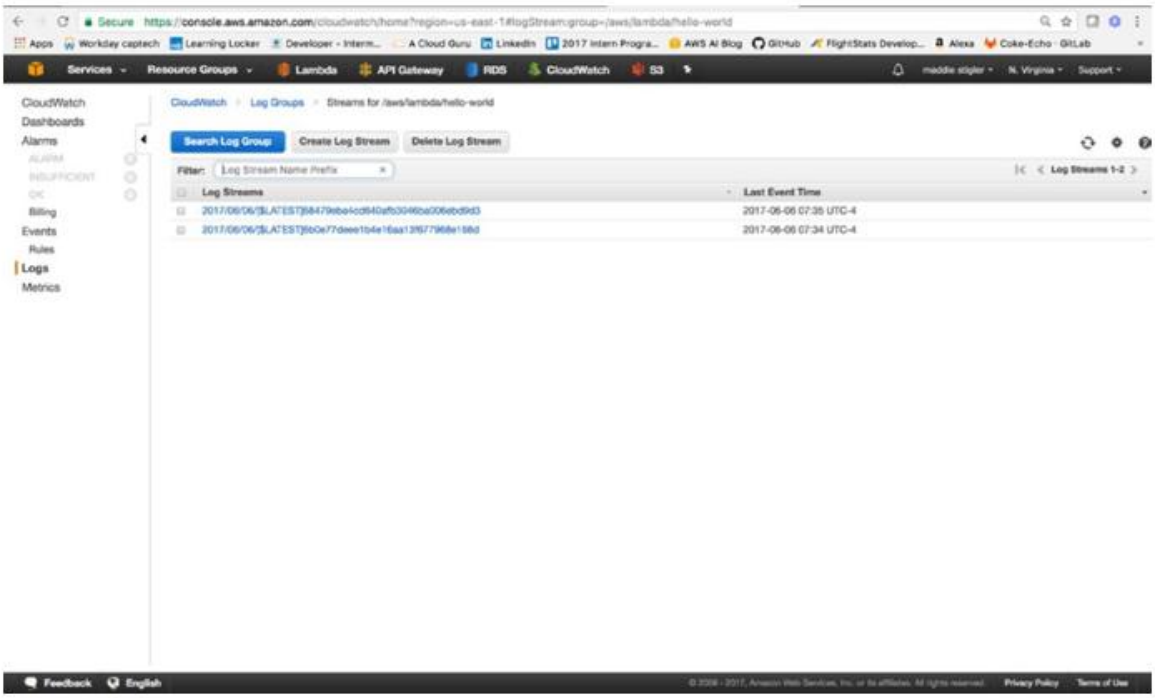


Gambar 3.15 Fungsi Lambda Yang Telah Kita Perbarui

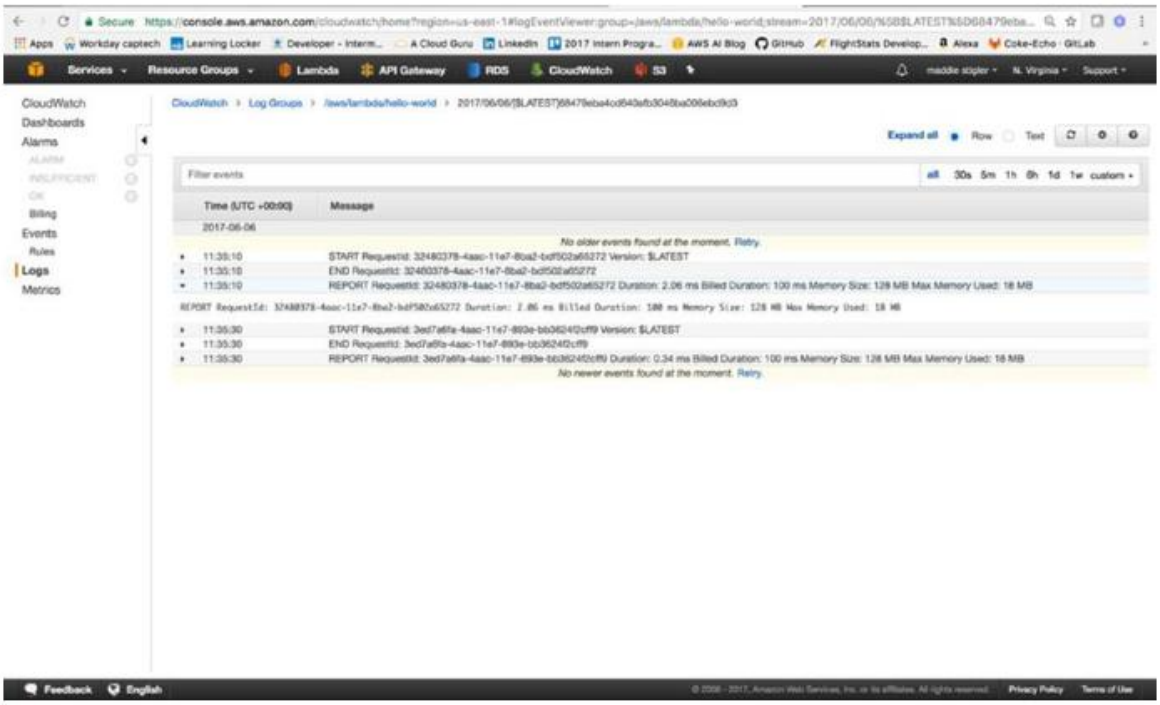
Untuk menyempurnakan fungsi Hello World ini, coba parsing peristiwa dan tanggapi bagian-bagiannya. Sertakan juga pencatatan sebaris yang dapat Anda lihat di keluaran log dan di CloudWatch. Kiat Apakah Anda menggunakan konsol atau tidak, akan sangat membantu jika Anda memasukkan banyak pernyataan console.log. Pernyataan ini memungkinkan Anda melihat apa yang sebenarnya terjadi di dalam kode Anda dan dengan data yang masuk dan keluar, dan output-nya dikirim langsung ke CloudWatch.

3.3 CLOUDWATCH

Sekarang, kita akan memeriksa CloudWatch dan melihat di mana pencatatan kita terjadi dan semua metrik yang kita dapatkan di luar kotak pada fungsi kita. Di bagian keluaran log, navigasikan ke opsi Klik Di Sini. Ini akan membawa kita ke portal CloudWatch, tempat log fungsi kita berada. Gambar 3.16 menunjukkan dua eksekusi yang saya buat sebelumnya dalam latihan ini.



Gambar 3.16 Grup Log Cloudwatch Untuk Lambda Anda Menyimpan Aliran Untuk Eksekusi Anda. Anda Juga Dapat Mengeklik Aliran Tertentu Dan Menyelidiki Log Dalam Jangka Waktu Tersebut



Gambar 3.17 Aliran Yang Dibuka Memberi Anda Waktu UTC Dan Pesan Terperinci. Anda Dapat Melihat Log Dalam Aliran Tertentu Hingga Seminggu Yang Lalu

Log ini berguna untuk menganalisis dan memantau eksekusi fungsi Anda. Selain CloudWatch, ada tab Pemantauan di konsol fungsi Lambda yang memberi Anda ikhtisar tingkat tinggi

tentang eksekusi dan keluaran fungsi Anda. Untuk melihat lebih dalam ke dalam log, klik alirannya. Ini akan memberi Anda log terperinci lengkap untuk eksekusi tersebut. Gambar 3.17 menunjukkan eksekusi pada fungsi Hello World saya. Sekarang setelah kita terbiasa dengan konsol Lambda, acara pengujian, dan CloudWatch, kita akan membangun fungsi Hello World dengan variabel lingkungan.

Variabel Lingkungan

Untuk menambahkan fungsi Hello World, kita akan membahas variabel lingkungan: apa itu, bagaimana cara menggunakannya, dan bagaimana cara mengonfigurasinya di AWS.

Apa Itu Variabel Lingkungan

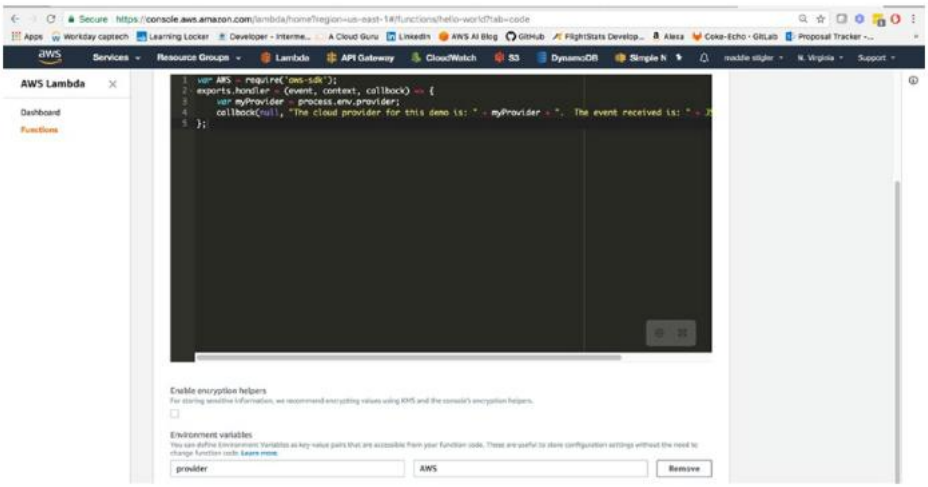
Variabel lingkungan adalah variabel yang ditetapkan secara global di seluruh aplikasi tanpa server Anda. Variabel tersebut diberi kunci dan nilai. Nilai adalah nilai variabel sebenarnya, sedangkan kunci adalah nama yang digunakan untuk variabel tersebut di seluruh aplikasi. Manfaat variabel lingkungan adalah keamanan dan kemudahan penggunaan.

Daripada membiarkan kunci API dan berbagai informasi akses tersebar di seluruh kode, Anda dapat menetapkan variabel aman yang sebenarnya ke variabel lingkungan untuk digunakan secara anonim. Selain itu, jika Anda tahu akan menggunakan variabel berulang kali, menetapkannya sebagai variabel lingkungan memungkinkan Anda mengaksesnya di seluruh proyek tanpa mendeklarasikannya ulang. Ini juga memungkinkan Anda membuat perubahan cepat di satu tempat.

Menggunakan Variabel Lingkungan di Hello World

Kita akan membuat variabel lingkungan dengan kunci provider dan nilai AWS. Ini juga menunjukkan praktik terbaik untuk memisahkan logika provider dari kode Anda guna mencegah vendor lock-in. Sementara untuk contoh ini kita hanya menggunakan nilai AWS, nanti dapat digunakan untuk mewakili layanan yang berbeda.

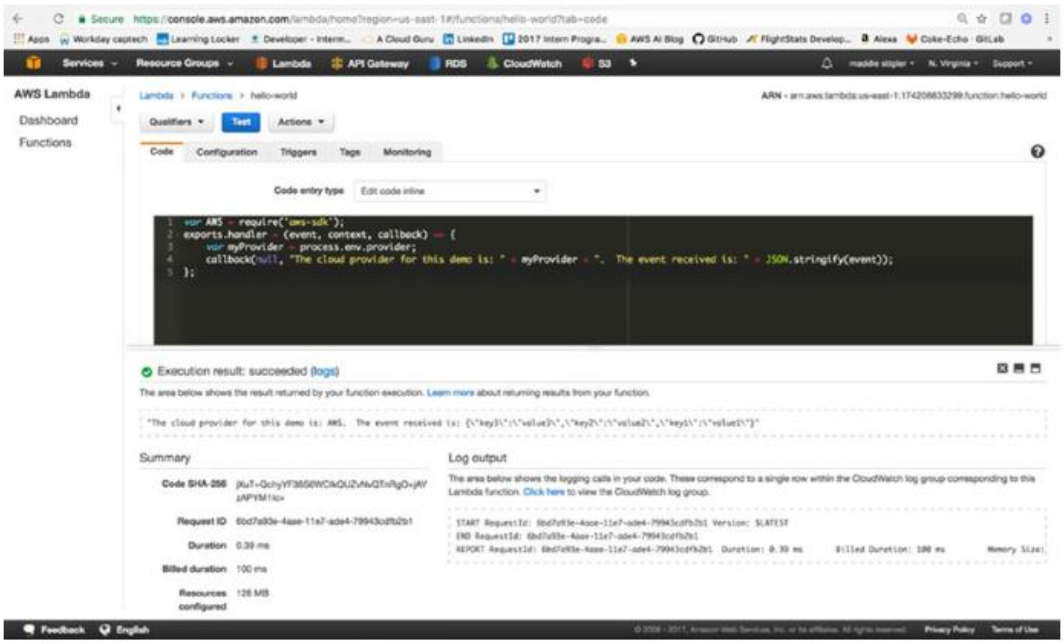
Misalnya, jika kita tahu kita ingin mengakses database, kita dapat menggunakan kunci DB_Host dan menetapkan nilai khusus untuk nama host database AWS. Ini membuatnya mudah dikonfigurasi jika kita memilih untuk pindah ke penyedia cloud yang berbeda. Gambar 3.18 menunjukkan di mana dan bagaimana kita mengonfigurasi variabel lingkungan kita.



Gambar 3.18 Anda Dapat Mengonfigurasi Variabel Lingkungan Dalam Konsol AWS Lambda

Sekarang kita dapat mengakses variabel lingkungan ini dalam kode kita. Gambar 3-19 menunjukkan cara kita merujuk variabel lingkungan dan keluaran log untuk pelaksanaan fungsi Lambda.

```
var AWS = require('aws-sdk');
exports.handler = (event, context, callback) => {
  var myProvider = process.env.provider;
  callback(null, "The cloud provider for this demo is: " +
  myProvider + ". The event
received is: "+ JSON.stringify(event));
};
```



Gambar 3.19 Variabel Lingkungan Diakses Melalui Process.Env.Variable-Key

Ini menunjukkan betapa mudahnya membuat dan mengakses variabel dalam kode Anda. Sekarang setelah kita menyelesaikan demonstrasi Hello World Lambda, kita akan melihat pembuatan aplikasi baru yang menggunakan peristiwa HTTP dan meresponsnya dengan mengembalikan data dari DynamoDB.

Peristiwa HTTP

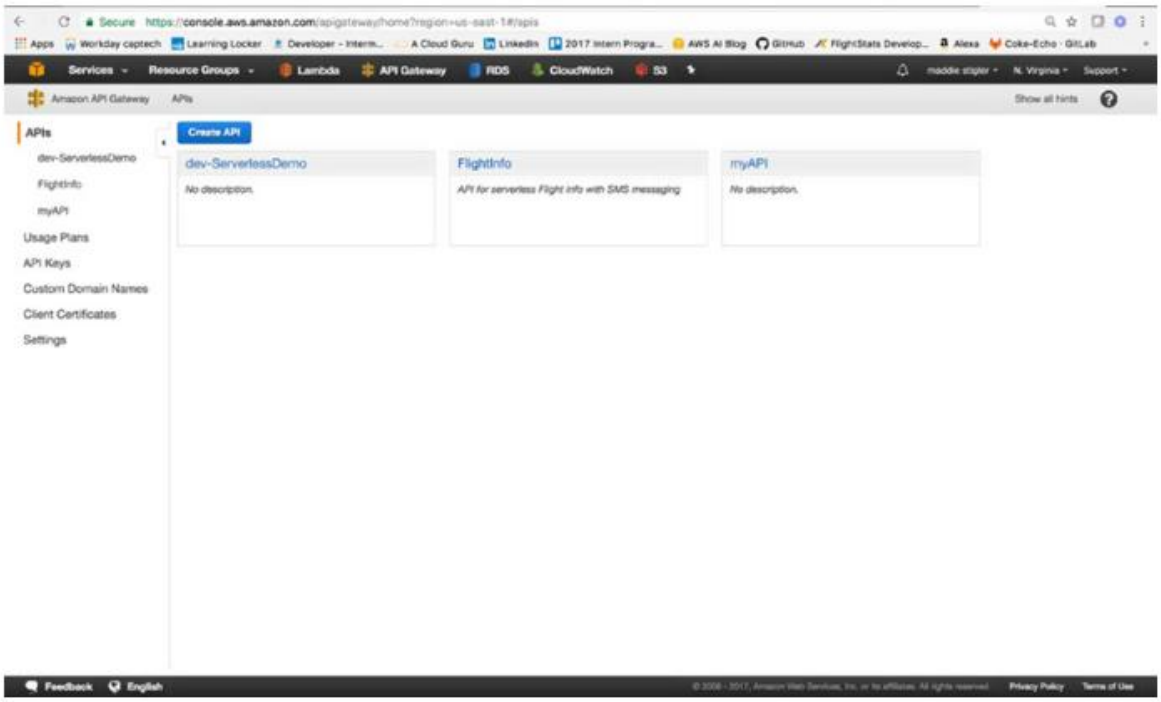
Untuk aplikasi nirserver pertama kita yang disempurnakan dengan Lambda, kita akan menggunakan AWS API Gateway untuk memicu fungsi Lambda yang mengembalikan data dari Basis Data DynamoDB NoSQL. API Gateway memungkinkan Anda membuat sumber daya dan metode HTTP dan menyetelnya ke titik akhir tertentu. Aplikasi ini akan meniru buku resep virtual. Kita akan membuat API Gateway dengan satu sumber daya, Recipes, dan satu metode untuk sumber daya tersebut, GET.

Untuk titik akhir permintaan GET ini, kita akan menyetel fungsi Lambda yang kita buat, yang disebut GetRecipes. Fungsi ini akan mengakses tabel DynamoDB yang telah kita isi

sebelumnya dengan resep dan mengembalikan nilai JSON dari resep ini sebagai respons. Sebelum menyiapkan API Gateway, kita akan membuat fungsi Lambda, dan membiarkan pemicu dan kode kosong untuk saat ini. Setelah Anda melakukannya, Anda dapat melanjutkan menjelajahi API Gateway.

3.4 MENJELAJAHI API GATEWAY

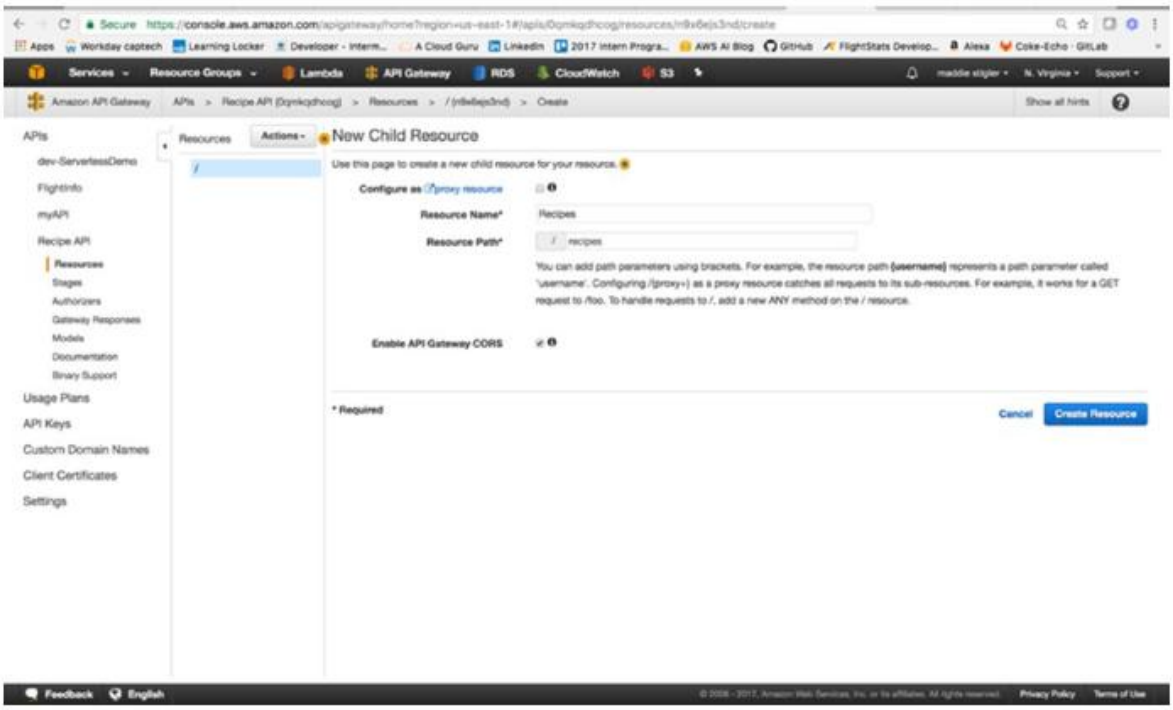
API Gateway adalah layanan AWS yang memudahkan Anda membuat dan mengakses API melalui konsol API Gateway.



Gambar 3.20 Dengan API Gateway, Anda Dapat Dengan Mudah Mengakses Dan Membuat API Serta Menetapkan Kunci Dan Rencana Penggunaan Untuk API Tersebut

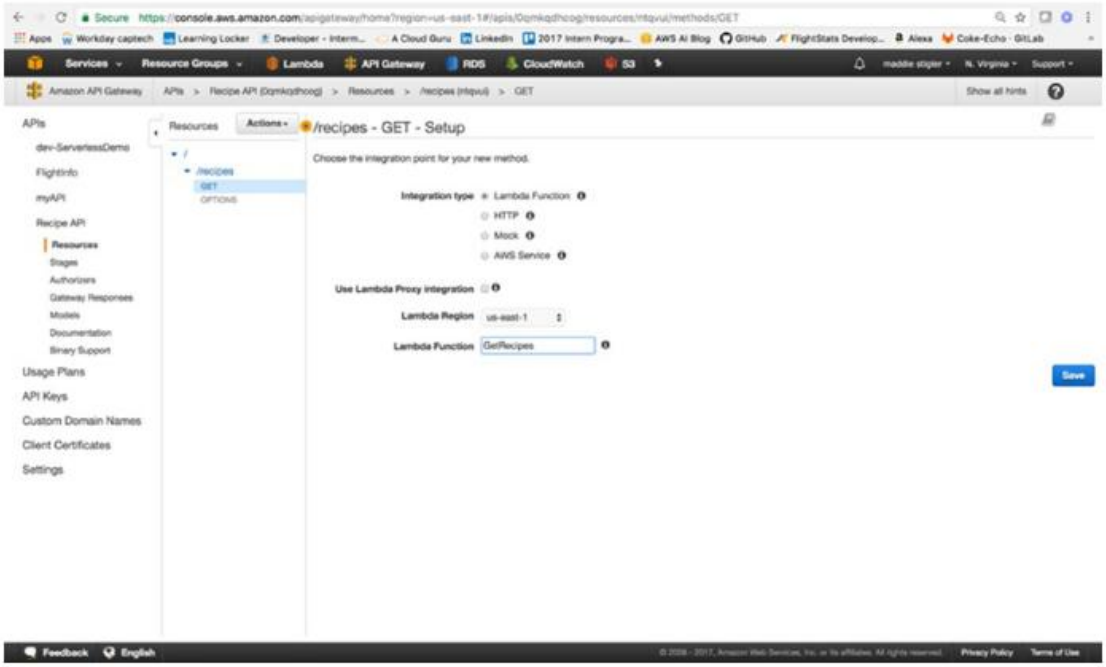
Layanan ini menyediakan antarmuka API RESTful publik ke sejumlah besar layanan AWS. Layanan ini memudahkan Anda berinteraksi dengan basis data, layanan pengiriman pesan, dan fungsi Lambda melalui gateway yang aman. Selain itu, layanan ini sangat mudah disiapkan dan dibuat titik akhir. Layanan ini memungkinkan pengembangan yang cepat. Untuk mulai menggunakan API Gateway, navigasikan ke layanan API Gateway di bawah Layanan Aplikasi. Konsol API Gateway menampilkan API, paket penggunaan, Kunci API, Nama Domain Kustom, Sertifikat Klien, dan Setelan Anda.

Gambar 3.20 menunjukkan contoh Konsol API Gateway. Untuk membuat API, cukup klik Buat API. Ini akan memandu Anda melalui proses penyiapan API Anda sendiri. Kita akan membuat API Baru dan menamainya API Resep. Setelah membuat API, Anda akan diarahkan ke konsol yang memungkinkan Anda mengonfigurasi API (Gambar 3.21). Kita ingin menambahkan sumber daya ke API kita yang disebut "resep".



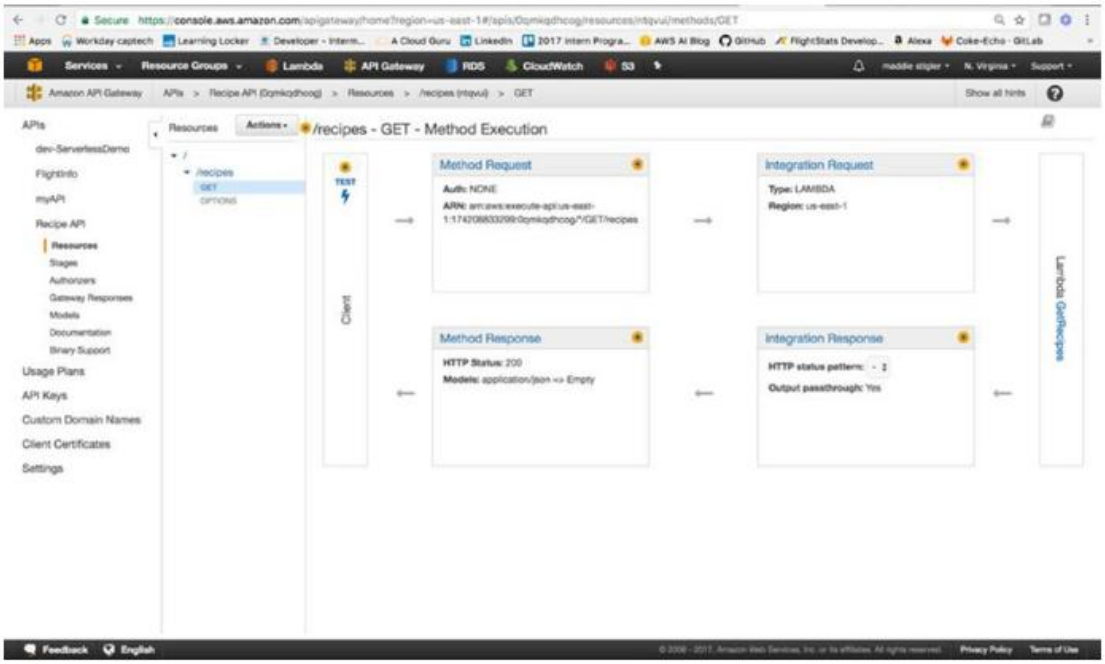
Gambar 3.21 Sumber Daya Anak Baru Dapat Ditambahkan Ke Sumber Daya API Anda Dan Akan Digunakan Dalam URL Permintaan

Sumber daya memungkinkan Anda memiliki beberapa objek untuk metode Anda. Kami juga menginginkan satu metode GET yang dilampirkan ke sumber daya kami. Gambar 3.22 menunjukkan seperti apa tampilan konsol API Anda setelah mengonfigurasi titik akhir Anda.



Gambar 3.22 Metode GET Dilampirkan Ke Sumber Daya Resep Di API Resep Kami

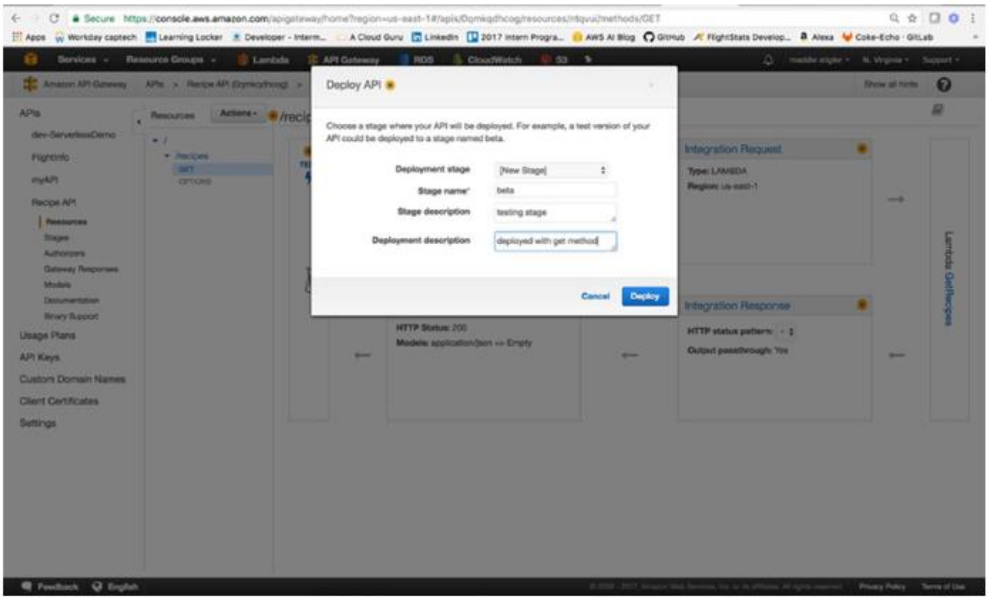
Kemudian, kami perlu mengonfigurasi integrasi untuk metode GET kami. Anda dapat meminta Lambda membuat titik akhir API Gateway untuk Anda, tetapi melakukannya dengan cara ini memberi Anda kendali atas seperti apa titik akhir itu dan konfigurasinya. Untuk ini, kami ingin memilih fungsi Lambda, menentukan wilayah, dan memilih fungsi Lambda GetRecipes kami. Ini memberi tahu gateway API saat metode GET diakses, untuk menjalankan fungsi Lambda. Gambar 3.23 menunjukkan integrasi lengkap fungsi Lambda.



Gambar 3.23 API Gateway Memungkinkan Anda Melihat Setiap Eksekusi Metode Dalam API Anda

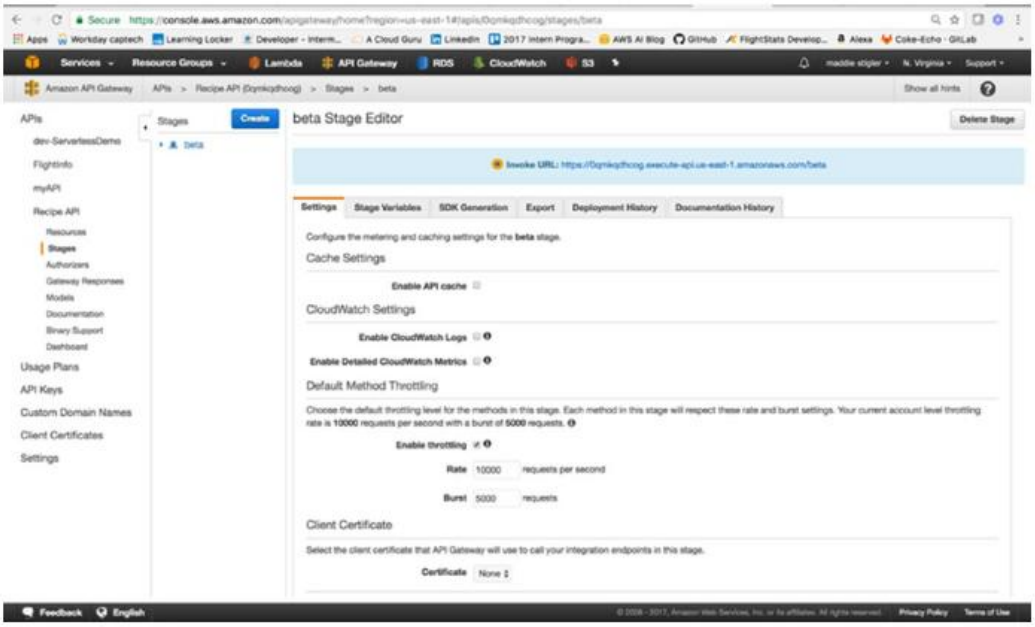
Sebelum kembali ke konsol Lambda untuk menetapkan API ini sebagai pemicu, kita perlu menerapkan API kita ke suatu tahap. Pementasan penting untuk kontrol versi dan penerapan. Karena kita masih dalam tahap pengembangan, kita akan menamai tahap kita "Beta." Di bawah Tindakan, di samping Sumber Daya, klik Terapkan.

Jendela pop-up yang ditunjukkan pada Gambar 3.24 akan meminta Anda untuk membuat tahap jika Anda belum melakukannya. Lanjutkan dan buat tahap baru dan beri nama "beta." Tahapan merupakan pengidentifikasi unik untuk versi API REST yang diterapkan yang dapat dipanggil oleh pengguna dan layanan yang berbeda.



Gambar 3.24 Kita Tentukan Nama Tahap Saat Kita Menerapkan API. Jika Kita Ingin Membuat Perubahan Dan Melihatnya, Kita Perlu Menerapkan Ulang API Kita Ke Tahap Ini

Setelah menerapkan, Anda akan diarahkan ke tab Tahap (Gambar 3.25). Di sinilah Anda mengonfigurasi pengaturan untuk tahap, variabel tahap, Pembuatan SDK, ekspor Swagger, ekstensi Postman, riwayat penerapan, dan riwayat dokumentasi. Lihat setiap tab ini untuk melihat semua yang tersedia bagi Anda melalui API Gateway. Swagger adalah kerangka kerja API yang memungkinkan Anda untuk melihat dan berinteraksi dengan titik akhir Anda dengan mudah. Saya menyukainya karena memudahkan pengujian layanan Anda dan juga mudah ditambahkan ke proyek Anda.



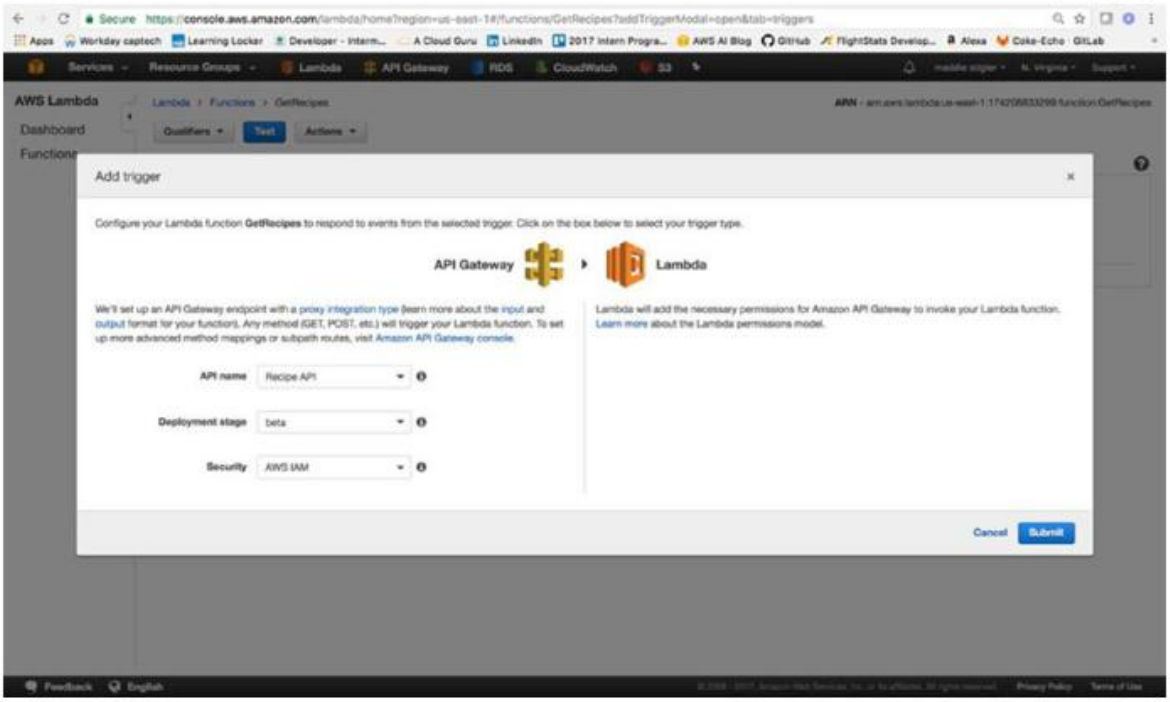
Gambar 3.25 Kita Akan Menggunakan URL Invoke Untuk Menguji Fungsionalitas API Gateway Kita Sebagai Pemicu Untuk Lambda

Pada titik ini, kita memiliki API Gateway dengan satu sumber daya dan satu metode, yang disebarkan ke tahap Beta, dan terintegrasi dengan fungsi Lambda kita. Kita siap untuk menetapkan API ini sebagai pemicu untuk fungsi Lambda kita dan mulai menanggapi permintaan. Anda mungkin bertanya-tanya mengapa kita hanya memberikan satu metode pada API kita.

Sebelumnya dalam buku ini, kita membahas pentingnya setiap fungsi memiliki satu tugas dan peristiwa pemicu. Kita dapat menambahkan beberapa metode ke API ini, tetapi kita juga ingin membuat beberapa fungsi Lambda untuk menanggapi permintaan yang berbeda (misalnya, Lambda untuk menangani permintaan POST, Lambda untuk menangani permintaan DELETE).

Menggunakan API Gateway sebagai Pemicu

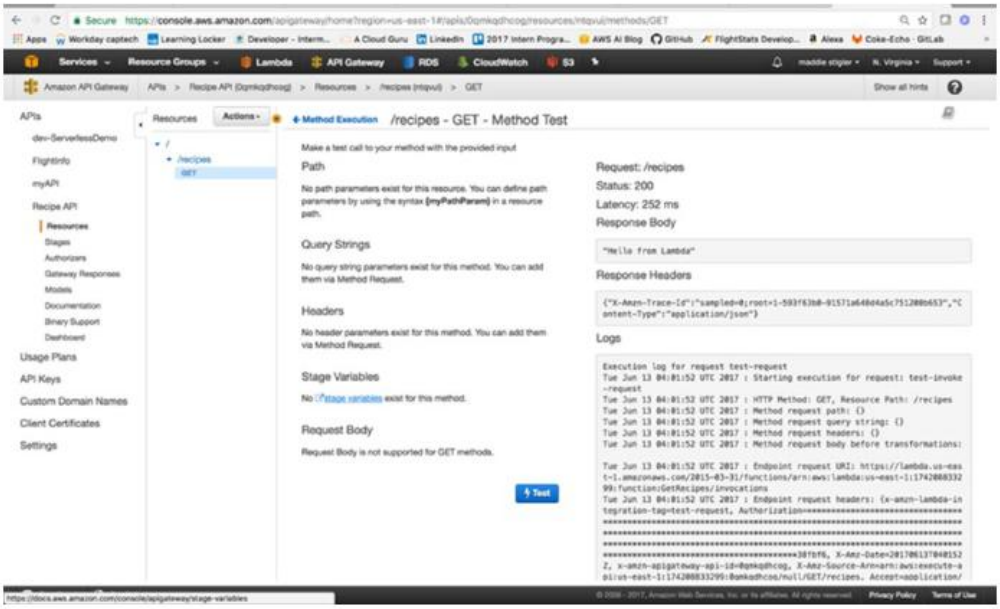
Kembali ke fungsi Lambda, GetRecipes, kita sekarang dapat mengonfigurasi API yang baru saja kita buat sebagai pemicu. Di bawah Pemicu dalam fungsi kita, klik Tambahkan Pemicu. Di sini, kita pilih API Gateway dari menu tarik-turun dan tentukan API yang baru saja kita buat. Ini akan memberi tahu fungsi Lambda kita untuk mengaktifkan peristiwa yang masuk dari layanan khusus ini. Gambar 3.26 menunjukkan konfigurasi yang benar untuk fungsi kita. Dengan membiarkan kode Lambda apa adanya, kita dapat masuk ke API Gateway dan menguji pemicu kita. Dengan mengeklik sumber daya GET, kita dapat memilih Uji dan menguji API kita (Gambar 3.27).



Gambar 3.26 API Resep Kita Dikonfigurasi Untuk Memicu Fungsi Getrecipes Lambda Kita

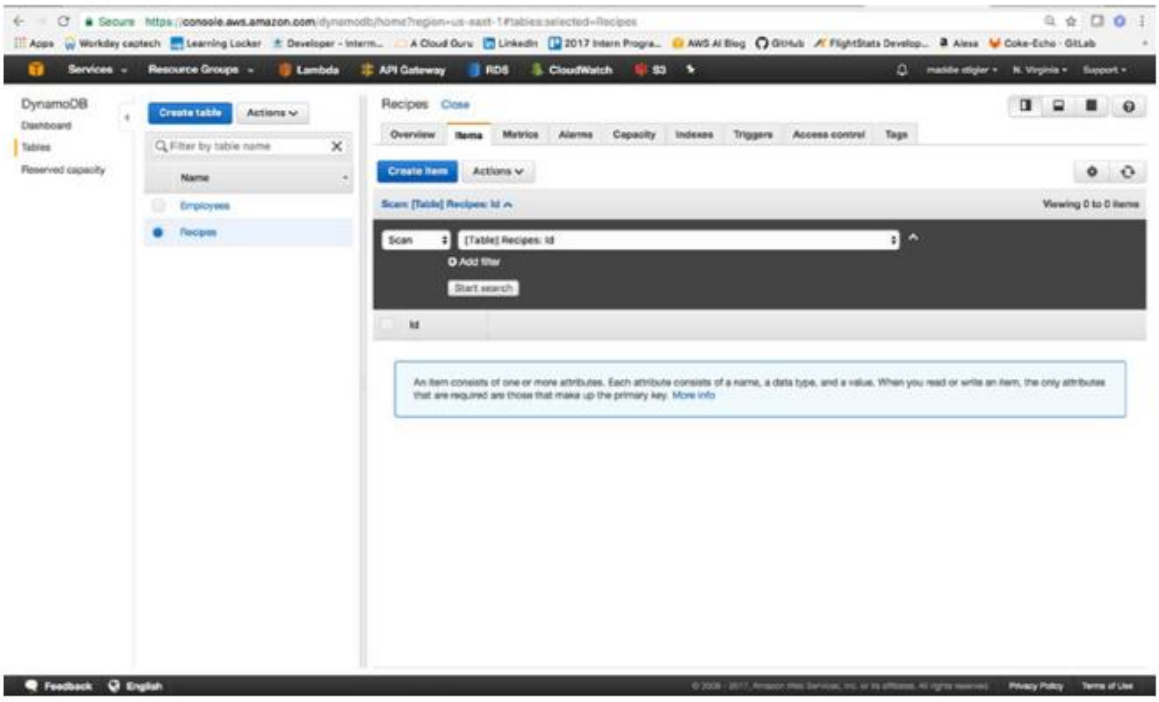
Karena ini adalah metode GET, metode ini tidak memerlukan isi permintaan. Di sebelah kanan di API Gateway, Anda akan melihat respons, status, dan latensi. Anda juga dapat melihatnya di browser hanya dengan menekan titik akhir GET. URI untuk contoh ini adalah:

https://lambda.us-east-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-east-1:174208833299:function:GetRecipes/incovations



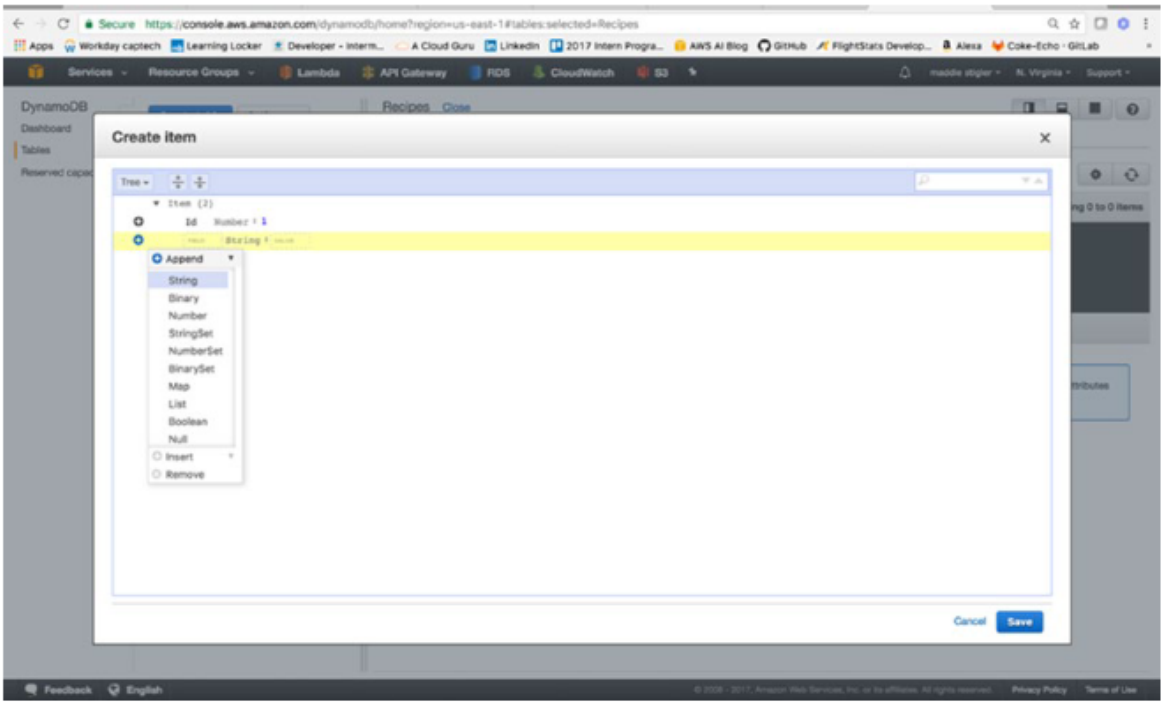
Gambar 3.27 Menguji Metode Dan Respons API Anda. Jika Kita Telah Menetapkannya Sebagai Metode POST, Kita Dapat Menempatkan Permintaan JSON Di Badan Permintaan

Sekarang pemicu kita telah dikonfigurasi, kita dapat mengembangkan fungsi Lambda kita untuk menanggapi permintaan ini dengan mengembalikan resep dari tabel DynamoDB. Untuk melakukannya, pertama-tama kita akan membuat tabel DynamoDB, Resep, dan mengisinya terlebih dahulu dengan beberapa resep. Gunakan tab Layanan untuk menavigasi ke layanan DynamoDB. Di sini kita akan mengklik Buat Tabel (Gambar 3.28) dan memberi nama Tabel kita "Resep" dan menetapkan Kunci Utama "Id" dengan tipe "Nomor." Kunci utama harus unik untuk item. Tabel DynamoDB terdiri dari item dan atribut. Di tabel kita, setiap item adalah resep yang memiliki atribut makanan, deskripsi, dan waktu persiapan.



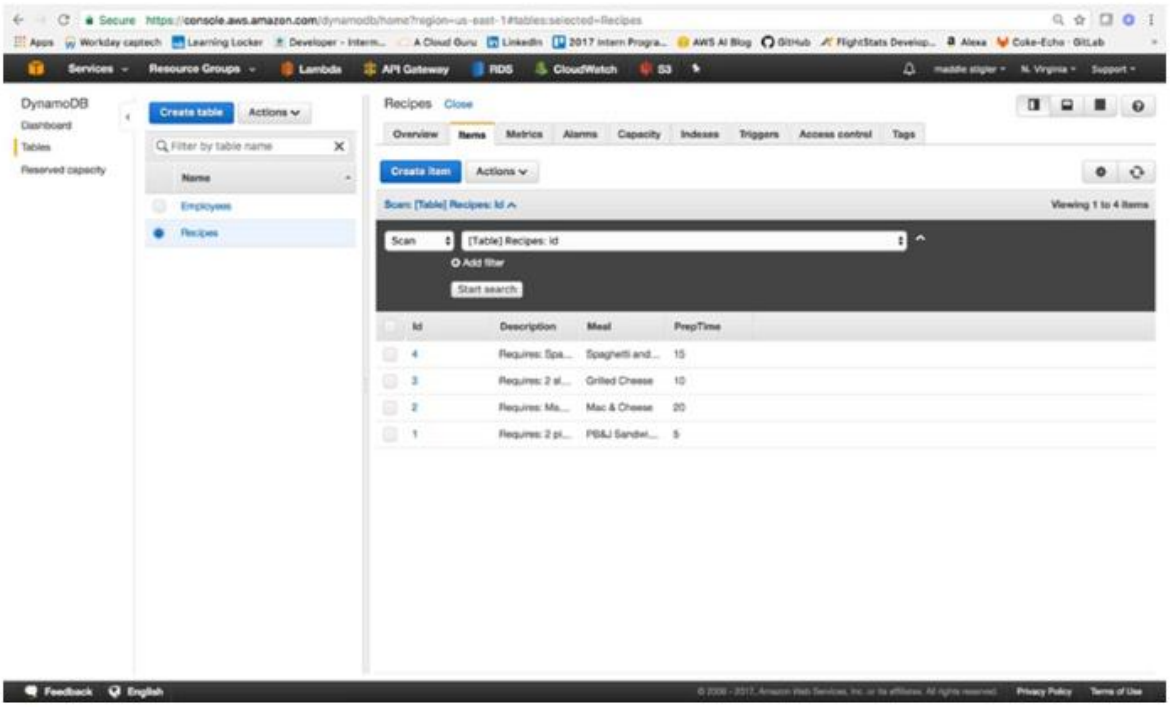
Gambar 3.28 Tabel Dynamodb Kosong Dengan Kunci Utama Id

Dari sini, kita akan membuat beberapa item resep untuk mengisi tabel kita. Untuk melakukannya, klik Buat Item dan tambahkan kolom Makanan, Deskripsi, dan Waktu Persiapan dengan berbagai nilai (Gambar 3.29).



Gambar 3.29 Gambar Ini Menunjukkan Cara Menambahkan Kolom Ke Item

Saat tabel kita telah diisi dengan beberapa resep, kita dapat membuat fungsi Lambda untuk mengembalikan resep ini sebagai respons terhadap API (Gambar 3.30).



Gambar 3.30 Tabel Dynamodb Yang Terisi

Untuk membuat bagian Lambda dari aplikasi tanpa server, kita akan menggunakan TypeScript, Node, dan NPM. Bagian berikutnya akan membahas cara melakukannya.

3.5 RESPONS TERHADAP PEMICU

Untuk fungsi Lambda, kita akan mengembangkan dalam Visual Studio Code, menggunakan TypeScript, Node, NPM, dan AWS SDK. Pertama, penting untuk memformat struktur proyek sehingga kita dapat dengan mudah mengompres dan menyebarkan fungsi. Gambar 3.31 menunjukkan struktur yang saya pilih untuk membuat fungsi Lambda ini. Dalam proyek AWS, saya membuat proyek HTTPTrigger dengan folder Bersama dan folder GetRecipes. Folder GetRecipes akan menampung file handler.js, yang akan dipicu oleh permintaan GET. Folder Bersama berisi model Resep yang menentukan struktur permintaan masuk.

Gambar 3.31 Struktur Proyek Yang Diusulkan Untuk Peristiwa HTTP

Untuk saat ini, kita akan membuat fungsi tersebut tanpa menggunakan Serverless Framework, jadi di luar proyek HTTPTrigger, kita akan memerlukan file package.json untuk menentukan NPM apa yang diinstal, dan file tsconfig.json untuk mengonfigurasi build TypeScript kita. File package.json Anda harus menyertakan aws-sdk dan TypeScript sebagai dependensi.

Daftar 3.1 File Package.Json Lengkap Untuk Aplikasi Ini

```
}
"name": "aws-nodejs",
"version": "1.0.0",
"description": "AWS Lambda sample for the Serverless framework",
"main": "handler.js",
"keywords": [
"aws",
'serverless"
1,
"dependencies": {
"aws-sdk": "^2.34.0"
},
"devDependencies": {
"@types/aws-lambda": "0.0.9",
```

```
"@types/aws-sdk": "0.0.42",
"@types/node": "^7.0.12",
"aws-sdk-typescript": "0.0.3",
"typescript": "2.1.6",
"typings": "^1.3.3"
}
}
```

tsconfig.json harus dikonfigurasi untuk dibuat saat disimpan dan dikompilasi saat disimpan. Ini akan mengkompilasi file JavaScript untuk file TypeScript saat Anda menyimpan. Daftar 3.2 menunjukkan file tsconfig.json.

Daftar 3.2. File Typescript Mengecualikan File Node_Modules, Vscode, Git, Dan Serverless Dalam Pembuatannya

```
{
"compilerOptions": {
"module": "commonjs",
"target": "es2015",
"noImplicitAny": false,
"sourceMap": true,
"emitDecoratorMetadata": true,
"experimentalDecorators": true,
"declaration": false,
"listFiles": false,
"moduleResolution": "node",
"rootDirs": [
" ./"
]

},
"exclude": [
".vscode",
".serverless",
".git",
"node_modules"
],
"compileOnSave": true,
"buildOnSave": true,
"atom": {
"rewriteTsconfig": false
}
}
```

Sekarang kita dapat melakukan instalasi NPM pada proyek kita untuk menginstal semua modul node yang kita perlukan untuk membuat fungsi Lambda. Ini akan membuat folder node_modules di proyek Anda dengan semua dependensinya. Kita juga akan membuat file recipeModel.ts (Daftar 3.3) di folder Shared. Model ini akan menentukan struktur resep yang kita buat di tabel DynamoDB kita. Kita kemudian dapat menggunakan ini di file handler.js kita untuk memformat respons kita terhadap permintaan GET. Di masa mendatang, dengan permintaan lain, Anda dapat menggunakan model ini untuk memformat permintaan.

Daftar 3.3. File Recipemodel.Ts Digunakan Untuk Memformat Permintaan Dan Respons Sehingga Cocok Dengan Struktur Tabel Dynamodb Kita

```
 export interface RecipeModel {
 Id:number,
 Description:string,
 Meal:string,
 PrepTime:number
}
```

Dalam file handler.ts, kita akan membuat modul GetRecipes yang akan menerima event, konteks, dan panggilan balik (seperti yang telah kita lakukan dalam contoh Hello World) dan akan menggunakan aws-sdk untuk berkomunikasi dengan Tabel DynamoDB dan menanggapi permintaan kita dengan daftar resep. Daftar 3.4 menunjukkan fungsi handler ini, diikuti dengan langkah-langkah yang akan memungkinkan kita untuk membahasnya lebih lanjut.

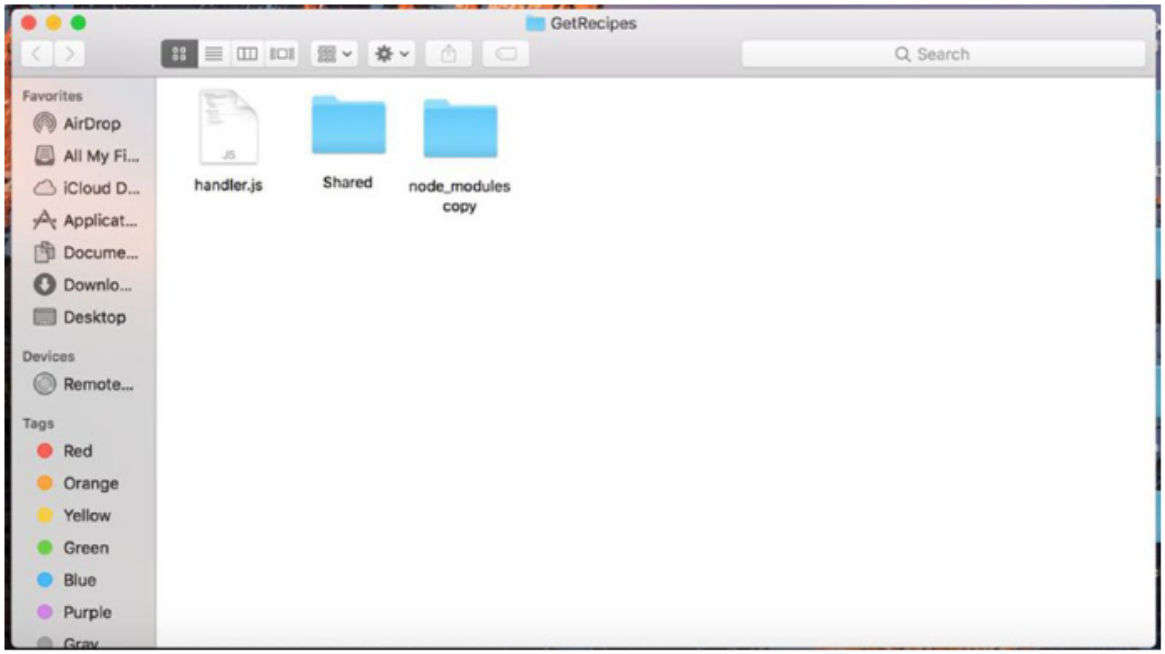
Daftar 3.4 Fungsi Handler.Ts Menerima Event HTTP Dan Menanggapinya Dengan Mengakses Dynamodb Dan Mengambil Daftar Lengkap Resep

```
'use strict';
exports. esModule = true;
var AWS = require("aws-sdk");
module.exports.GetRecipes = function (event, context, callback)
{
   console.info("Received event: ", JSON.stringify(event, null,
   2));
   var docClient = new AWS.DynamoDB.DocumentClient();
   var table = process.env.TABLE_NAME;
   var response = {
   statusCode: 200,
   message: []
};
var params = {
    TableName: table,
```

```
    ProjectionExpression: "#id, #m, #d, #pt",
    ExpressionAttributeNames: {
    "#id": "Id",
    "#m": "Meal",
    "#d": "Description",
    "#pt": "PrepTime"
 }
};
console.log("Scanning Recipes.");
docClient.scan(params, onScan);
function onScan(err, data) {
 if (err) {
  response.statusCode = 500;
  console.error("Unable to scan the table. Error JSON:",
  JSON.stringify(err, null, 2));
  callback(null, response);
}
else if (data == null) {
  response.statusCode = 404;
  callback(null, response);

else {
   console.log("Scan succeeded.");
   data.Items.forEach(function (recipe) {
    response.message.push(recipe);
  });
  callback(null, response);
  }
}
};
```

1. Impor AWS SDK dan RecipeModel.
2. Buat klien DynamoDB untuk berkomunikasi dengan tabel.
3. Manfaatkan variabel lingkungan untuk nama tabel (Kami akan menyetel variabel ini di AWS).
4. Setel respons pesan.
5. Buat koneksi tabel dengan parameter (nama tabel, ekspresi, atribut).
6. Format respons untuk dipindai.

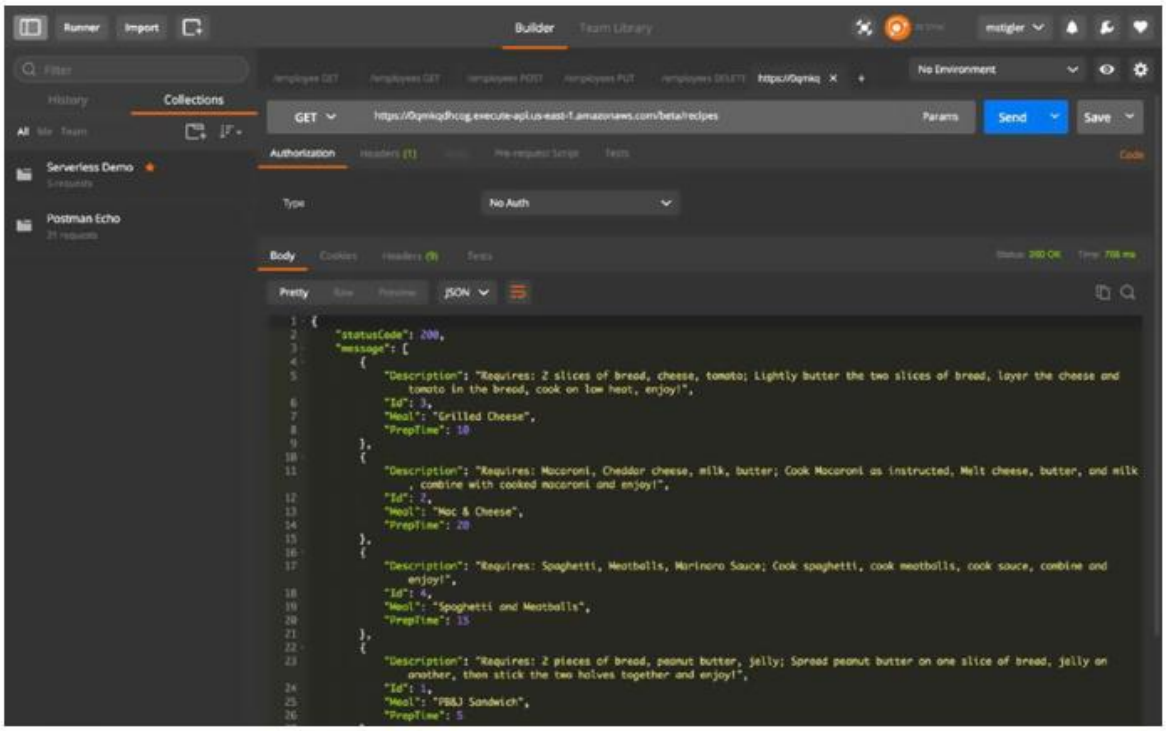
Sekarang kita dapat mengompilasi file TypeScript kita menjadi file JavaScript. Setelah kita membuat handler, model, modul node, dan mengompilasi TypeScript kita, kita dapat mengompres dan mengunggah aplikasi kita. Penting untuk diingat bahwa file handler.js harus tetap berada di root file terkompresi Anda. Kompresi harus terjadi pada level yang ditunjukkan pada Gambar 3.32.



Gambar 3.32 Fungsi Handler.Js Tetap Berada Di Level Root. Kami Hanya Menyertakan Folder Shared Dan Node_Modules, Karena Hanya Itu Yang Diperlukan Dalam Kompresi Ini

Setelah mengunggah file zip ke GetRecipes Lambda, masih ada beberapa konfigurasi yang harus diurus. Pertama, kami perlu memperbarui handler kami. Handler kami sekarang harus terdaftar sebagai handler.GetRecipes. Ini karena modul yang kami ekspor disebut GetRecipes dan ditemukan di file handler.js. Kami juga harus menambahkan variabel lingkungan TABLE_NAME dengan nilai yang tepat.

Kami juga perlu menambahkan kebijakan yang memberi kami akses ke DynamoDB ke peran Lambda kami. Ini dapat dilakukan di AWS IAM di bawah Roles. Terakhir, kami dapat menguji fungsi Lambda kami menggunakan Postman dan URL yang diberikan kepada kami di konsol API Staging. Gambar 3.33 menunjukkan permintaan Postman dan respons yang kami dapatkan.



Gambar 3.33 Kami Menggunakan URL Yang Disediakan Di API Gateway Untuk Membuat Permintaan GET

Respons kami adalah respons JSON dengan semua resep kami yang tercantum. Kami sekarang memiliki aplikasi tanpa server dari awal hingga akhir menggunakan API Gateway, Lambda, dan DynamoDB.

3.6 MENINGKATKAN FUNGSI SERVERLESS

Peningkatan dengan memisahkan logika dan memanfaatkan Serverless Framework. Pisahkan logika AWS dari pengendali:

1. Gunakan variabel Lingkungan untuk logika khusus AWS atau pindahkan logika AWS ke folder bersama
2. Buat folder Layanan yang khusus untuk AWS dan menyajikan data DynamoDB Manfaatkan

Serverless Framework:

1. Ikuti petunjuk untuk pengaturan AWS pada Serverless Framework.
2. Kembangkan dan terapkan fungsi menggunakan Serverless, bukan secara manual.

Kode untuk kedua peningkatan proyek ini dapat ditemukan di sini:
https://github.com/mgstigler/Serverless/tree/master/AWS/aws-service/HTTPTrigger

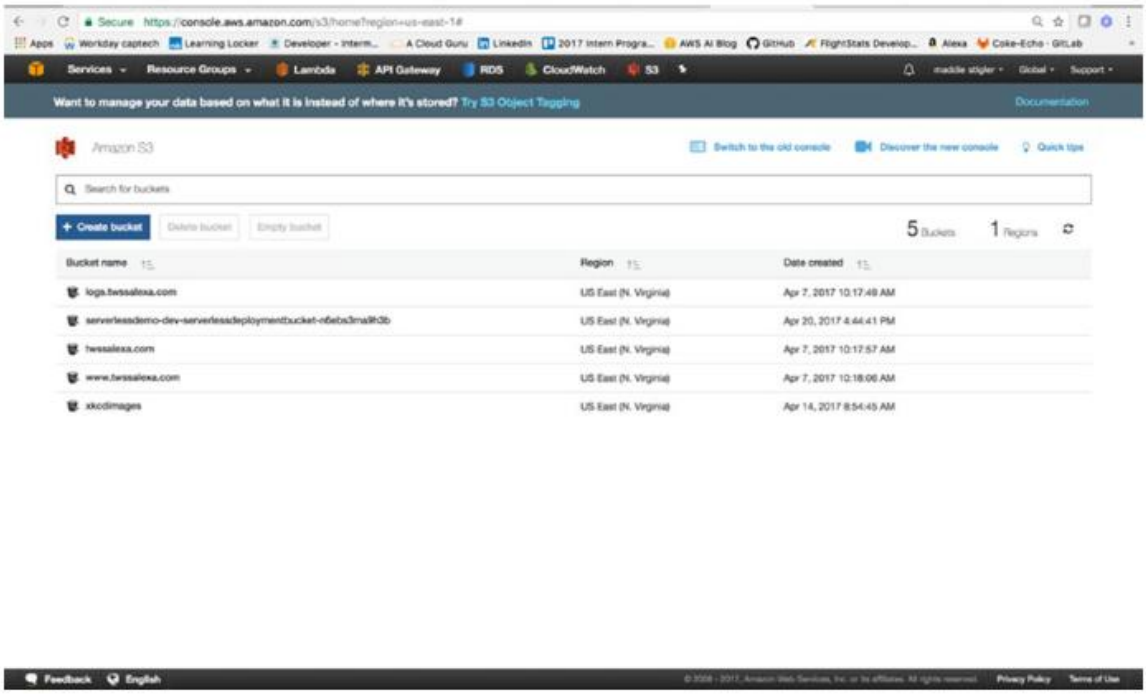
Di bagian berikutnya, kita akan menggunakan keterampilan dan alat yang kita pelajari dengan Pemicu HTTP untuk membuat fungsi Lambda terpisah yang dipicu oleh peristiwa penyimpanan.

Peristiwa Penyimpanan

Di bagian ini, kita akan menggunakan peristiwa penyimpanan untuk memicu fungsi Lambda yang merespons permintaan PUT. Tujuan aplikasi ini akan dibangun dari contoh resep kita sebelumnya. Sekarang, kita ingin menyediakan gambar resep kita beserta deskripsi, makanan, dan waktu persiapan. Untuk melakukannya, kita akan menggunakan S3 (Simple Storage Service) sebagai pemicu, dan fungsi Lambda yang menambahkan URL gambar yang baru diunggah ke resep yang terkait dengannya.

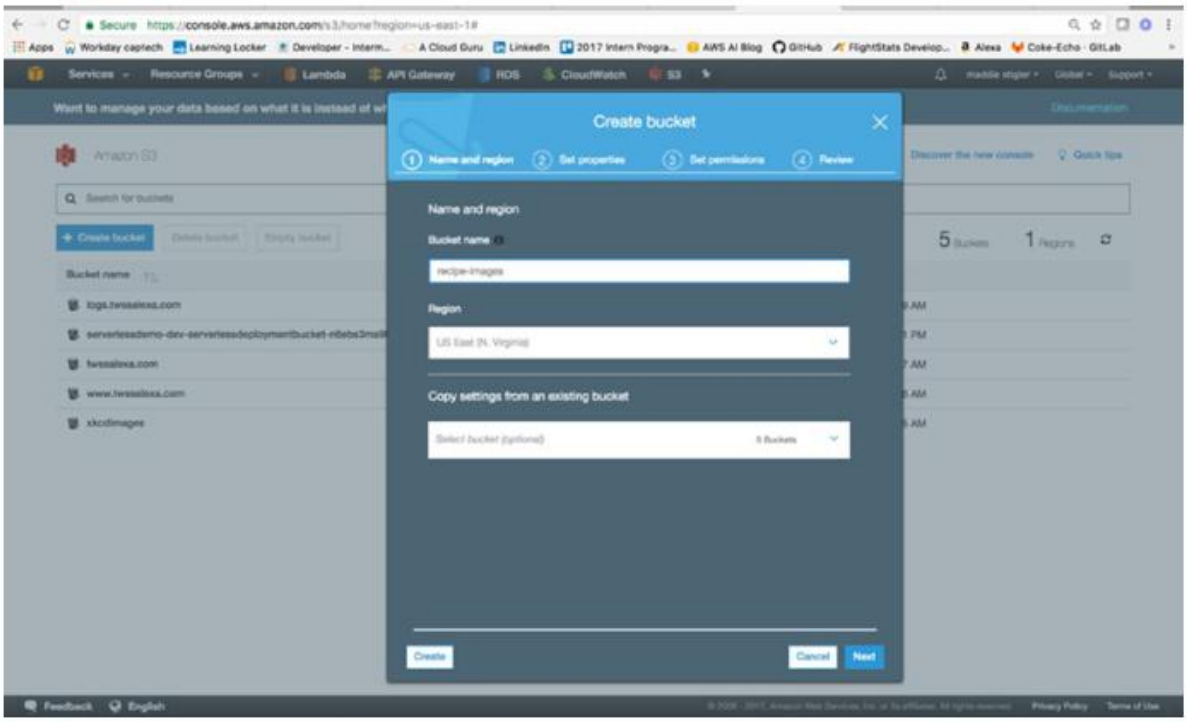
Amazon S3

AWS menawarkan banyak opsi penyimpanan mulai dari Basis Data Relasional, Basis Data NoSQL, hingga penyimpanan Blob. Dalam latihan ini, kita akan mengeksplorasi penggunaan AWS S3, penyimpanan blob, sebagai pemicu fungsi Lambda. Di dalam layanan S3, mari buat bucket. Konsol S3 Anda saat ini akan terlihat seperti Gambar 3.34.



Gambar 3.34 Konsol S3 Memungkinkan Anda Melihat Semua Bucket Yang Saat Ini Aktif, Membuat Bucket Baru, Dan Mencari Bucket

Seperti yang ditunjukkan pada Gambar 3.35, saya menamai bucket saya recipe-images-ms. Nama bucket bersifat unik secara universal, jadi Anda harus memastikan nama bucket Anda belum pernah digunakan sebelumnya. Yang saya maksud dengan "universal" adalah di semua wilayah dan akun AWS. Dalam pengaturan S3 saat mengonfigurasi bucket, Anda akan diberikan opsi seperti Versioning, Logging, dan Tags. Untuk saat ini, kami akan membiarkan nilai default. Satu-satunya hal yang akan kami ubah adalah izin publik. Kami akan membuat objek dalam bucket ini terbuka untuk umum sehingga kami dapat mengakses gambar resep ini dari web.

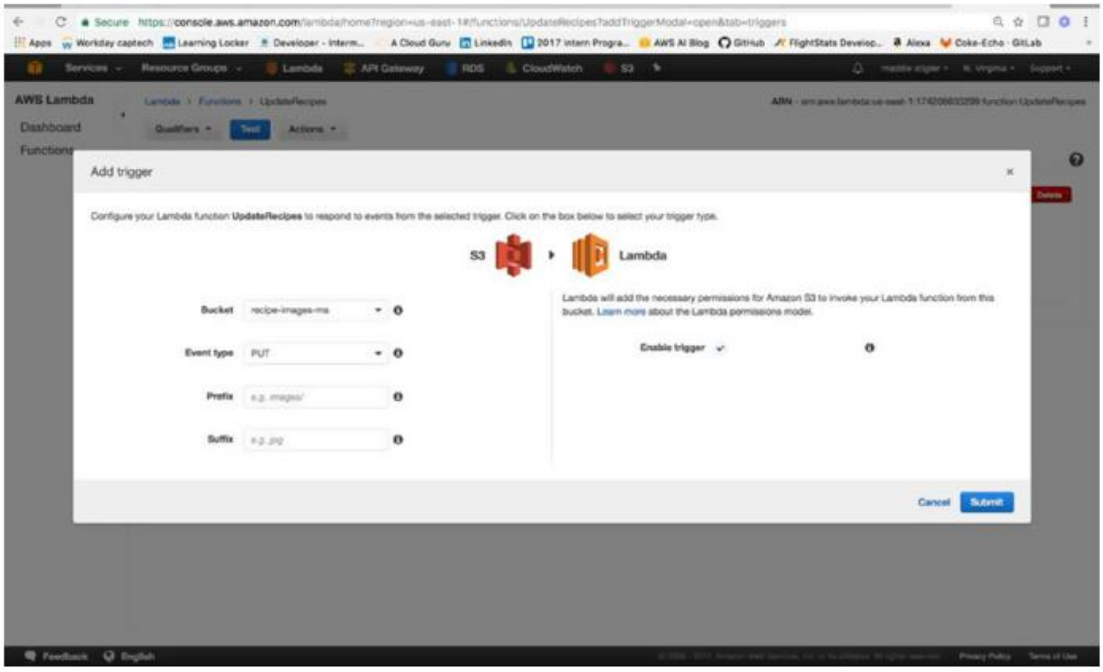


Gambar 3.35 Opsi Buat Bucket Memungkinkan Anda Mengonfigurasi Bucket Penyimpanan Blob Dengan Cepat. Anda Juga Dapat Menyesuaikan Pengaturan Ini Setelah Membuat Bucket

Setelah bucket dan gambarnya dikonfigurasi, kita dapat melanjutkan untuk menetapkan S3 sebagai pemicu dan membuat fungsi Lambda.

3.7 MENGGUNAKAN S3 SEBAGAI PEMICU

Dari konsol Lambda, buat fungsi Lambda yang disebut UpdateRecipe. Fungsi Lambda ini akan menerima peristiwa dari S3 saat objek diunggah (PUT). Fungsi ini kemudian akan memperbarui resep objek terkait dengan URL gambar. Untuk mempermudah, kita akan memberi nama unggahan gambar dengan kunci resep terkait. Misalnya, resep untuk Makaroni & Keju memiliki Id 2. Untuk mengaitkan URL gambar dengan resep tersebut, kita akan mengunggah file bernama 2. Di dalam konsol Lambda, konfigurasikan pemicu Anda menjadi S3, dan pilih bucket yang Anda buat sebelumnya. Pemicu Anda akan tampak seperti Gambar 3.36.



Gambar 3.36 S3 Dikonfigurasi Untuk Memicu Update Lambda Pada Permintaan Put. Ini Berarti Lambda Akan Dipicu Setiap Kali Objek Diunggah Ke Bucket S3 Yang Kita Tentukan.

Sekarang setelah peristiwa S3 ditetapkan untuk memicu Lambda, kita dapat memformat fungsi kita untuk menangani peristiwa tersebut sesuai keinginan kita. Langkah pertama adalah memahami permintaan peristiwa yang masuk. Untuk mensimulasikan permintaan ini, saya menggunakan cetak biru Set Test Event untuk operasi S3 PUT. JSON berikut adalah yang disediakannya:

```
{
"Records": [
"eventVersion": "2.0",
"eventTime": "1970-01-01T00:00:00.000Z",
"requestParameters": {
"sourceIPAddress": "127.0.0.1"
},
"s3": {
"configurationId": "testConfigRule",
"object": {
"eTag": "0123456789abcdef0123456789abcdef",
"sequencer": "OA1B2C3D4E5F678901",
"key": "HappyFace.jpg",
"size": 1024
},
"bucket": {
"arn": "arn:aws:s3 ::: mybucket",
"name": "sourcebucket",
```

```
"ownerIdentity": {
"principalId": "EXAMPLE"
}
},
"s3SchemaVersion": "1.0"
},
"responseElements": {
"x-amz-id-2":
"EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDE
FGH",
"x-amz-request-id": "EXAMPLE123456789"
},
"awsRegion": "us-east-1",
"eventName": "ObjectCreated:Put",
"userIdentity": {
"principalId": "EXAMPLE"
},
"eventSource": "aws:s3"
}
]
}
```

Kita dapat memformat Lambda kita untuk menangani permintaan ini dengan membuat model TypeScript untuk permintaan S3 PUT. Kita ingin menentukan kunci objek sehingga kita dapat menguraikannya dan mengambil item DynamoDB terkait untuk menempatkan URL gambar dengan resep yang benar. Bagian berikutnya akan membahas bagaimana fungsi Lambda kita menangani permintaan ini.

Respons terhadap Pemicu

Seperti yang kita lakukan dengan permintaan HTTP, kita juga akan mengandalkan AWS SDK untuk memperbarui tabel DynamoDB kita. Kita ingin mengambil kunci objek dari peristiwa masuk kita dan menentukannya sebagai kunci kita ke tabel DynamoDB. Setelah kita memiliki item yang kita inginkan dari tabel kita, kita dapat membuat perubahan padanya, seperti menambahkan atribut imageUrl dan melampirkan URL publik ke gambar makanan.

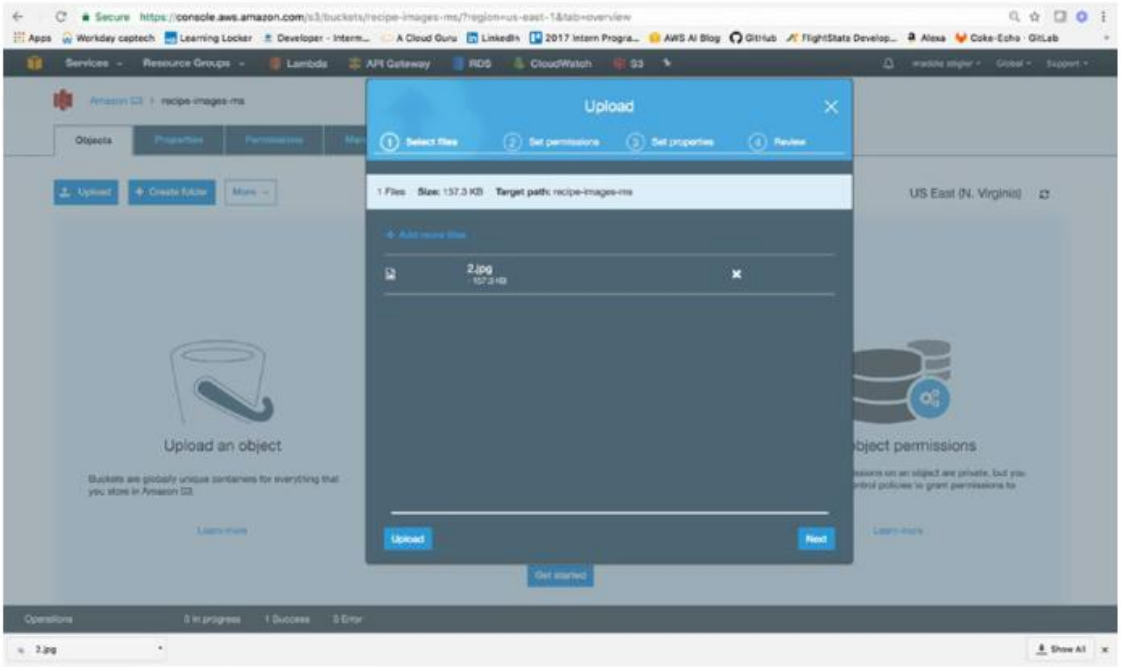
```
'use strict';
exports ._ esModule = true;
var AWS = require("aws-sdk");
module.exports.UpdateRecipe = function (event, context,
callback) {
console.info("Received event: ", JSON.stringify(event, null,
2));
var docClient = new AWS.DynamoDB.DocumentClient();
var table = process.env.TABLE_NAME;
```

```
var image = event.Records[o].s3.object.key.split('.');
var id = parseInt(image[o]);
// Update the item, unconditionally,
var params = {
TableName: table,
Key: {
"Id": id
},
UpdateExpression: "set ImageURL =: iurl",
ExpressionAttributeValues: {
":iurl":     "https://s3.amazonaws.com/recipe-images-ms/"     +
event.Records[o].s3.object.key
},
ReturnValues: "UPDATED_NEW"
};
var response = {
statusCode: 200,
message:
};
console.log("Updating the item ... ");
docClient.update(params, function (err, data) {
if (err) {
response.statusCode = 500;
console.error("Unable   to   update   item.   Error   JSON:",
JSON.stringify(err, null, 2));
response.message = "Unable to update";
callback(null, response);
}
else {
console.log("UpdateItem succeeded:", JSON.stringify(data, null,
2));
response.message = "Updated recipe successfully.";
callback(null, response);
}
});
};
```

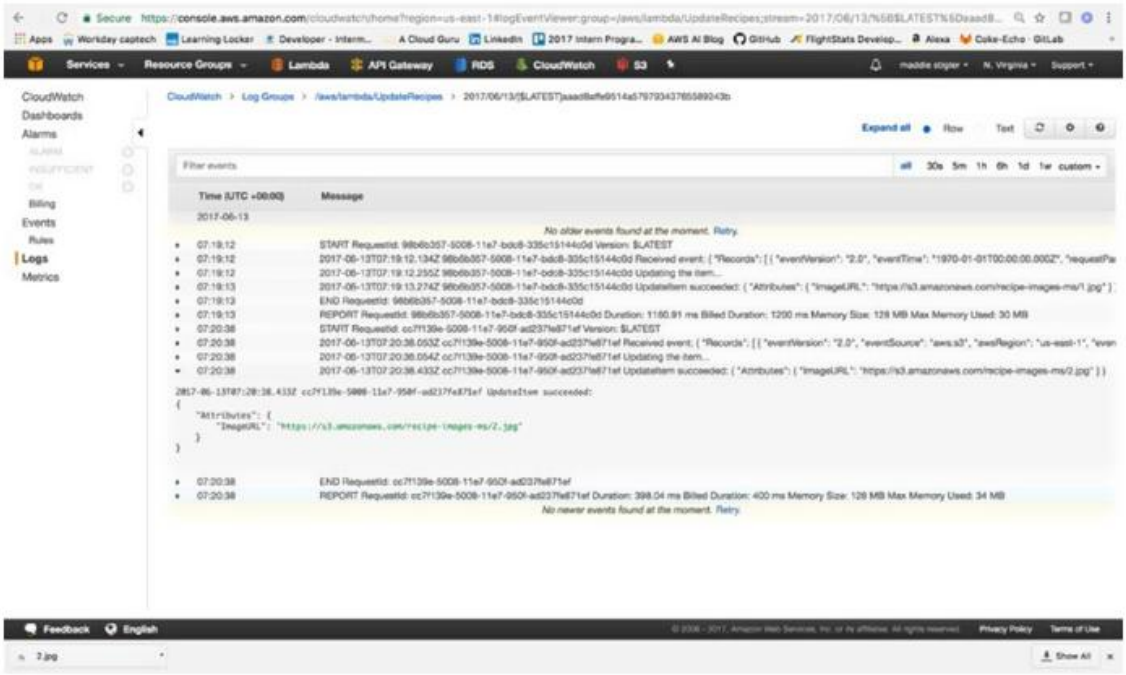
Langkah-langkah berikut merangkum proses ini:

1. Mengurai permintaan masuk untuk mengumpulkan kunci dari gambar.
2. Mengatur ID Gambar ke ID Kunci untuk mencari DynamoDB dan menemukan resep yang benar.
3. ARN bucket Anda adalah awal dari URL gambar objek S3. Gunakan ini dan nama gambar untuk mengatur URL di tabel DynamoDB.
4. Terakhir, kembalikan status eksekusi.

Untuk menguji keberhasilan fungsi kita setelah mengompresnya dengan folder Bersama dan modul node, kita dapat mengunggah gambar ke bucket S3 kita. Saya menemukan gambar makaroni & keju, salah satu resep saya, dan mengunggahnya ke bucket. Untuk mengunggah ke bucket, cukup buka lokasi bucket dan klik Unggah. Perintah tersebut ditunjukkan pada Gambar 3.37.

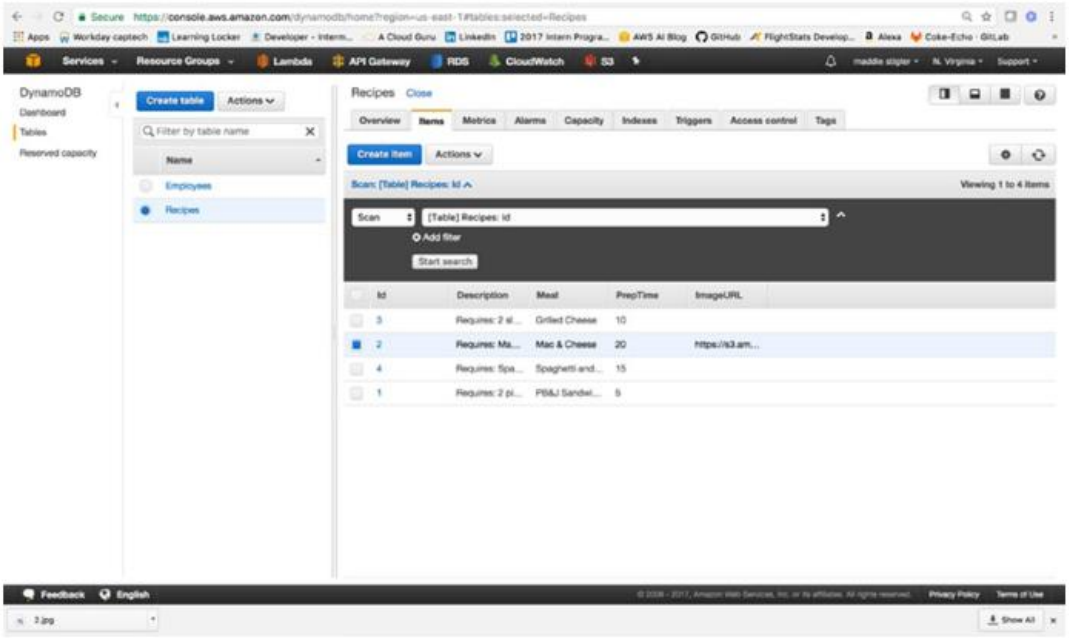


Gambar 3.37 Anda Dapat Mengunggah Berkas Dalam Konsol S3 Dan Mengonfigurasi Izinnya Dengan Cepat



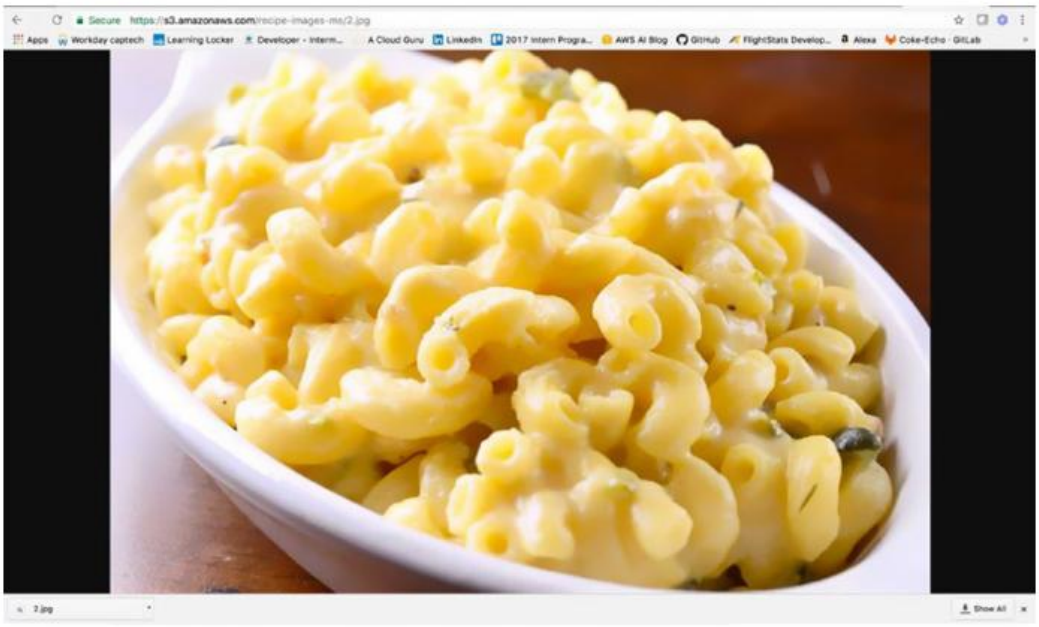
Gambar 3.38 Hasil Log Yang Berhasil Dari Unggahan Kami Di Cloudwatch

Kami juga dapat menguji unggahan dengan melihat langsung pada tabel DynamoDB. Seperti yang ditunjukkan pada Gambar 3.39, ImageUrl sekarang ditambahkan sebagai atribut dengan URL gambar yang diberikan.



Gambar 3.39 Atribut Imageurl Telah Ditambahkan Ke Tabel Bersama Dengan URL

Selain itu, saat Anda membuka URL gambar yang diberikan, Anda akan diarahkan ke gambar resep (Gambar 3.40).



Gambar 3.40 Gambar Dapat Diakses Publik Dengan URL S3

Seperti sebelumnya, ada banyak cara untuk meningkatkan fungsi Lambda ini. Pemisahan logika penyedia sangat dianjurkan serta penerapan dengan Serverless Framework. Kode untuk bagian latihan ini dapat ditemukan di *https://github.com/mgstigler/Serverless/tree/master/AWS/aws-service/ StorageTrigger*

3.8 KESIMPULAN

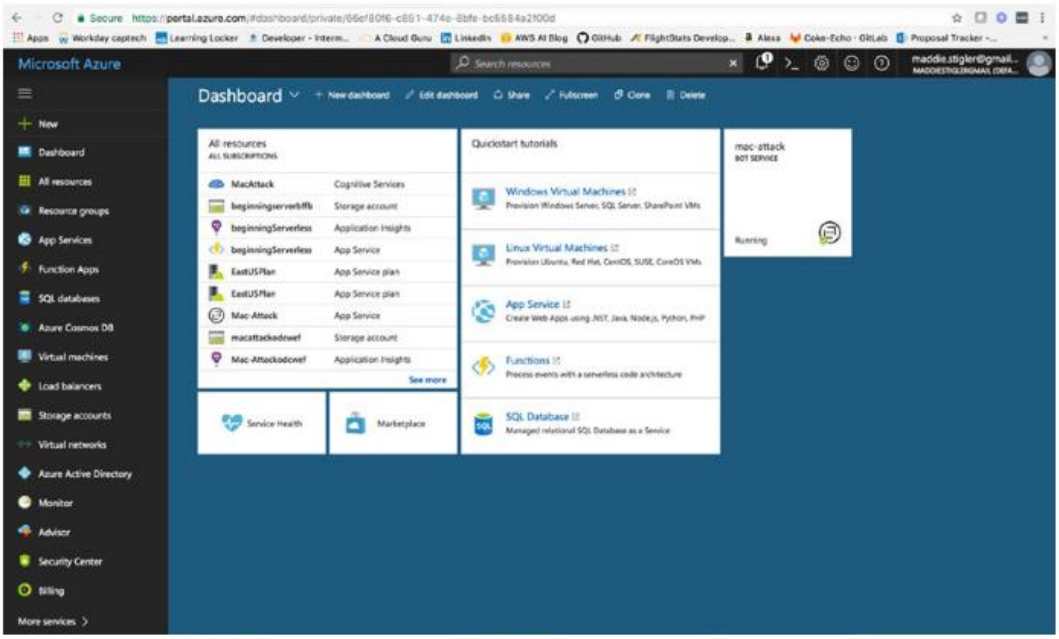
Dalam bab ini, kami menjelajahi dua aplikasi tanpa server dengan AWS Lambda. Anda mempelajari cara menavigasi konsol, mengonfigurasi fungsi Lambda, dan menetapkan pemicu, serta cara membangun beberapa layanan untuk memicu dan merespons fungsi Lambda. Sekarang Anda seharusnya memiliki pemahaman yang baik tentang cara kerja fungsi Lambda dan kekuatannya dalam AWS. Anda juga seharusnya memiliki pemahaman yang lebih aplikatif tentang fungsi tanpa server secara keseluruhan dan cara penggunaannya. Dalam bab berikutnya, kita akan menjelajahi UI Azure dan membangun fungsi tanpa server dalam Azure.

BAB 4
Azure

4.1 PENDAHULUAN

Dalam bab ini, kita akan memanfaatkan Azure Functions untuk membuat beberapa aplikasi tanpa server. Sebelumnya, kita telah mempelajari AWS dan membuat dua fungsi, satu dipicu oleh unggahan bucket S3 dan satu dipicu oleh permintaan HTTP. Kita akan membuat ulang kedua pemicu ini dengan contoh dunia nyata yang berbeda menggunakan layanan Azure. Ini akan memberi kita gambaran tentang sumber daya Azure dan perbedaannya dengan AWS. Ini juga akan memungkinkan kita untuk mengeksplorasi cara lain untuk membuat dan menggunakan aplikasi tanpa server. Untuk latihan berikut, kita akan menggunakan fungsi Azure, WebHooks, pemicu API, dan Azure Queue Storage. Di akhir bab ini, kita akan memiliki tiga aplikasi tanpa server dan pengalaman dengan beberapa layanan Azure serta pemahaman yang lebih baik tentang pemilihan penyedia cloud.

Seperti yang kami lakukan dengan AWS, sebelum kami mulai menulis aplikasi, kami akan membahas UI Azure dan cara menavigasinya, beserta berbagai opsi harga, dan portal Functions.



Gambar 4.1 Azure memberi Anda ikhtisar tentang sumber daya dan layanan yang sedang berjalan. Azure juga memberi Anda akses untuk membuat sumber daya baru di panel kiri.

Saya adalah seseorang yang mulai mengembangkan di AWS dan menjadi sangat nyaman dengan lingkungan dan UI yang disediakan Amazon. Pada akhirnya, beralih dari AWS ke Azure bukanlah hal yang sulit, tetapi antarmukanya sedikit berbeda, dan itu bisa jadi sedikit sulit untuk dibiasakan. Untuk membantu meringankan perbedaan ini, saya akan

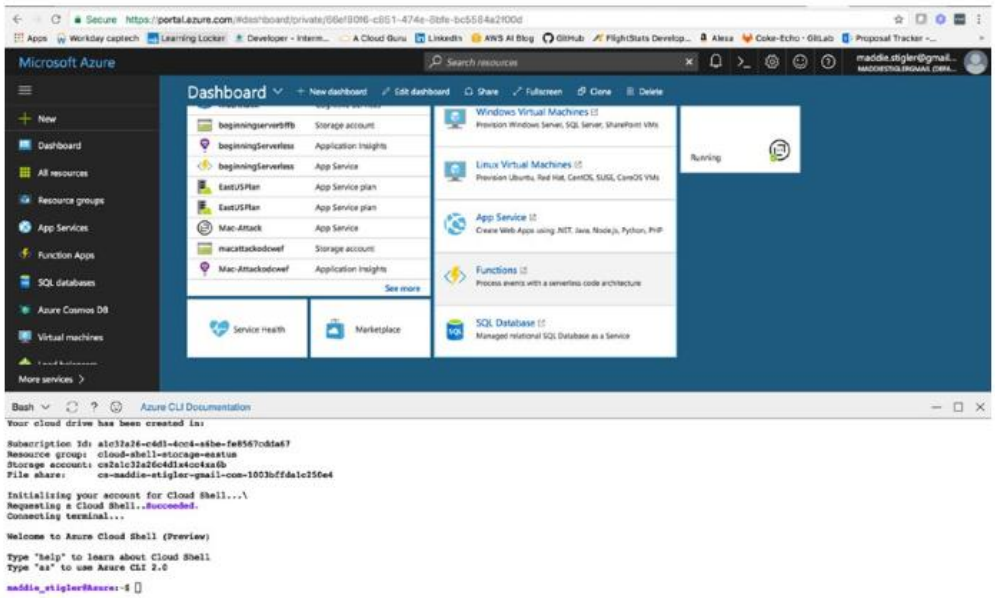
membahasnya dengan menunjukkan berbagai perbandingan dan perbedaan antara keduanya. Untuk masuk ke portal Azure, buka http://www.portal.azure.com. Setelah masuk, Anda akan diarahkan ke dasbor Anda. Dasbor memberi Anda ikhtisar tentang sumber daya Anda, beberapa kiat dan trik, dan layanan Anda yang sedang berjalan. Gambar 4.1 menunjukkan contoh Dasbor.

Hal yang sangat saya sukai dari dasbor Azure adalah kemampuannya untuk disesuaikan. Dengan mengeklik opsi Edit Dasbor di bagian atas, Anda dapat dengan mudah menambahkan sumber daya lain, memindahkan sumber daya, dan menambahkan petak. Petak hanyalah satu blok di dasbor Anda (contohnya termasuk Semua Sumber Daya, Kesehatan Layanan, dan Marketplace pada Gambar 4.1). Galeri Petak adalah alat yang memungkinkan Anda mencari petak untuk sumber daya tertentu dan menyeretnya ke bilah Anda saat ini (seperti yang akan Anda lihat sebentar lagi, UI untuk sumber daya tertentu). Melalui ini, Anda dapat membuat tampilan manajemen yang mencakup sumber daya. Anda juga memiliki kemampuan untuk membuat dan melihat beberapa dasbor. Saya suka menggunakan dasbor yang berbeda untuk proyek yang berbeda sehingga saya dapat memisahkan semua sumber daya saya secara visual. Anda dapat memberi setiap dasbor nama proyek agar tetap teratur.

Navigasi

Kami telah menjelajahi banyak kemampuan dasbor, tetapi ada banyak hal menarik lainnya langsung di halaman portal yang akan bermanfaat bagi kami untuk dipahami sejak awal. Seperti di AWS, ada ikon lonceng di bilah hitam atas yang memberi tahu kami pembaruan layanan apa pun dari Azure. Di sebelah lonceng, ada Cloud Shell. Ini adalah fitur yang sangat saya sukai, jadi saya akan meluangkan waktu untuk membahasnya. Gambar 4.2 menunjukkan layar awal Cloud Shell.

Saat ini, Anda dapat menggunakan Shell untuk menjalankan perintah Bash atau mengalihkannya ke pratinjau PowerShell. Penulis menduga opsi PowerShell akan segera hadir, tetapi untuk saat ini, pratinjau sudah cukup. Dengan mengetik az, Anda akan mendapatkan Azure CLI lengkap. Ini adalah fitur yang luar biasa. Kemampuan untuk mengambil CLI dalam satu perintah langsung dari portal membuat segalanya jauh lebih mudah. Bagi saya, ini selangkah lebih maju dari sekadar mengandalkan apa pun secara lokal dan selanjutnya menempatkannya di lokasi penyedia cloud. Ini adalah sesuatu yang mungkin akan Anda perhatikan dengan Azure seiring berjalannya waktu; Microsoft telah melakukan pekerjaan yang hebat dengan menghapus sebanyak mungkin dari ruang kerja lokal. Saya perkirakan ini akan terus menjadi tren dan akan menjadi sesuatu yang akan diikuti oleh penyedia cloud lainnya, seperti AWS dan Google.



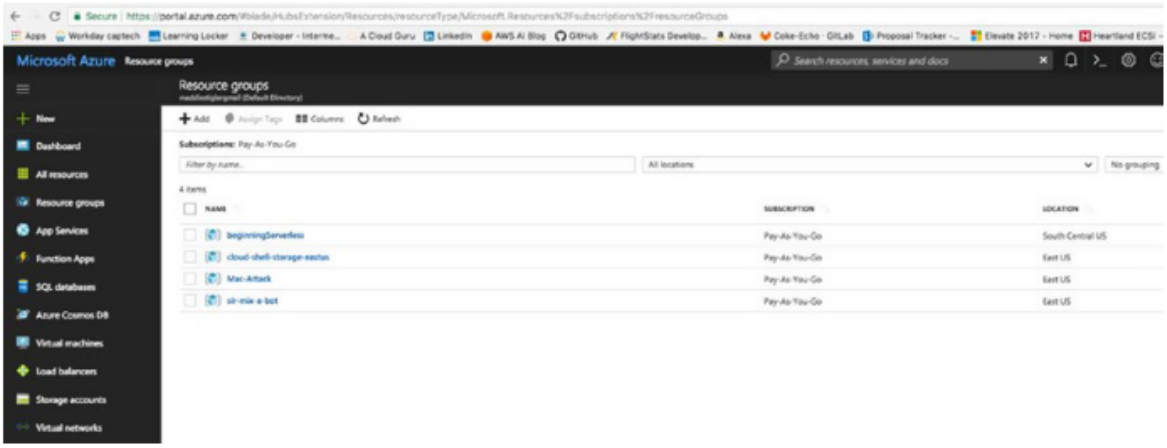
Gambar 4.2 Saat Anda mengeklik ikon Cloud Shell, Azure akan menampilkan Bash Shell yang disertai Azure CLI dan banyak fitur lainnya.

Azure memberi kita daftar beberapa hal yang perlu diingat saat menggunakan Azure Cloud Shell:

- ✓ Cloud Shell akan habis masa berlakunya setelah 10 menit tanpa aktivitas interaktif.
- ✓ Cloud Shell hanya dapat diakses dengan file share yang dilampirkan.
- ✓ Cloud Shell diberi satu mesin per akun pengguna.
- ✓ Izin ditetapkan sebagai pengguna Linux biasa.
- ✓ Cloud Shell berjalan pada mesin sementara yang disediakan per sesi, per pengguna.

Pada akhirnya, kami mendapatkan banyak hal langsung dari shell berbasis browser yang memungkinkan kami mengembangkan dengan cepat tanpa menghabiskan banyak waktu untuk pengaturan dan instalasi lingkungan. Ide ini sejalan dengan banyak hal yang dimaksud dengan komputasi tanpa server. Penyedia cloud hadir untuk memudahkan proses pengembangan dengan menyediakan sebanyak mungkin hal yang mereka bisa.

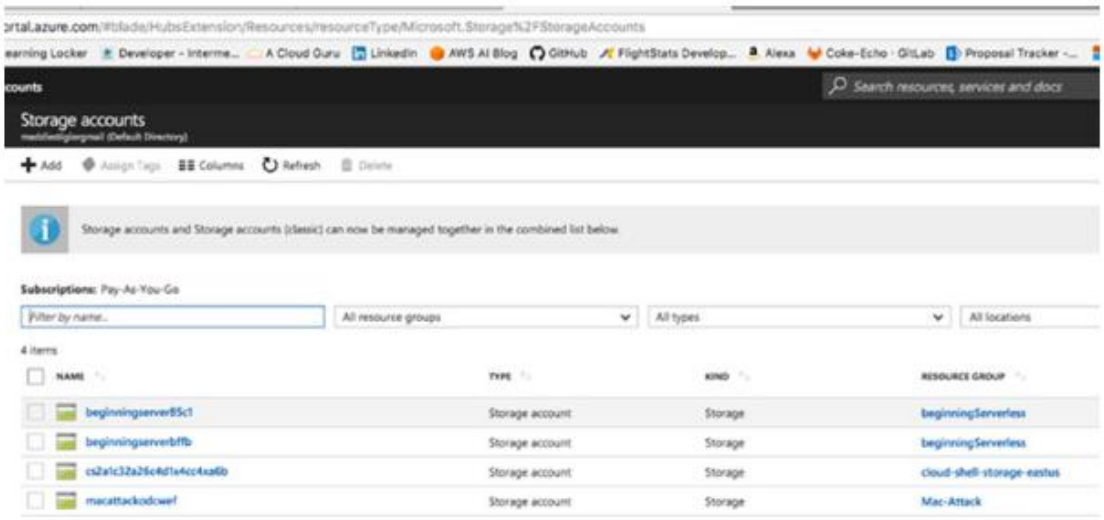
Beberapa poin navigasi penting lainnya yang menarik termasuk grup sumber daya, akun penyimpanan, penagihan, dan aplikasi fungsi. Meskipun AWS memberi Anda kemampuan untuk membuat grup sumber daya, penerapannya jauh lebih sedikit. Sebaliknya, membuat sumber daya apa pun di Azure mengharuskan Anda menetapkannya ke grup sumber daya. Meskipun terkadang agak merepotkan untuk membuat grup sumber daya saat Anda benar-benar mencoba menjalankan aplikasi, menurut saya itu ide yang bagus. Grup sumber daya pada dasarnya adalah wadah yang digunakan untuk menampung semua sumber daya terkait untuk suatu solusi. Azure juga menggunakan konsep blade untuk menjelajahi berbagai sumber daya. Blade pada dasarnya adalah UI untuk sumber daya tertentu. Gambar 4.3 menunjukkan blade Grup Sumber Daya yang disediakan portal Azure.



Gambar 4.3 Bilah Resource Groups memungkinkan Anda untuk melihat semua resource group, membuat resource group lain, dan mengatur resource group Anda saat ini.

Di portal ini, saya memiliki resource group untuk memulai serverless, yang berisi semua resource saya untuk contoh serverless pertama kami, penyimpanan cloud shell (ini diperlukan untuk menggunakan Cloud Shell), lalu beberapa bot storage group. Anda juga dapat memberi tag yang berbeda pada resource group yang berbeda untuk mengurutkannya secara lebih umum. Sebagian orang menggunakan ini sebagai cara untuk memisahkan lingkungan pengembangan yang berbeda. Contohnya adalah membuat resource group untuk solusi Anda yang berbeda dan menandainya sebagai Dev, QA, Production, atau apa pun yang Anda pilih. Anda kemudian dapat memfilternya di masa mendatang berdasarkan lingkungan pengembangan tertentu.

Resource Storage Account (Gambar 4.4) adalah layanan Azure lain yang akan sering Anda akses saat menggunakan Azure sebagai penyedia cloud Anda. Mirip dengan resource group, sebagian besar layanan Azure mengharuskan Anda untuk menetapkannya ke storage group. Menurut pengalaman saya, ini sebenarnya menyederhanakan banyak hal di kemudian hari karena Anda tahu persis di mana semuanya berakhir dan cara memantau dan mengonfigurasinya. Salah satu aspek rumit dari AWS adalah Amazon menangani banyak pengaturan ini untuk Anda, dan jika Anda tidak terbiasa dengan penyediaan dan konfigurasi, mungkin ada grup keamanan dan pengaturan acak yang tidak begitu mudah ditemukan. Dengan membuat elemen-elemen ini khusus untuk grup sumber daya Anda, Anda tahu persis di mana harus melakukan perubahan konfigurasi.

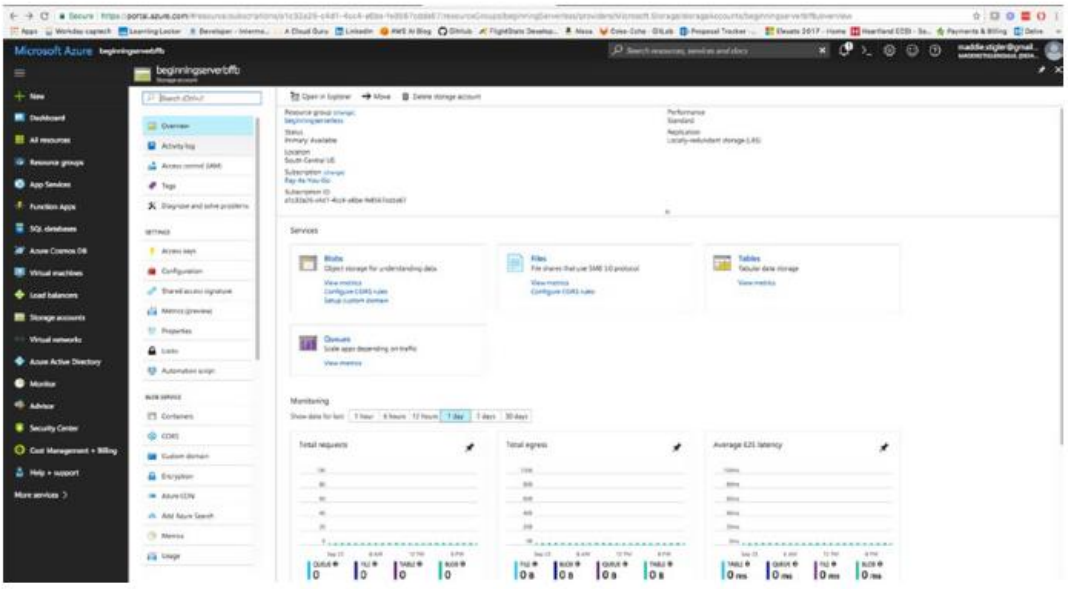


Gambar 4.4 Storage Accounts Blade memungkinkan Anda melihat semua grup penyimpanan dan grup sumber daya yang terkait dengannya

Seperti yang Anda lihat, saya memiliki akun penyimpanan yang terkait dengan setiap grup sumber daya yang saya lihat sebelumnya. Semua objek dalam akun penyimpanan ditagih bersama. Azure menyarankan Anda untuk mengingat hal berikut saat membuat akun penyimpanan:

- ✓ Jenis akun mengacu pada apakah Anda menggunakan akun penyimpanan tujuan umum atau akun penyimpanan Blob. Dengan akun penyimpanan Blob, tingkatan akses juga menentukan model penagihan untuk akun tersebut.
- ✓ Kapasitas penyimpanan mengacu pada seberapa banyak jatah akun penyimpanan yang Anda gunakan untuk menyimpan data.
- ✓ Replikasi menentukan berapa banyak salinan data Anda yang disimpan pada satu waktu, dan di lokasi mana. Penting juga untuk dicatat bahwa harga berbeda untuk beberapa opsi replikasi ini. Sebaiknya pertimbangkan sebelum membuat keputusan dan mendapatkan tagihan yang tidak Anda harapkan.
- ✓ Transaksi mengacu pada semua operasi baca dan tulis ke Azure Storage. • Data egress mengacu pada data yang ditransfer keluar dari wilayah Azure. Saat data di akun penyimpanan Anda diakses oleh aplikasi yang tidak berjalan di wilayah yang sama, Anda akan dikenai biaya untuk data egress. (Untuk layanan Azure, Anda dapat mengambil langkah-langkah untuk mengelompokkan data dan layanan Anda di pusat data yang sama untuk mengurangi atau menghilangkan biaya data egress).
- ✓ Wilayah mengacu pada wilayah geografis tempat akun Anda berada.

Akun Layanan untuk akun tertentu memberi Anda opsi termasuk melihat layanan yang disertakan dalam akun, menambahkan kunci akses, menyediakan metrik pada sumber daya Anda, dan memberi Anda akses ke layanan seperti Tabel, Antrean, Blob Storage, dan VM Azure. Gambar 4.5 memberi Anda ikhtisar dari semua opsi ini.



Gambar 4.5 Storage Accounts Blade juga memberi Anda wawasan tentang akun penyimpanan tertentu dengan berbagai tindakan dan metrik pada akun itu sendiri

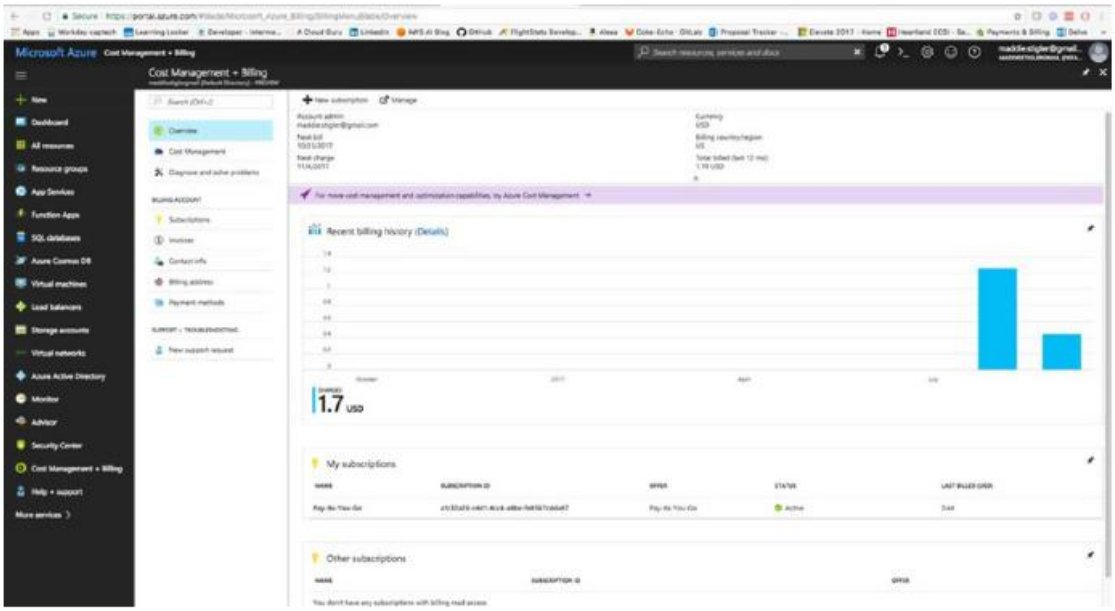
Anda juga diberikan titik akhir akun penyimpanan agar dapat mengakses sumber daya di akun penyimpanan Anda dengan mudah dan cepat. Setiap alamat URL bersifat unik untuk objek yang disimpan di Azure Storage. Nama akun membentuk subdomain dari alamat tersebut, jadi kombinasi subdomain dan nama domain membentuk titik akhir untuk akun penyimpanan Anda. Misalnya, grup sumber daya tanpa server awal saya berada di bawah akun penyimpanan beginningserverbffb.

- Layanan Blob: http://beginningserverbffb.blob.core.windows.net
- Layanan Tabel: http://beginningserverbffb.table.core.windows.net
- Layanan File: http://beginningserverbffb.file.core.windows.net
- Layanan Antrean: http://beginningserverbffb.queue.core.windows.net

Selain penyimpanan dan sumber daya, kita juga akan melihat cara mengakses harga dan penagihan dari portal Azure.

Harga

Dari bilah Penagihan (Gambar 4.6), kita memiliki akses ke informasi penagihan yang sangat umum seperti faktur, metode pembayaran, dan langganan.



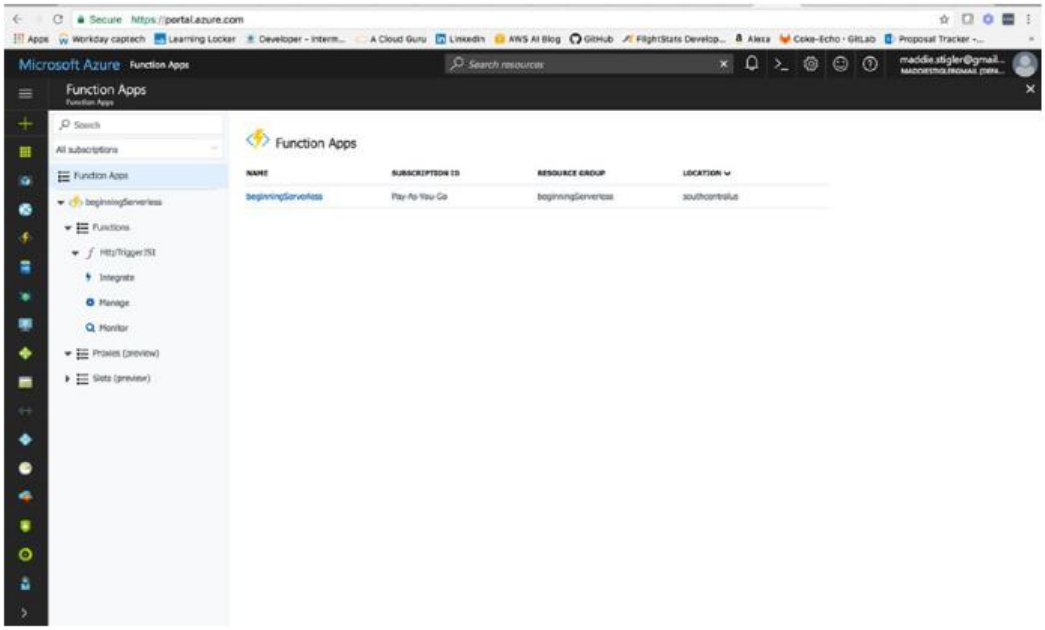
Gambar 4.6. Bilah Penagihan memberi Anda gambaran umum yang sangat umum tentang informasi penagihan dan informasi langganan Anda

Saya pribadi lebih suka portal Penagihan di AWS daripada Azure, terutama karena aksesibilitas dan kemudahan penggunaan dibandingkan dengan fungsionalitas sebenarnya. Di AWS, kami dapat melakukan semua hal dalam manajemen penagihan kami dari portal yang sama. Di Azure, Anda diberikan gambaran yang sangat mendasar dan diminta untuk membuka halaman akun Azure yang sebenarnya (di portal yang berbeda) untuk mengelola penagihan akun Anda. Perbedaan lainnya adalah bahwa di Azure, Anda dapat menetapkan batas penagihan dan menghentikan item Anda jika mencapai batas tersebut. Ini berbeda dengan di AWS di mana Anda hanya dapat mengatur alarm penagihan.

Dari langganan sebenarnya dalam penagihan Anda, Anda dapat melihat berbagai metrik seperti biaya berdasarkan sumber daya, analisis biaya, dan penggunaan. Anda juga dapat mengontrol IAM, pemindahan langganan, dan sumber daya yang terkait dengan langganan tertentu.

4.2 AZURE FUNCTIONS

Azure Function Apps dapat diakses dari panel Sumber Daya samping. Saat Anda membukanya, Anda diarahkan ke bilah Fungsi. Blade ini menampilkan semua aplikasi fungsi dan struktur proyek masing-masing. Dalam Gambar 4.7 menunjukkan blade Fungsi.



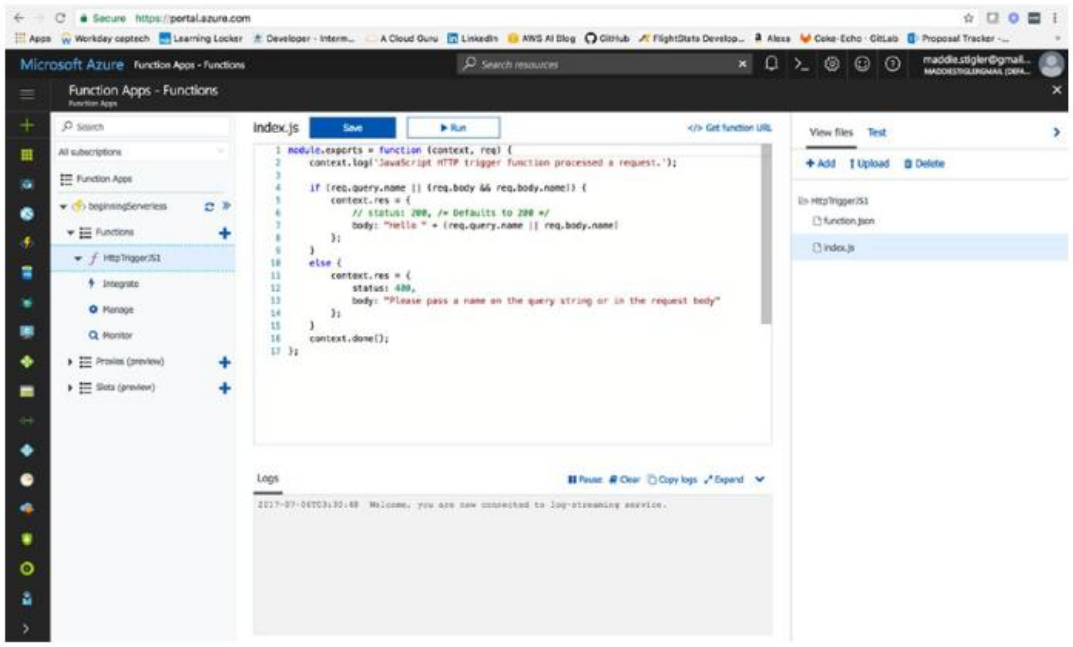
Gambar 4.7 Blade Aplikasi Fungsi dengan semua fungsi dan struktur proyek Anda

Saya enggan menggunakan fungsi Azure karena saya telah melakukan banyak pengembangan dengan AWS Lambda dan pengembangan cloud dalam lingkup AWS. Namun, setelah menjelajahi Functions sebentar, saya dengan cepat menjadi penggemar berat. Salah satu keluhan saya tentang AWS Lambda adalah ketidakmampuan untuk melihat dan menavigasi struktur proyek secara efisien. Azure memecahkan masalah ini dengan memberi Anda cara untuk mengakses struktur proyek Anda, membuat perubahan dan mengonfigurasi aplikasi tanpa server Anda dengan mudah, dan melakukan semua ini di satu tempat.

Saya pikir ini adalah sesuatu yang akan tersedia dengan AWS seiring pertumbuhannya. Untuk saat ini, ini adalah fitur yang sangat membantu dalam fungsi Azure dan sesuatu yang seharusnya membuat pengembangan cepat menjadi lebih cepat. Azure juga menggabungkan gagasan input dan output, yang dapat dikonfigurasi dari blade Integrasi dalam fungsi tertentu Anda. Kami membahas input dan output secara singkat, tetapi pada dasarnya keduanya hanya memungkinkan Anda untuk mengikat data tambahan ke fungsi Anda (seperti mengambil dari tabel) saat dipicu. Gambar 4.8 menunjukkan struktur proyek fungsi Hello World sederhana yang akan kita bahas nanti.

Kita akan menjelajahi Function Apps dan semua kapabilitasnya secara lebih mendetail saat kita mulai membuat solusi. Namun, sebelum kita dapat mulai membuat fungsi, kita harus melihat keamanan Azure. Kita juga melakukan ini dengan AWS saat mempelajari IAM. Merupakan praktik yang baik untuk memiliki pemahaman yang baik tentang kapabilitas keamanan setiap penyedia sehingga Anda dapat membuat aplikasi yang aman namun dapat diakses. Azure membantu Anda lebih dari yang Anda harapkan dengan mengharuskan Anda untuk mengaitkan semua sumber daya Anda dengan aplikasi. Hal ini menghasilkan organisasi yang lebih baik dan lebih sedikit kebingungan saat mengonfigurasi izin. Saya cenderung sedikit

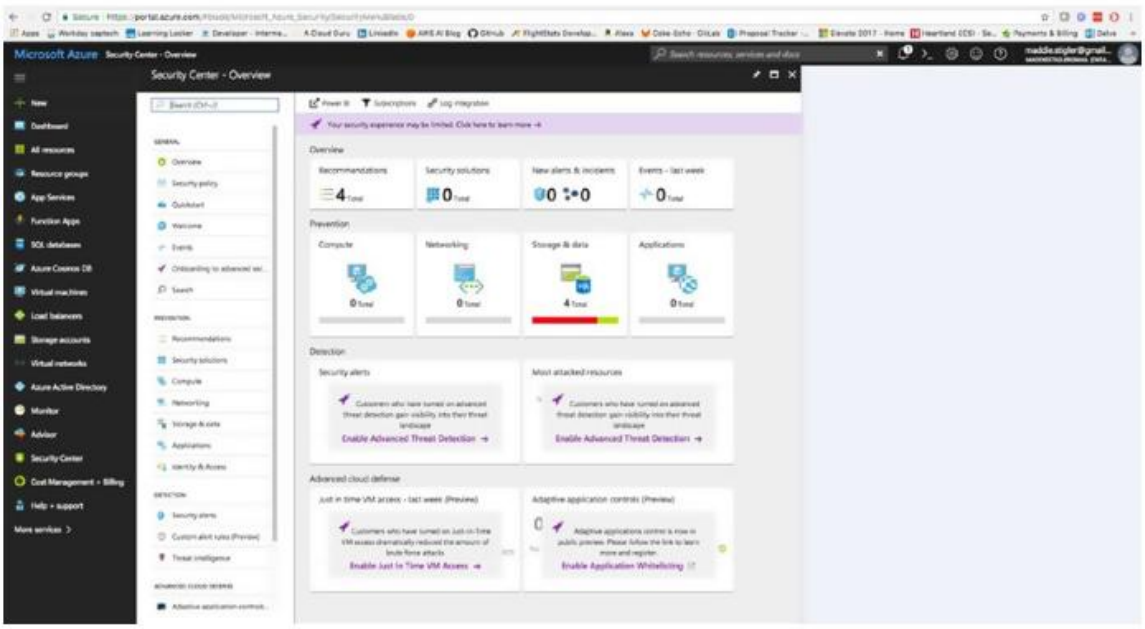
bingung saat memiliki banyak peran dan izin yang berbeda, jadi membuatnya dari awal dan mengetahui bahwa itu khusus untuk sumber daya yang saya buat sangat membantu saya.



Gambar 4.8 Azure Function Apps memungkinkan Anda melihat fungsi berdasarkan konfigurasinya, seperti yang terlihat di sebelah kiri. Aplikasi ini juga memungkinkan Anda melihat struktur proyek fungsi dan mengedit berbagai file yang ada di dalamnya.

4.3 KEAMANAN AZURE

Kita dapat menavigasi ke Keamanan Azure dengan mengeklik Hamburger dan membuka Layanan Lainnya dan Pusat Keamanan. Pusat Keamanan akan terlihat seperti dasbor pada Gambar 4.9.



Gambar 4.9 Azure Security Blade memberi Anda akses ke ikhtisar, pencegahan, deteksi, dan pertahanan cloud tingkat lanjut dengan akun dan layanan Anda

Azure Security Center membantu Anda mencegah, mendeteksi, dan menanggapi ancaman dengan peningkatan visibilitas dan kontrol atas keamanan sumber daya Azure Anda. Azure Security Center menyediakan pemantauan keamanan terintegrasi dan manajemen kebijakan di seluruh langganan Azure Anda, membantu mendeteksi ancaman yang mungkin tidak terdeteksi, dan bekerja dengan ekosistem solusi keamanan yang luas. Microsoft mencantumkan beberapa kemampuan utama:

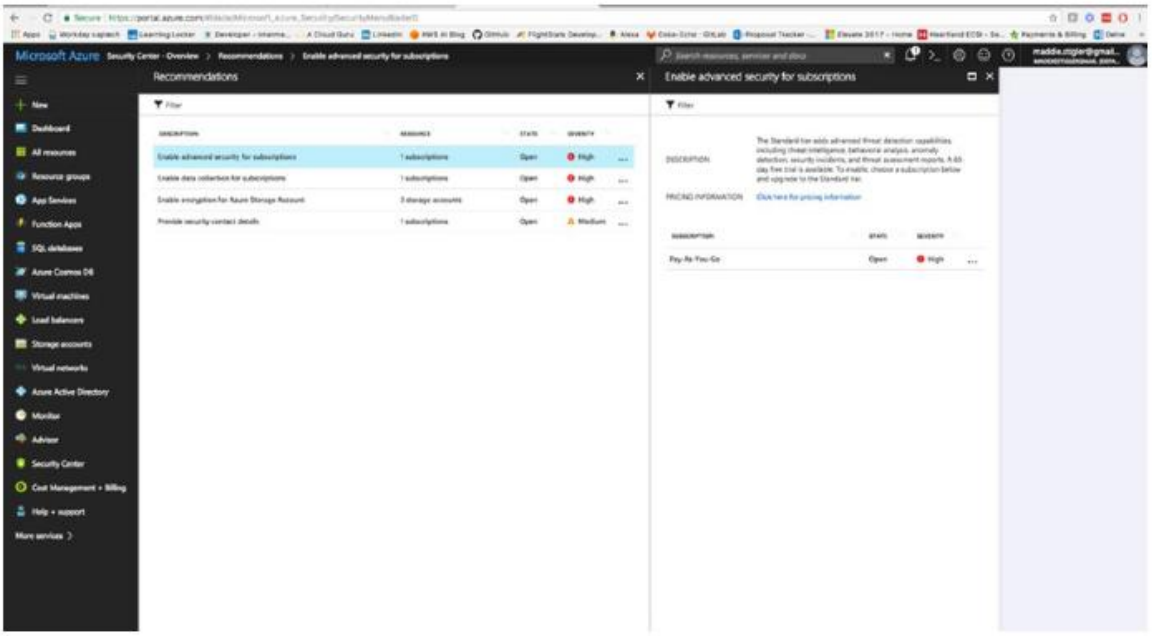
Tahapan keamanan azure	
Platform	**Kemampuan**
Mencegah	Memantau status keamanan sumber daya azure anda.
Mencegah	Menentukan kebijakan untuk langganan azure anda berdasarakan persyaratan keamanan perusahaan anda, jenis aplikasi yang anda gunakan, dan sensitivitas data anda.
Mencegah	Menggunakan rekomendasi keamanan berbasis kebijakan untuk memandu pemilik layanan melalui proses penerapan kontrol yang diperlukan
Mencegah	Dengan cepat menyebarkan layanan dan peralatan keamanan dari microsoft dan mitra
Deteksi	Secara otomatis mengumpulkan dan menganalisis data keamanan dari sumber daya azure anda, jaringan, dan solusi mitra seperti program antimalware dan firewall
Deteksi	Menggunakan intelijen ancaman global dari produk dan layanan microsoft, unit kejahatan digital microsoft (DCU), pusat respons keamanan microsoft (MSRC), dan umpan eksternal.

Deteksi	Menerapkan analisis tingkat lanjut, termasuk pembelajaran mesin dan analisis perilaku.
Menanggapi	Memberikan peringatan/insiden keamanan yang diprioritaskan.
Menanggapi	Memberikan wawasan mengenai sumber serangan dan sumber daya yang terkena dampak.
Menanggapi	Menyarankan cara untuk menghentikan serangan saat ini dan membantu mencegah serangan di masa mendatang.

Terapkan Rekomendasi

Pusat Keamanan menganalisis status keamanan sumber daya Azure Anda secara berkala. Saat Pusat Keamanan mengidentifikasi potensi kerentanan keamanan, ia membuat rekomendasi yang memandu Anda melalui proses konfigurasi kontrol yang diperlukan.

Rekomendasi yang disediakan oleh Azure ini adalah sesuatu yang harus Anda periksa secara rutin. Saya kembali setelah membuat aplikasi saya untuk bab ini dan menerapkan rekomendasi keamanan dari Pusat Keamanan. Gambar 4-10 menunjukkan apa saja rekomendasi saya dan bagaimana Anda dapat menemukannya dengan mudah di blade.



Gambar 4.10. Azure Security Blade memberi Anda akses ke ikhtisar, pencegahan, deteksi, dan pertahanan cloud tingkat lanjut dengan akun dan layanan Anda

Rekomendasi ditampilkan dalam format tabel, yang setiap barisnya mewakili satu rekomendasi tertentu. Kolom-kolom tabel ini adalah:

1) Deskripsi: Menjelaskan rekomendasi dan apa yang perlu dilakukan untuk mengatasinya.
2) Sumber Daya: Mencantumkan sumber daya yang menerapkan rekomendasi ini.
3) Status: Menjelaskan status rekomendasi saat ini:

- Terbuka: Rekomendasi belum ditangani.
- Sedang Berlangsung: Rekomendasi saat ini sedang diterapkan ke sumber daya, dan Anda tidak perlu melakukan tindakan apa pun.
- Telah Diselesaikan: Rekomendasi telah diselesaikan (dalam kasus ini, baris berwarna abu-abu).

4) Keparahan: Menjelaskan tingkat keparahan rekomendasi tertentu:
   - Tinggi: Kerentanan ada pada sumber daya yang penting (seperti aplikasi, VM, atau grup keamanan jaringan) dan memerlukan perhatian.
   - Sedang: Kerentanan ada dan langkah-langkah yang tidak kritis atau tambahan diperlukan untuk menghilangkannya atau untuk menyelesaikan suatu proses.
   - Rendah: Kerentanan ada yang harus ditangani tetapi tidak memerlukan perhatian segera. (Secara default, rekomendasi rendah tidak ditampilkan, tetapi Anda dapat memfilter rekomendasi rendah jika ingin melihatnya.)

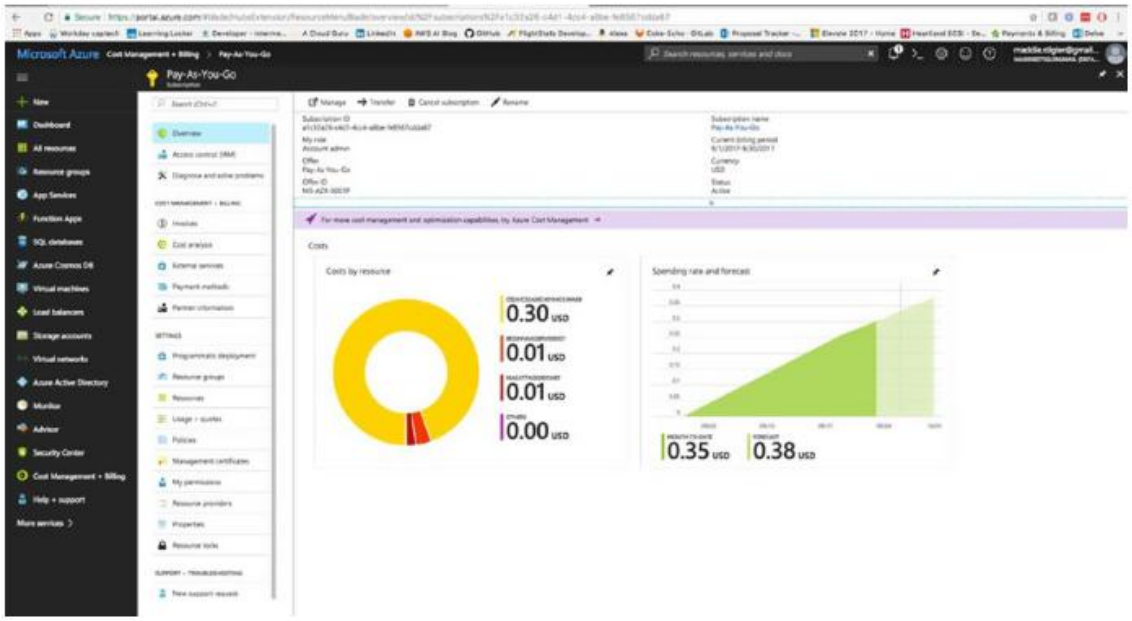
Beberapa rekomendasi saya berasal dari paket Bayar-Sesuai-Pemakaian saya. Azure ingin Anda menggunakan paket Standar, yang sedikit lebih mahal. Demi proyek ini, Anda dapat tetap menggunakan paket gratis dan kemudian beralih ke paket yang lebih tinggi saat Anda mulai membutuhkan lebih banyak dari paket Anda.

Tetapkan Kebijakan Keamanan

Hal terakhir yang ingin saya lihat di Pusat Keamanan sebelum beralih ke fungsi kita adalah kebijakan keamanan. Kebijakan keamanan mendefinisikan serangkaian kontrol yang direkomendasikan untuk sumber daya dalam langganan yang ditentukan. Di Pusat Keamanan, Anda menentukan kebijakan untuk langganan Azure sesuai dengan kebutuhan keamanan perusahaan/pribadi Anda dan jenis aplikasi atau sensitivitas data di setiap langganan.

Misalnya, sumber daya yang digunakan untuk pengembangan atau pengujian mungkin memiliki persyaratan keamanan yang berbeda dari sumber daya yang digunakan untuk aplikasi produksi. Saat ini saya sedang mengembangkan aplikasi untuk klien saya di AWS dan kami juga menggunakan kebijakan dan akun yang berbeda untuk menyiapkan sumber daya yang berbeda untuk lingkungan pengembangan yang berbeda.

Demikian pula, aplikasi yang menggunakan data yang diatur seperti informasi identitas pribadi mungkin memerlukan tingkat keamanan yang lebih tinggi. Kebijakan keamanan yang diaktifkan di Pusat Keamanan Azure mendorong rekomendasi dan pemantauan keamanan untuk membantu Anda mengidentifikasi potensi kerentanan dan mengurangi ancaman. Jika Anda mengembangkan untuk perusahaan berdasarkan kebutuhan keamanannya, saya sarankan untuk membaca Panduan Perencanaan dan Operasi Pusat Keamanan Azure: https://docs.microsoft.com/en-us/azure/security-center/security-center-planning-and-operations-guide. Kita dapat mengonfigurasi kebijakan keamanan untuk setiap langganan. Mulailah dengan mengklik petak Kebijakan di dasbor Pusat Keamanan (Gambar 4-11).



Gambar 4.11. Kebijakan untuk setiap langganan tersedia di dasbor Pusat Keamanan

Di bilah ini, kita dapat mengedit kebijakan keamanan dengan mengeklik langganan yang ingin kita edit. Opsi yang tersedia untuk setiap langganan meliputi:

- Kebijakan pencegahan: Gunakan opsi ini untuk mengonfigurasi kebijakan per langganan.
- Pemberitahuan email: Gunakan opsi ini untuk mengonfigurasi pemberitahuan email yang dikirim pada kemunculan peringatan harian pertama dan untuk peringatan dengan tingkat keparahan tinggi. Preferensi email hanya dapat dikonfigurasi untuk kebijakan langganan.
- Tingkat harga: Gunakan opsi ini untuk meningkatkan pilihan tingkat harga.
- Kebijakan Keamanan: Di bilah ini, klik Kebijakan Pencegahan untuk melihat opsi yang tersedia. Klik Aktif untuk mengaktifkan rekomendasi keamanan yang relevan untuk langganan ini.

Karena kita menggunakan fungsi untuk aplikasi tanpa server, kita tidak perlu banyak mengedit di bagian Kebijakan Keamanan. Namun, opsi di sini sebaiknya diketahui untuk selanjutnya, jadi saat Anda mulai menggabungkan berbagai layanan di aplikasi Anda, Anda akan tahu di mana harus menetapkan kebijakan keamanan.

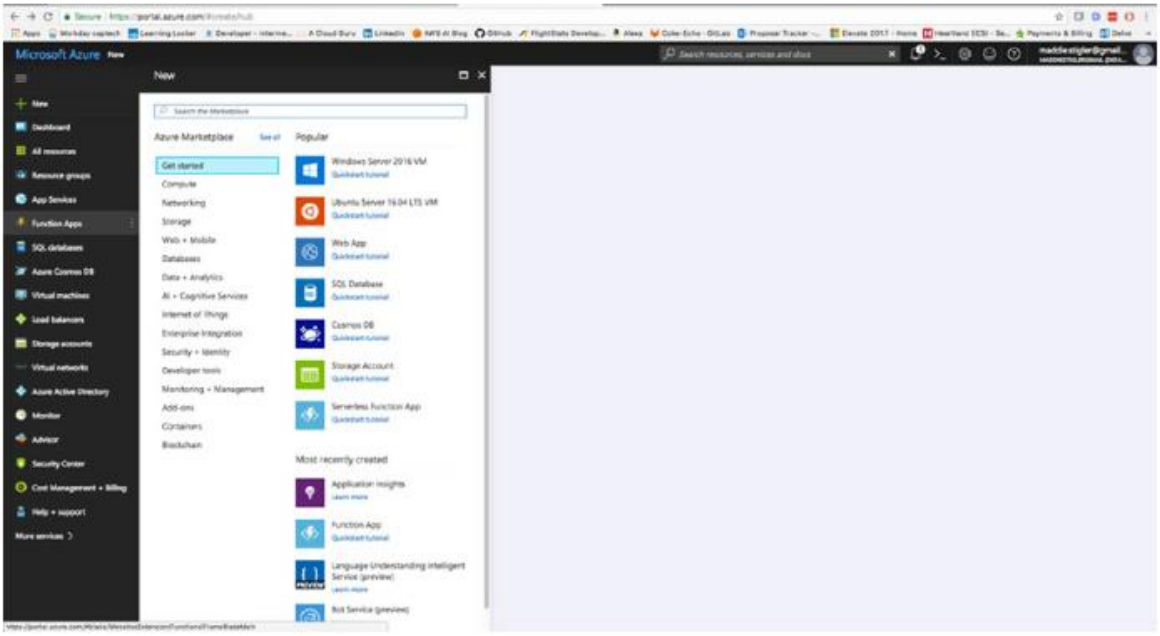
4.4 KODE PERTAMA ANDA

Di bagian ini, kita akan membahas dasar-dasar untuk menyiapkan fungsi Hello World, mirip dengan yang kita buat di AWS. Penting untuk mengulang langkah-langkah ini dengan fungsi kecil terlebih dahulu karena meskipun Anda mendapatkan banyak hal yang sama, setiap penyedia sangat berbeda dan pengaturannya juga bisa berbeda.

Halo World

Kita akan mulai dengan membuat fungsi Halo Dunia melalui portal Azure. Untuk memulai, alih-alih langsung membuka bilah Aplikasi Fungsi, klik tombol Baru di sudut kiri atas

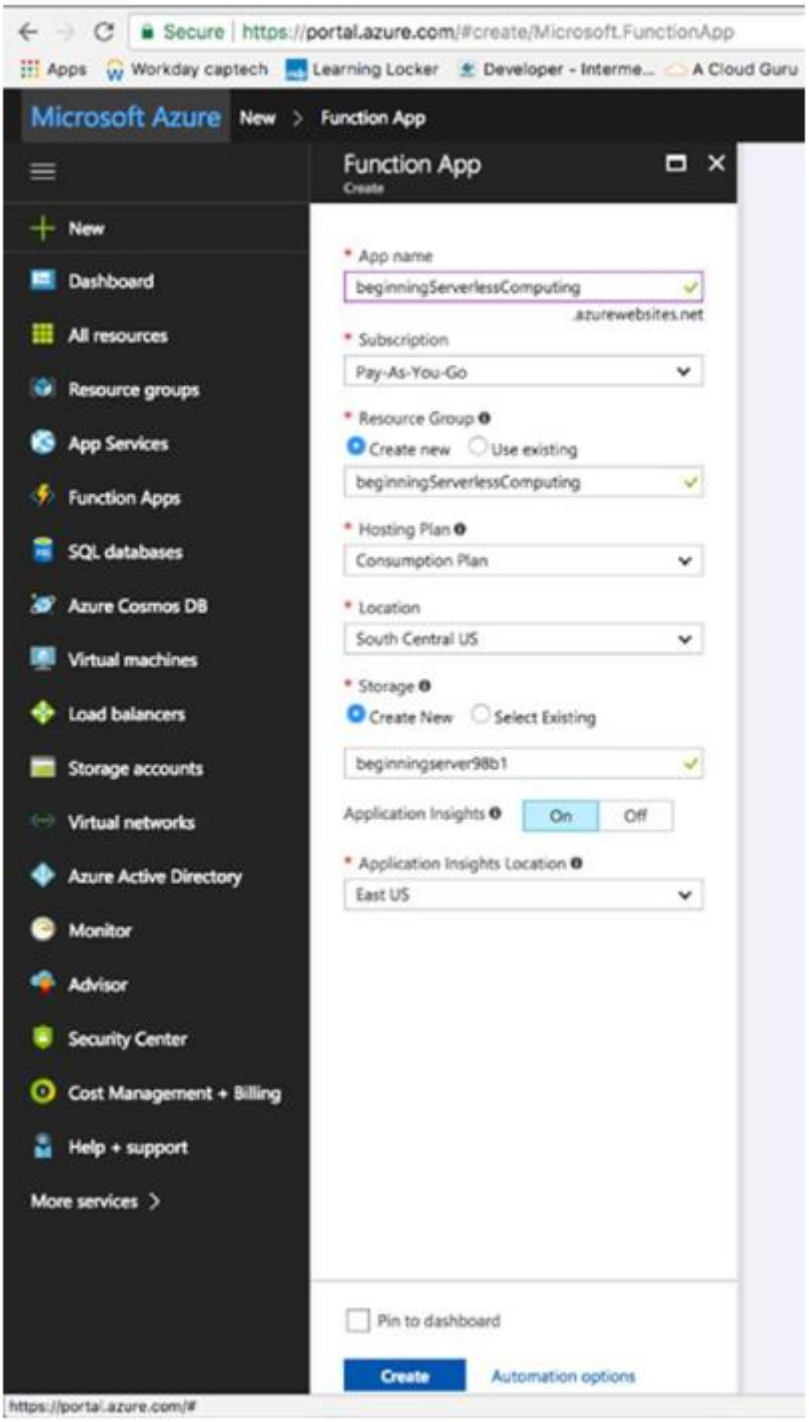
portal Azure dan buka Compute ➤ Aplikasi Fungsi (Gambar 4.12) lalu klik langganan Anda. Langganan saya adalah "bayar sesuai pemakaian", yang saya rekomendasikan untuk memulai.



Gambar 4.12. Kita akan membuat Fungsi Azure baru melalui opsi Aplikasi Fungsi di bilah Sumber Daya Baru

Anda perlu mengisi beberapa hal untuk membuat fungsi hello world, termasuk Nama Aplikasi, Langganan, Grup Sumber Daya, Paket Hosting, Lokasi, Penyimpanan, dan ya atau tidak untuk Wawasan Aplikasi. Saya telah mencantumkan beberapa petunjuk bermanfaat yang diberikan Microsoft kepada Anda terkait pengisian berbagai bidang ini:

- ✓ Nama Aplikasi: Unik secara global.
- ✓ Grup Sumber Daya: Nama grup sumber daya untuk membuat aplikasi Anda.
- ✓ Paket Hosting: Menentukan cara sumber daya dialokasikan ke aplikasi Anda. Secara default, sumber daya ditambahkan secara dinamis sebagaimana diperlukan oleh fungsi Anda. Anda hanya membayar untuk waktu fungsi Anda berjalan.
- ✓ Lokasi: Pilih satu yang dekat dengan tempat layanan Anda akan berjalan.
- ✓ Akun Penyimpanan: Juga unik secara global, nama akun penyimpanan baru atau yang sudah ada. Gambar 4.13 menunjukkan pengaturan akhir untuk aplikasi fungsi tanpa server saya.

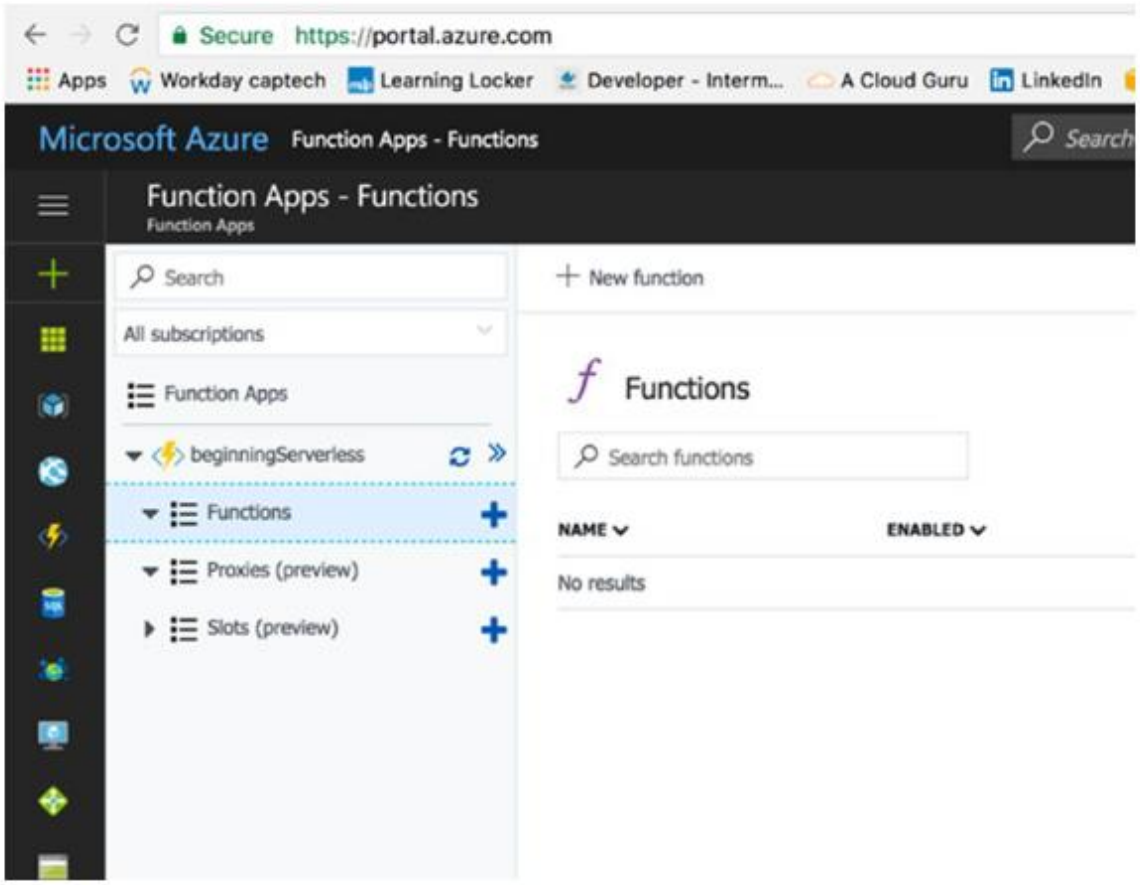


Gambar 4.13. Saya meletakkan fungsi Hello World di wilayah saya dan membuat grup sumber daya dengan nama yang sama

■ Catatan Penting untuk diingat bahwa nama aplikasi dan akun penyimpanan Anda harus unik secara global. Anda tidak akan dapat menamainya sama dengan nama saya, jadi pilih sesuatu yang masih mendefinisikan apa yang sedang Anda buat.

Saya memilih untuk menyematkan fungsi ke dasbor saya agar mudah diakses untuk selanjutnya. Setelah Anda siap, klik Buat untuk membuat dan menginisialisasi fungsi Hello

World Anda. Saat Anda mengklik Buat, Azure akan membawa Anda kembali ke portal saat fungsi sedang dibuat (Gambar 4.14). Jika Anda memilih untuk menyimpannya ke dasbor, Anda akan melihat kemajuan pembuatan Aplikasi langsung di dasbor Anda. Jika tidak, Anda masih dapat melihat kemajuan dengan mengklik bel di sudut kanan atas portal.



Gambar 4.14 Perhatikan bahwa proyek telah dibuat, tetapi tidak ada fungsi yang tercantum di bawahnya

Di Azure, kami membuat grup sumber daya yang lebih besar sebelum membuat sumber daya yang sebenarnya. Di AWS, kami membuat fungsi secara langsung. AWS memerlukan lebih sedikit langkah, tetapi Azure memberi Anda sedikit lebih banyak organisasi dan memungkinkan Anda untuk melihatnya sebagaimana Anda biasa melakukannya sebagai pengembang. Metode ini juga memungkinkan Anda untuk menyimpan semua fungsi yang terkait dengan proyek yang sama secara bersamaan. Di AWS, mungkin agak merepotkan untuk mencoba menyimpan semua fungsi proyek secara bersamaan. Itu benar-benar tergantung pada apa yang Anda inginkan sebagai arsitek.

Saat kita mengklik ikon plus di samping fungsi, kita dibawa ke halaman awal, yang memberi Anda tiga templat siap pakai untuk membuat fungsi. Opsi-opsi ini adalah:

- WebHooks dan API
- Timer

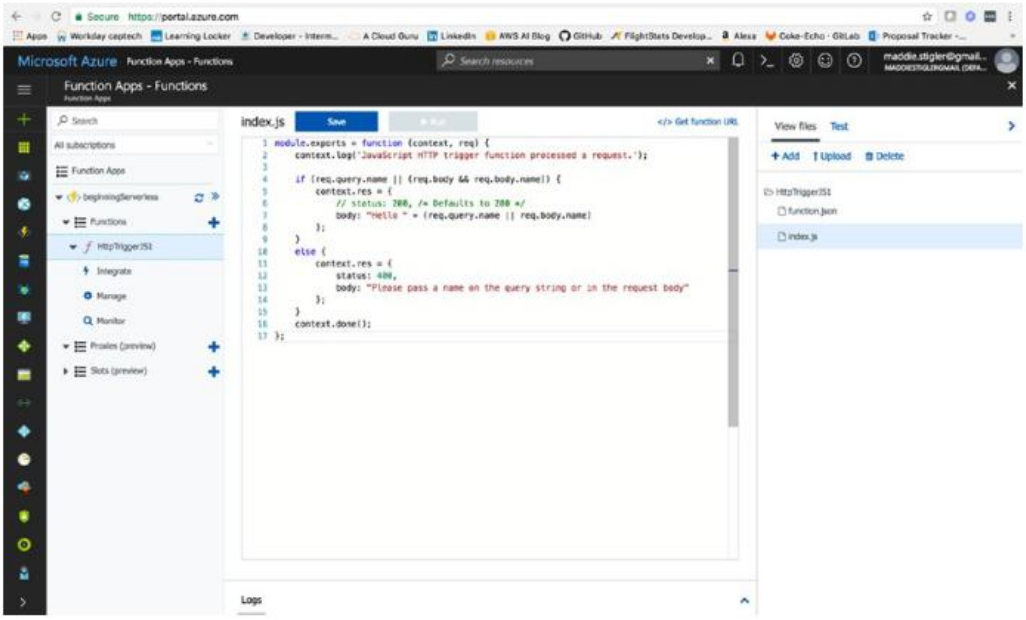
- Pemrosesan Data

Anda juga diberikan tiga opsi runtime:

- FSharp
- CSharp
- JavaScript

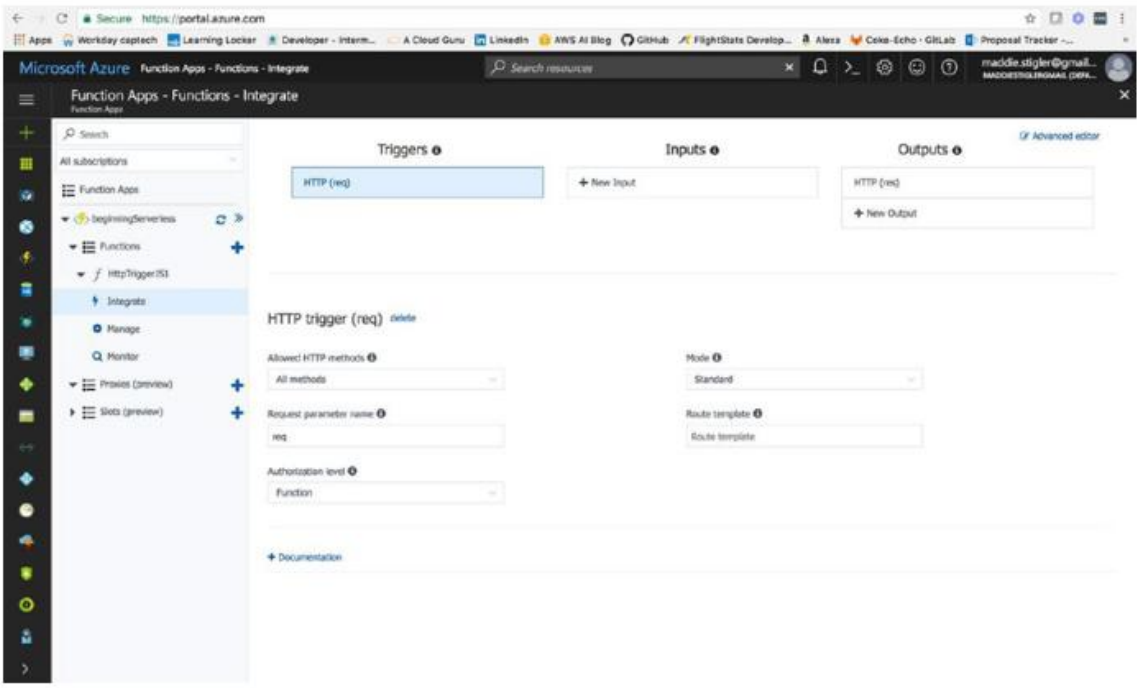
Kita akan memilih templat WebHooks dan API sehingga kita dapat membangunnya nanti untuk fungsi kita berikutnya. Azure sebenarnya memberi Anda lebih banyak opsi templating daripada ini jika Anda mengklik Buat Fungsi Kustom.

Bagi saya, nama opsi ini agak menyesatkan. Semua templat ini sudah memiliki pemicu yang dikonfigurasi untuk setiap runtime dan setiap opsi pemicu. Di AWS, Anda diberi lebih banyak opsi templat untuk memulai, serta runtime lainnya. Ini memberi Anda sedikit lebih banyak fleksibilitas sebagai pengembang. Azure juga memungkinkan Anda membuat fungsi kustom Anda sendiri baik di PowerShell maupun dari konsol. Kita akan memilih runtime JavaScript dan membuat fungsi kita. Gambar 4.15 menunjukkan seperti apa fungsi baru Anda dalam proyek Hello World kita.



Gambar 4.15. Fungsi yang baru saja dibuat menggunakan templat Webhook + API

Seperti yang Anda lihat, kami benar-benar mendapatkan banyak hal yang tidak terduga dengan templat ini. Sekarang kami memiliki fungsi di bawah opsi Functions dalam proyek kami. Jika kami mengklik fungsi itu sendiri, kami dapat melihat dan mengedit file index.js untuk melakukan apa yang kami inginkan. Saat ini, fungsi tersebut mengambil permintaan POST dengan objek name dan mengembalikan string yang mengatakan “Hello {Name}”. Dengan mengklik Integrate dalam opsi fungsi, kami dapat melihat konfigurasi pemicu untuk permintaan HTTP (Gambar 4.16).

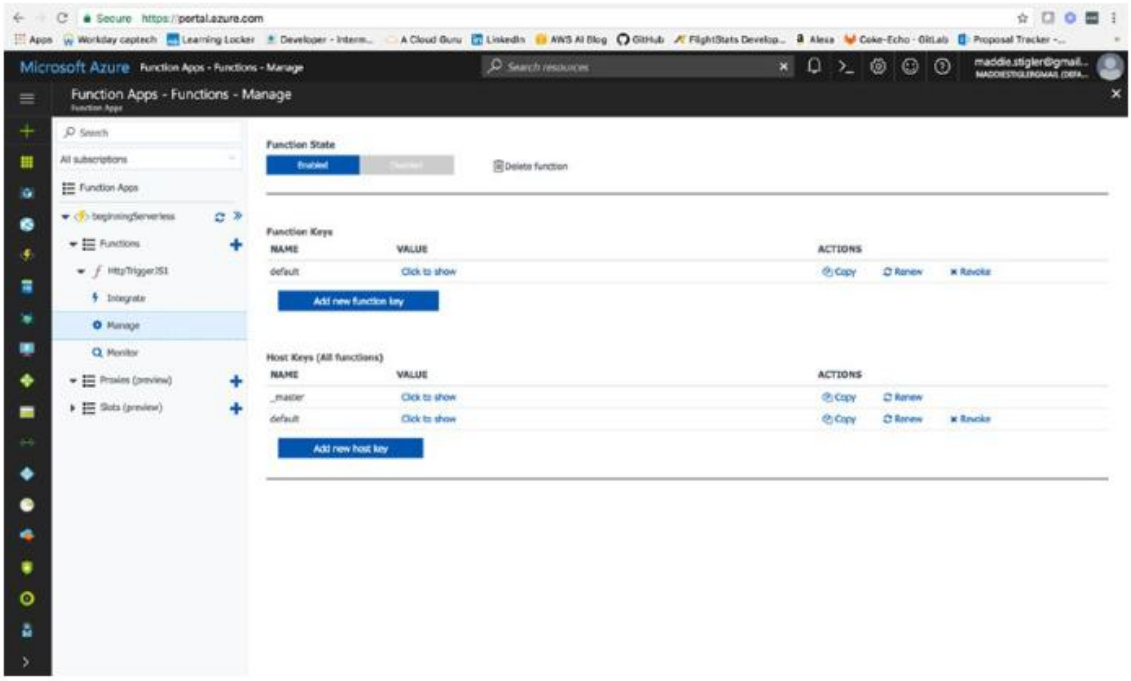


Gambar 4.16. Blade Integrate memungkinkan Anda melihat pemicu (fitur wajib fungsi Azure), mengonfigurasi input dan output, dan fitur HTTP tingkat lanjut

Di bagian Pemicu HTTP, kita diberikan opsi berikut:

- Metode HTTP yang Diizinkan: Dapat Dikonfigurasi.
- Nama Parameter Permintaan: Nama yang digunakan untuk mengidentifikasi pemicu ini dalam kode.
- Tingkat Otorisasi: Mengontrol apakah fungsi memerlukan kunci API dan kunci mana yang akan digunakan:
- Mode: Standar atau WebHook.
- Templat Rute: Memungkinkan Anda mengubah URI yang memicu fungsi.

Bagi Anda yang lebih suka mengonfigurasi ini secara manual, Anda dapat menggunakan Editor Lanjutan di sudut kanan atas untuk mengedit file function.json. File ini menentukan pengikatan dan pengaturan pemicu fungsi. Untuk memulai, saya pikir lebih mudah mengonfigurasi menggunakan UI, tetapi saat saya menjadi lebih nyaman, saya senang membuat perubahan ini di Editor Lanjutan karena menjadi lebih cepat. Di bagian Kelola di bawah fungsi kami, Anda diberikan opsi untuk mengedit Status Fungsi, Tombol Fungsi, dan Tombol Host (Gambar 4.17).



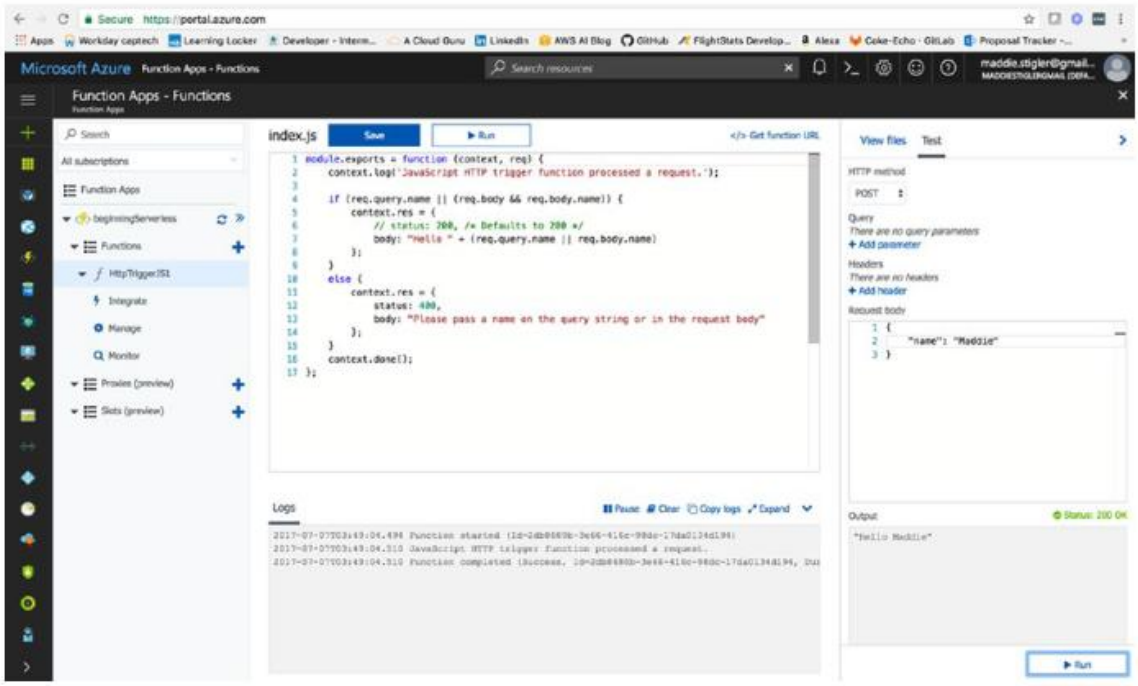
Gambar 4.17 Blade Kelola adalah tempat kunci dan status Anda dapat dikonfigurasi dan dikelola

Anda dapat menggunakan kunci ini sebagai parameter kueri dalam permintaan Anda. Jika fungsi Anda adalah WebHook (bukan fungsi HTTP biasa), saat menggunakan kunci selain yang default, Anda juga harus menentukan clientId sebagai parameter kueri (ID klien adalah nama kunci baru Anda). Kita akan membahasnya lebih rinci nanti saat kita membangun fungsi kita. Sekarang, kita akan mulai menguji fungsi kita dan melihat apa saja yang ada di dalamnya langsung dari templat.

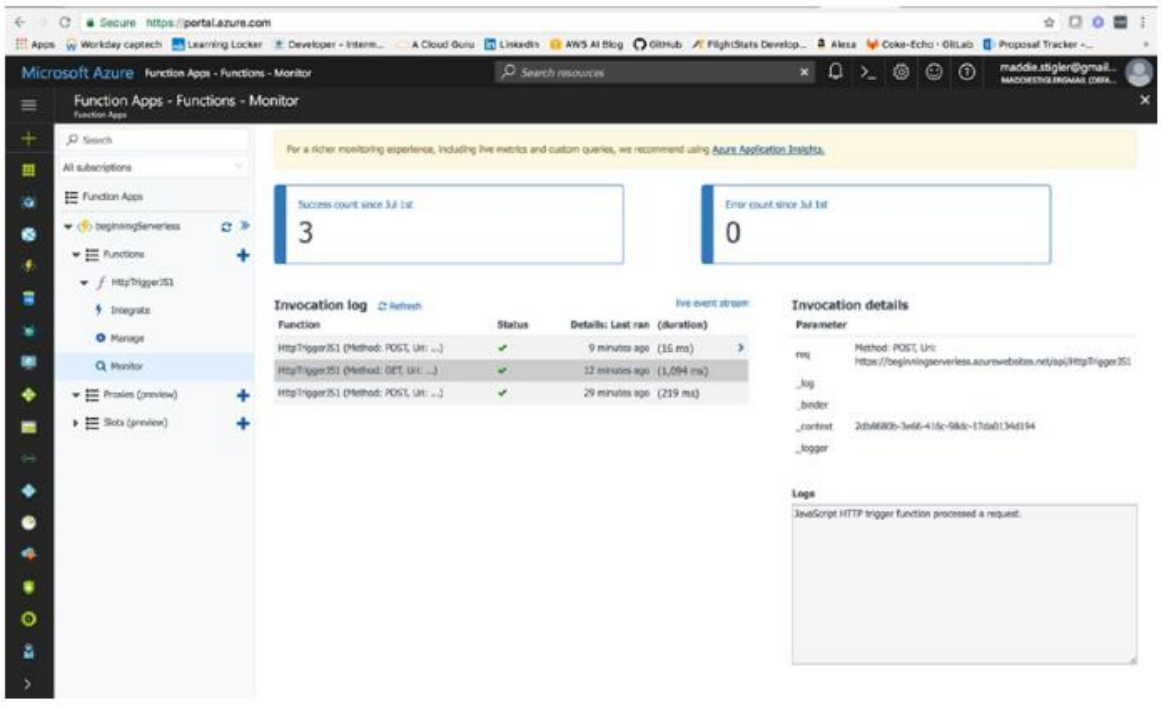
Pengujian

Untuk menguji fungsi HTTP Hello World kita, kita cukup mengklik kembali nama fungsi, dan membuka blade Uji di sisi kanan portal. Seperti yang Anda lihat, opsi pengujian disediakan untuk kita konfigurasikan dan kirimkan. Opsi pengujian sudah dikonfigurasi untuk pemicu spesifik Anda, jadi kita dapat melihat opsi permintaan HTTP termasuk metode, kueri, tajuk, dan isi permintaan.

Untuk fungsi kita, yang kita perlukan untuk memulai adalah isi permintaan dengan nama. Saya akan menyampaikan nama saya sendiri sebagai permintaan (Gambar 4.18).



Gambar 4.18. Contoh acara pengujian dengan nama saya yang diteruskan ke permintaan. Azure memberi Anda output dan log segera setelah permintaan selesai.



Gambar 4.19. Gambar ini menunjukkan bilah Monitor di bawah fungsi yang sedang kita uji dan semua yang Anda dapatkan dari bilah tersebut

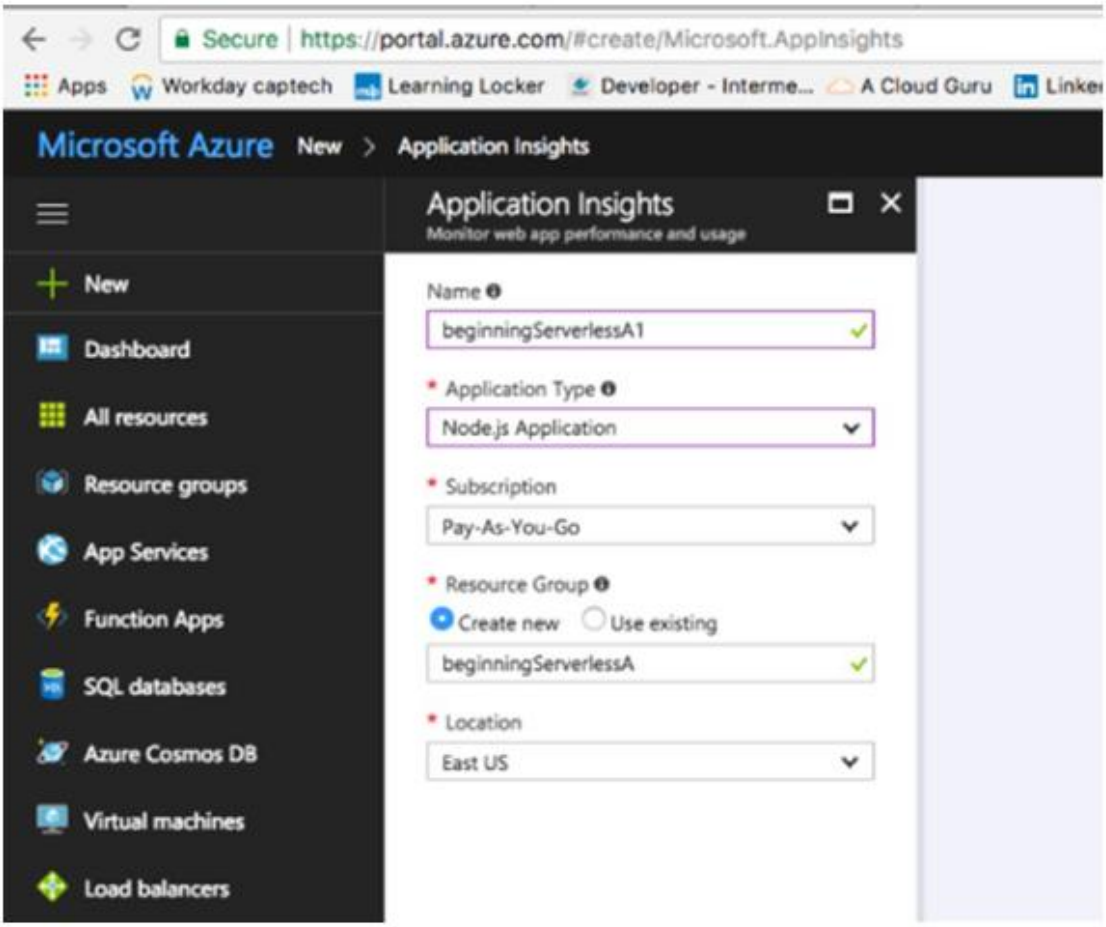
Output dan log menyerupai AWS dan cukup mudah diikuti. Azure memberi Anda beberapa opsi tambahan untuk log termasuk Jeda, Hapus, Salin, dan Perluas. Azure juga

menyediakan semua pemantauan untuk eksekusi fungsi di bilah fungsi. Untuk memantau log ini secara saksama, kita akan menavigasi ke bilah Monitor di bawah fungsi kita. Di sini kita diberikan jumlah keberhasilan, jumlah kesalahan, dan log pemanggilan dengan detail pemanggilan (Gambar 4.19).

Meskipun ini memberi kita pemantauan yang kita butuhkan untuk proyek Hello World, kita akan melanjutkan dan melihat ke dalam Application Insights, menyiapkannya, dan apa yang dapat kita pelajari darinya.

Application Insights

Application Insights (Gambar 4.20) adalah alat metrik yang hebat yang memerlukan sedikit usaha untuk menyiapkan dan mengaitkannya dengan fungsi Anda. Azure merekomendasikan pengembang memanfaatkan Application Insights dengan semua fungsi mereka. Untuk memulai, klik tombol Baru pada panel sumber daya dan pilih Alat Pengembang ➤ Application Insights.

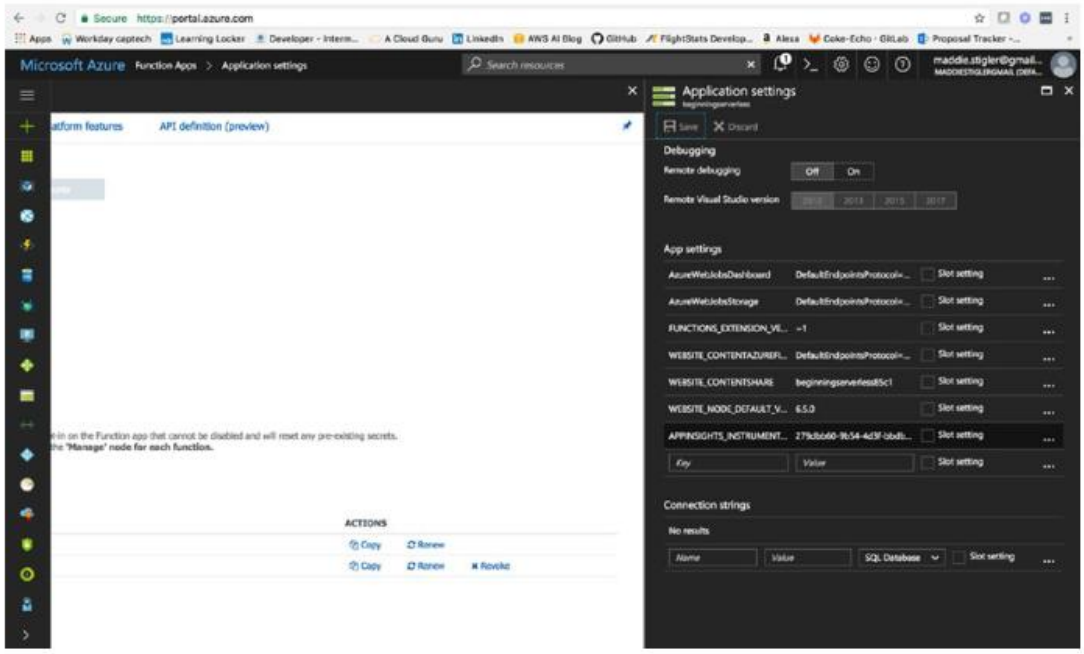


Gambar 4.20. Konfigurasi Application Insights untuk fungsi serverless awal saya. Perhatikan bahwa saya menaruhnya di grup sumber daya yang sama yang telah saya buat sebelumnya.

Setelah Anda membuat Application Insights, ambil Kunci Instrumentasi dari Essentials dan salin. Kunci ini akan digunakan untuk menautkan fungsi kita ke instans Insights. Dari sini, navigasikan kembali ke fungsi

Hello World Anda. Klik proyek Anda ➤ Setelan ➤ Kelola Setelan Aplikasi. Di bawah Setelan Aplikasi

(Gambar 4.21), cari APPINSIGHTS_INSTRUMENTATIONKEY dan tempel kunci Anda ke kotak di sebelahnya. Simpan perubahan ini. Ini akan memberi tahu App Insights apa yang harus diperhatikan dan dipantau. Segera setelah kunci dikaitkan dengan fungsi Anda, aplikasi Anda akan mulai mengirimkan informasi pemantauan App Insights tentang fungsi Anda tanpa konfigurasi lain.



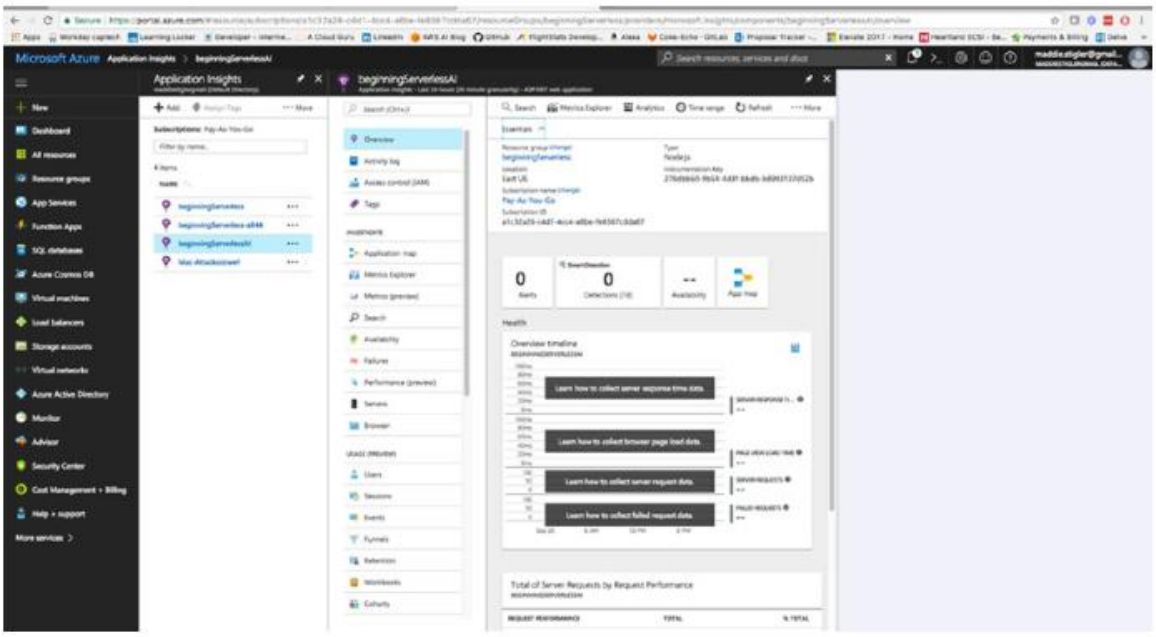
Gambar 4.21. Gambar ini menunjukkan cara menambahkan Kunci App Insights ke fungsi Anda

Application Insights adalah alat yang hebat bukan hanya karena kemudahan penggunaannya, tetapi juga karena ekstensibilitasnya. Layanan ini menjangkau banyak platform termasuk .NET, Node.js, dan J2EE. Azure saat ini adalah satu-satunya penyedia yang mendukung .NET Core untuk pemantauan aplikasi. Layanan ini juga dapat digunakan di tempat atau di cloud. Saya baru-baru ini mulai menggunakan Application Insights di situs klien saya dengan aplikasi lokal, dan penggunaannya pun mudah. Layanan ini juga terintegrasi dengan baik dengan layanan cloud lainnya dan memiliki beberapa titik koneksi ke berbagai alat pengembangan, termasuk:

- Azure Diagnostics
- Docker logs

- PowerBI
- REST API
- Continuous Export

Alat-alat ini dapat digunakan untuk mengukur Application Insights Anda. Di Azure, Application Insights dapat dieksplorasi dengan lebih mudah. Azure menyediakan dasbor bawaan yang memungkinkan Anda menjelajahi semua wawasan untuk fungsi Anda dan mengekspor wawasan tersebut. Untuk merasakan sepenuhnya Wawasan Aplikasi, navigasikan ke sumber daya baru yang kita buat di dasbor (Gambar 4.22).



Gambar 4.22. Application Insights Blade untuk Aplikasi Hello World kita

Dengan hanya mengeklik blade, Anda akan diberikan banyak informasi di awal, termasuk peringatan, ketersediaan, peta aplikasi, kesehatan, dan total permintaan. Informasi ini memberi Anda gambaran umum yang baik tentang fungsi Anda, tetapi masih banyak kekuatan Application Insights yang belum terlihat. Dengan membuka tab Analytics, Anda akan diarahkan ke halaman Application Insights yang menyediakan semua yang Anda butuhkan untuk terus memantau aplikasi Anda. Microsoft juga menyediakan fitur-fitur tambahan berikut:

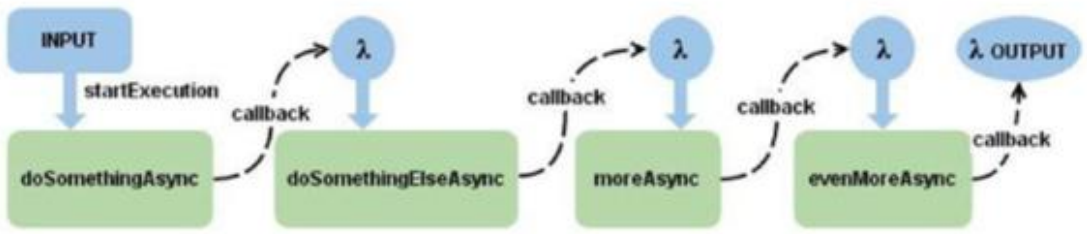
- ❖ Tingkat permintaan, waktu respons, dan tingkat kegagalan: Cari tahu halaman mana yang paling populer, pada jam berapa, dan di mana pengguna Anda berada. Lihat halaman mana yang berkinerja terbaik. Jika waktu respons dan tingkat kegagalan Anda tinggi saat ada lebih banyak permintaan, maka mungkin Anda memiliki masalah sumber daya.
- ❖ Tingkat ketergantungan, waktu respons, dan tingkat kegagalan: Cari tahu apakah layanan eksternal memperlambat Anda.

- Pengecualian: Analisis statistik agregat, atau pilih contoh tertentu dan telusuri jejak tumpukan dan permintaan terkait. Pengecualian server dan browser dilaporkan.
- Tampilan halaman dan kinerja pemuatan: Dilaporkan oleh browser pengguna Anda.
- Panggilan AJAX dari halaman web: Nilai, waktu respons, dan rasio kegagalan.
- Penghitung kinerja dari mesin server Windows atau Linux Anda, seperti penggunaan CPU, memori, dan jaringan.
- Diagnostik host dari Docker atau Azure.
- Log jejak diagnostik dari aplikasi Anda — sehingga Anda dapat menghubungkan peristiwa jejak dengan permintaan.
- Peristiwa dan metrik kustom yang Anda tulis sendiri dalam kode klien atau server, untuk melacak peristiwa bisnis seperti barang yang terjual atau permainan yang dimenangkan.

4.5 PERISTIWA HTTP

Kita akan melihat peristiwa HTTP dalam dua bagian: WebHook sebagai pemicu dan API sebagai pemicu. Untuk fungsi Azure pertama yang kita buat, kita akan membuat aplikasi WebHook sederhana. Pada bagian kedua, kita akan membangun fungsi Hello World. Saya ingin menjelajahi WebHook karena ini adalah fitur yang disediakan Azure bagi pengguna dan sedikit berbeda dari sumber daya AWS. Sebelum memulai, penting untuk memahami konsep WebHook. Saya cenderung suka terjun ke proyek dan langsung terjun ke lapangan, jadi saya tidak menghabiskan banyak waktu untuk memahami WebHook sebagaimana mestinya. Setelah beberapa kali gagal, akhirnya saya meluangkan waktu untuk benar-benar meneliti WebHook dan cara kerjanya.

WebHook hanyalah metode untuk mengubah perilaku aplikasi web atau halaman web dengan panggilan balik khusus. Panggilan balik adalah konsep penting di sebagian besar aplikasi, terutama di aplikasi Node.js. Sederhananya, panggilan balik adalah fungsi yang diteruskan ke fungsi yang menandakan akhir dari tugas tertentu. Di Node. js, panggilan balik sangat diperlukan karena sifat Node yang asinkron. Ini berarti fungsi dapat dijalankan secara paralel. Dalam kasus di mana urutan penting, penting untuk menyertakan panggilan balik untuk menunjukkan akhir suatu fungsi. Gambar 4-23 mengilustrasikan gagasan tentang panggilan balik di Node.



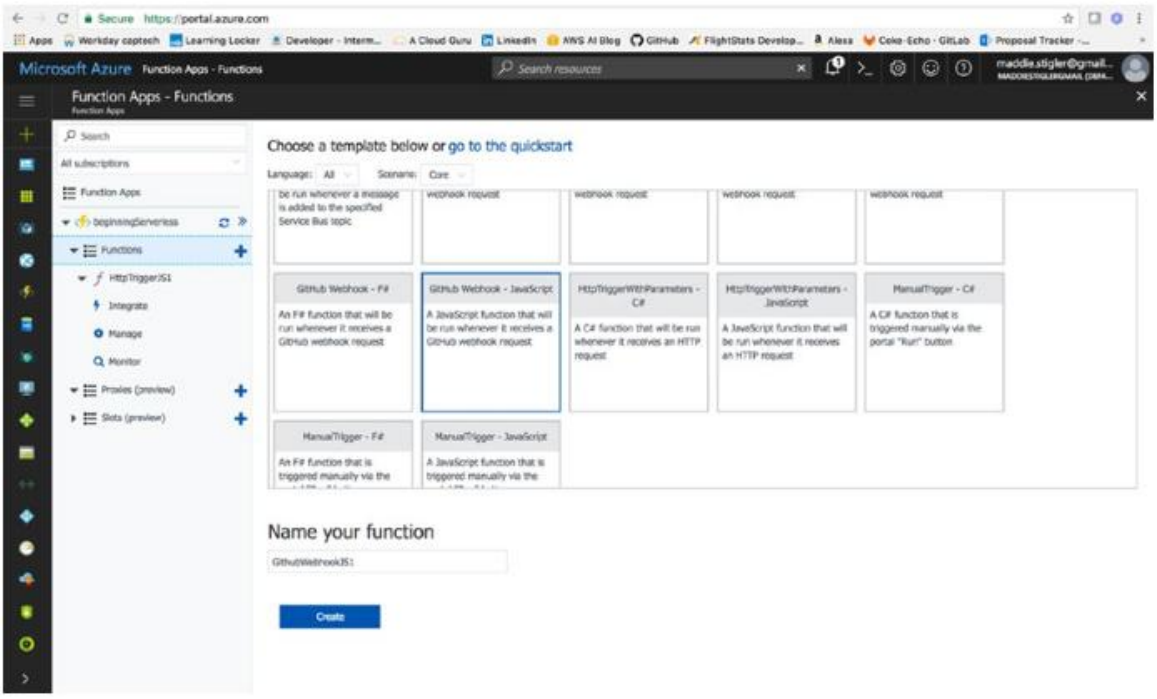
Gambar 4.23 Panggilan balik yang menandakan akhir suatu fungsi dan awal fungsi lain

Karena panggilan balik merupakan topik yang cukup penting dalam aplikasi berbasis peristiwa dan aplikasi tanpa server cenderung berbasis peristiwa, saya sarankan untuk meneliti topik ini lebih lanjut jika belum dipahami. Saya sangat menyukai artikel ini di tutorialspoint: https://www.tutorialspoint.com/nodejs/nodejs_callbacks_ concept.htm. Saya mengalami kesulitan dalam mengimplementasikan panggilan balik (saya sering kali mengabaikannya saat pertama kali mulai menulis aplikasi di Node), dan tutorial ini membantu saya memahami penggunaannya.

Panggil balik kustom yang akan kita gunakan untuk WebHook kita akan dikelola, dimodifikasi, dan dikelola oleh GitHub. Untuk tujuan latihan ini, kita akan menggunakan WebHook yang dipicu GitHub untuk memperoleh pemahaman yang lebih baik tentang cara kita dapat memanfaatkan WebHook dalam fungsi Azure kita dan cara kita dapat menggunakan sumber pihak ketiga (GitHub) untuk mengonfigurasinya.

Buat Pemicu GitHub WebHook

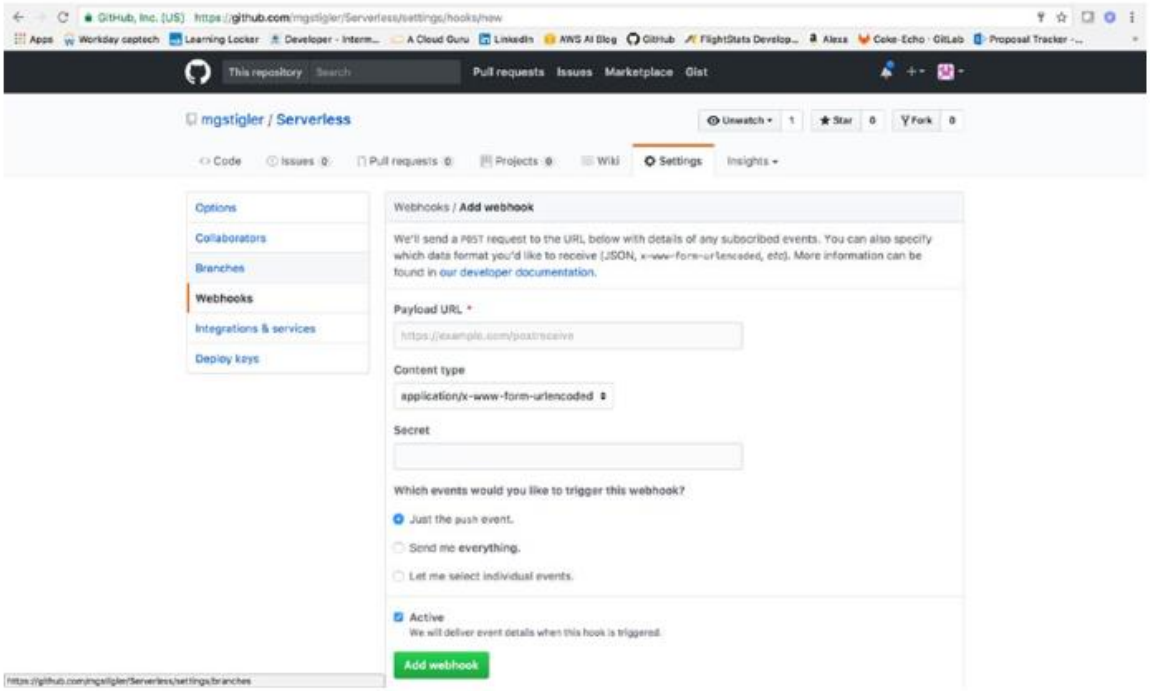
Di bilah Azure Functions, kita akan membuat fungsi baru dan memilih templat GitHub WebHook – JavaScript yang disediakan oleh pustaka Azure (Gambar 4.24). Templat awal untuk ini akan memberi kita URL fungsi dan rahasia GitHub yang akan kita gunakan untuk menautkan akun GitHub kita ke fungsi Azure kita.



Gambar 4.24. Pilih Template GitHub WebHook dari pustaka Azure

Dengan mengeklik URL fungsi dan Rahasia GitHub, kita dapat mengumpulkan informasi yang diperlukan untuk mengonfigurasi WebHook di GitHub. Fungsi ini dipicu oleh suatu tindakan, yang akan diputuskan, yang berasal dari akun GitHub kita. Saya menggunakan akun GitHub pribadi saya untuk memicu fungsi ini (Gambar 4.25). Jika Anda belum memiliki akun, silakan mendaftar untuk mendapatkan akun gratis di http://www.github.com. Jika Anda

belum memiliki repositori, silakan buat repositori saya di https://github.com/mgstigler/Serverless/tree/master/Azure/azure-service.



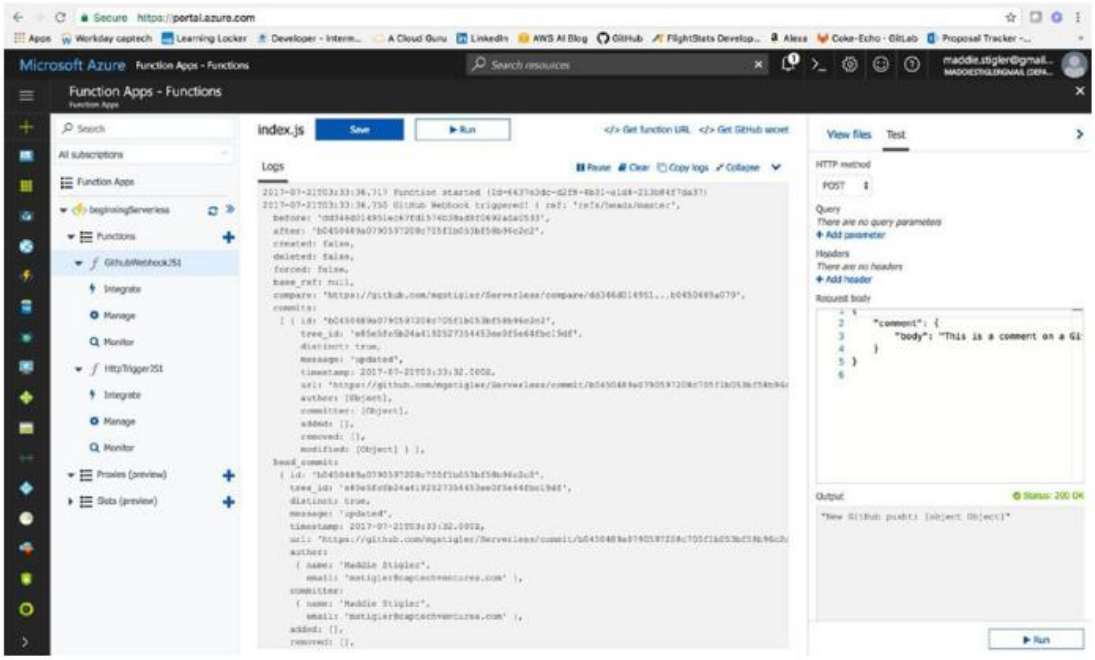
Gambar 4.25 Tempel URL dan Rahasia di pengaturan Github WebHooks Anda

Jika Anda membuka pengaturan repo Anda, Anda akan melihat tab WebHooks tempat Anda dapat menempelkan URL Payload dan Rahasia Anda. Biarkan tipe konten sebagai application/x-www-form-urlencoded. Anda dapat menentukan peristiwa mana yang ingin Anda picu WebHook, dari daftar berikut:

- Komentar komit
- Buat
- Penerapan
- Garpu
- Gollum
- Komentar masalah
- Masalah
- Label
- Anggota
- Tonggak Sejarah
- Pembuatan halaman
- Proyek
- Kartu proyek
- Kolom proyek
- Publik
- Tarik permintaan

- Dorong
- Rilis
- Repositori
- Status
- Tambahkan tim
- Pantau

Saya memilih peristiwa push untuk WebHook saya, tetapi jangan ragu untuk bereksperimen dengan apa pun yang Anda inginkan. Setelah menyiapkan konfigurasi ini, saya cukup membuat perubahan pada repo saya dan mengirimkannya ke cabang utama saya. Saya dapat melihat efek pemicu WebHook dengan melihat log untuk fungsi saya di portal Azure (Gambar 4.26).



Gambar 4.26. Log menunjukkan pemicu dan eksekusi Webhook

Isi fungsi saya hanya menuliskan permintaan ke log. Anda dapat melihat semua hal tentang pengiriman ke repo saya, termasuk komiter, stempel waktu, dan apa yang saya ubah. Ini mungkin tidak begitu membantu saat ini, tetapi Anda dapat melihat bagaimana WebHook dapat digunakan untuk menyiapkan aplikasi GitHub yang berlangganan berbagai acara di GitHub. Aplikasi ini dapat digunakan untuk memperbarui pelacak masalah, memicu pembuatan integrasi berkelanjutan, dan bahkan menyebarkan ke lingkungan yang berbeda. Contoh ini merangkum contoh GitHub WebHook kami. Jika Anda tertarik untuk mempelajari lebih lanjut, saya rasa ini merupakan sumber yang bagus: https://developer.github.com/webhooks/.

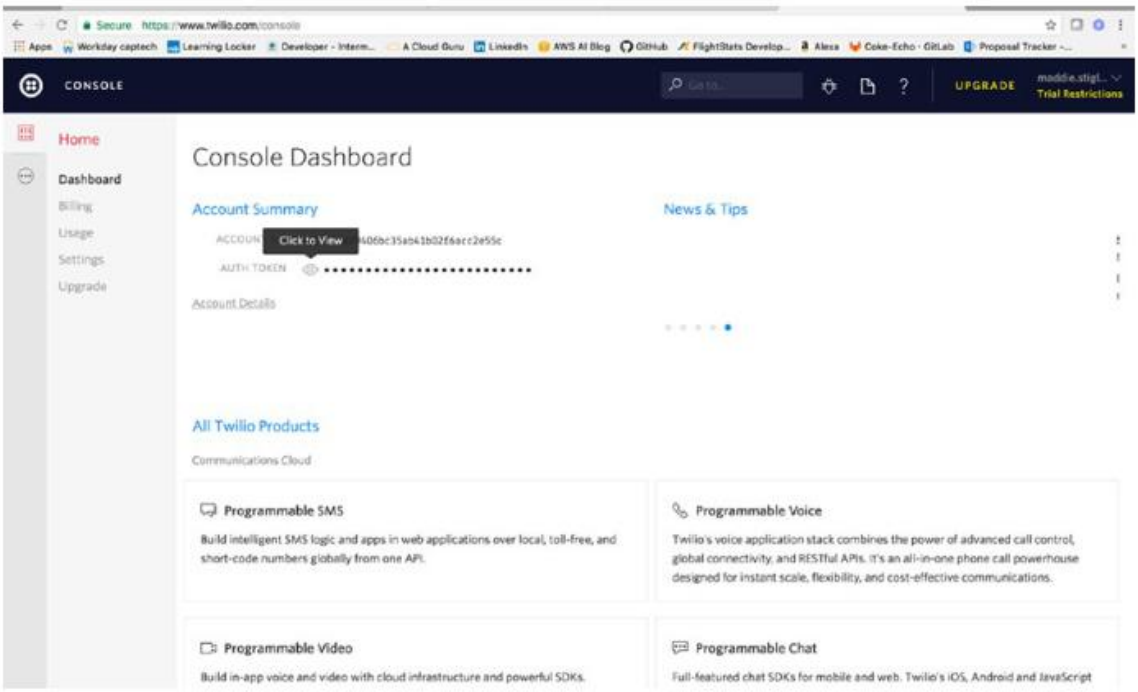
Membangun Berdasarkan Pemicu API Hello World Kita

Untuk membangun pemicu HTTP kita, kita akan membuat layanan notifikasi yang memanfaatkan Twilio dan pemicu API Hello World, serta pengikatan Output. Saya akan membuat layanan ini menyenangkan dan mengatakan bahwa ini akan menjadi layanan

notifikasi pengiriman makanan, tetapi Anda dapat membuatnya sesuai keinginan Anda. Untuk membuatnya, kita akan kembali ke fungsi Hello World kita dan mengubah permintaan HTTP dari "apa pun" menjadi POST. Saya suka memisahkan panggilan API saya ke dalam fungsi yang berbeda. Karena tujuan penyampaian solusi tanpa server adalah desain yang cepat dan berorientasi pada peristiwa, kita akan tetap menggunakan pendekatan ini. Untuk memulai, kita akan mengonfigurasi ulang fungsi kita, jadi navigasikan kembali ke Hello World Anda.

Lakukan hal berikut untuk mengonfigurasi ulang:

- Di Integrasikan, navigasikan ke Pemicu. Ubah metode yang diizinkan ke Metode Terpilih dan pilih metode POST. Saya menetapkan templat rute saya menjadi orders/. Jangan ragu untuk melakukan hal yang sama.
- Di Output, klik Tambahkan Output baru dan pilih Twilio dari output yang tersedia. • Untuk Nama Parameter Pesan, pilih untuk menggunakan nilai pengembalian fungsi (Anda juga dapat mengatur nama parameter Anda sendiri jika Anda mau
- Saya membiarkan Pengaturan Twilio sebagai TwilioAuthToken dan TwilioAccountSid.
- Saya juga membiarkan parameter lainnya kosong sehingga saya dapat mengonfigurasinya dalam fungsi saya.

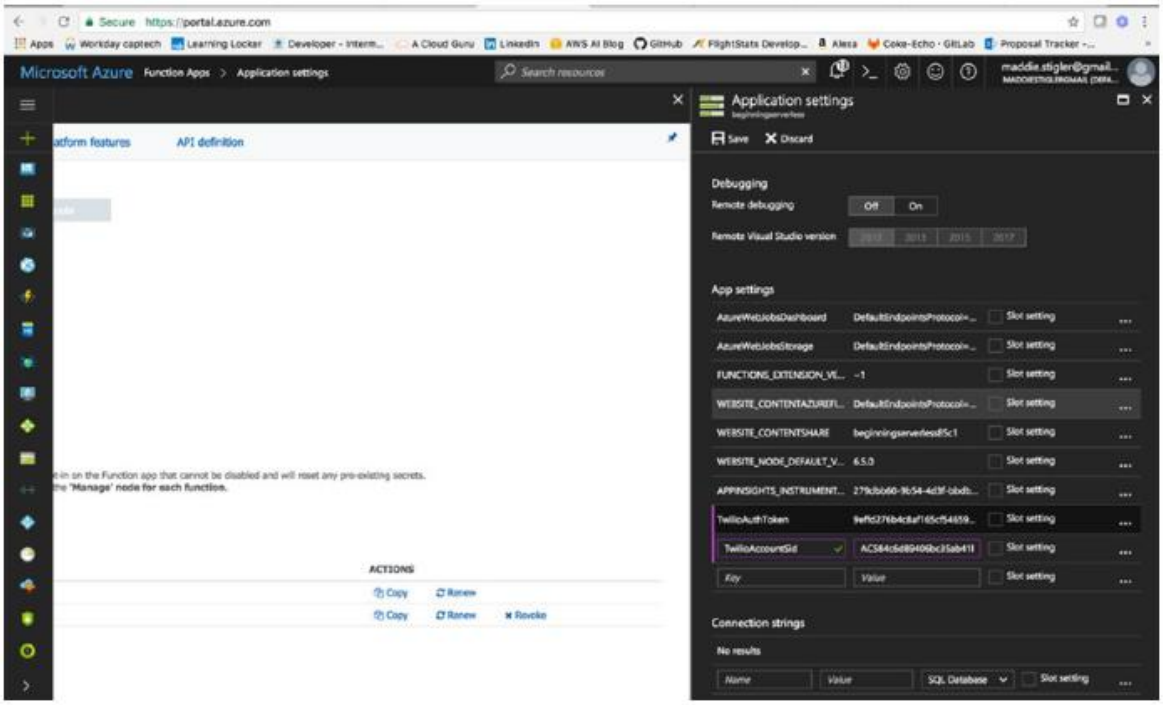


Gambar 4.27. Salin dua token dari Konsol Twilio sehingga kita dapat menggunakannya dalam fungsi kita

Kami telah menyiapkannya di portal Azure, tetapi sekarang kami memerlukan akun Twilio dan informasi yang terkait dengan akun tersebut. Untuk mendapatkan informasi ini, buka https://www.twilio.com/try-twilio dan daftar secara gratis. Setelah Anda melakukannya, Anda akan mencapai halaman awal Twilio (Gambar 4.27), yang memberi Anda Token Otorisasi dan Sid Akun Anda.

Saya akan berhenti sejenak di sini untuk plug-in Twilio. Twilio adalah platform pengembang yang unik untuk komunikasi. Pengembang menggunakan API Twilio untuk menambahkan kemampuan suara, pengiriman pesan, dan video ke aplikasi mereka. Dengan melakukan ini, mereka dapat memberikan pengalaman komunikasi yang tepat kepada pelanggan mereka dengan mudah dan murah. Twilio beroperasi dengan menyediakan lapisan perangkat lunak yang menghubungkan dan mengoptimalkan jaringan komunikasi di seluruh dunia. Dengan cara inilah pengembang dapat memungkinkan pengguna untuk menelepon dan mengirim pesan kepada siapa pun di mana pun dengan biaya yang murah atau bahkan gratis. Perusahaan yang saat ini menggunakan API Twilio termasuk Uber, Lyft, dan Netflix.

Meskipun saya adalah seseorang yang senang membangun layanan saya sendiri dalam lingkungan cloud, saya juga sangat menyarankan untuk mempertimbangkan API publik seperti Twilio jika diperlukan. Padanan AWS untuk menggunakan Twilio adalah SNS, Simple Notification Service. Kedua alat tersebut dapat membuat hidup Anda jauh lebih mudah dan memberi Anda lebih banyak keamanan dan keandalan daripada yang Anda harapkan. Setelah Anda mengambil token dari Twilio, simpan token tersebut di Setelan Aplikasi Anda. Saya menyimpan token saya sebagai TwilioAuthToken dan TwilioAccountSid (Gambar 4-28). Anda dapat memberi nama apa pun yang Anda suka, tetapi Anda perlu mengonfigurasinya di keluaran fungsi Anda.



Gambar 4.28. Simpan Token Twilio dan Sid di pengaturan lingkungan aplikasi Anda

Agar dapat menggunakan integrasi Twilio secara efektif, kita juga perlu mendapatkan nomor telepon keluar dari Twilio. Mereka akan menyediakannya secara gratis di https://www.twilio.com/console/phone-numbers/incoming. Ini adalah nomor yang akan Anda gunakan untuk mengirim teks dari fungsi Anda. Simpan nomor ini dan ingat untuk digunakan nanti.

Jika Anda menavigasi kembali ke fungsi kita, kita akan mengonfigurasi file function.json untuk pengikatan keluaran Twilio. Pengikatan keluaran Twilio memiliki struktur khusus yang akan kita ikuti agar dapat menggabungkannya dalam fungsi kita. Function.json akan menyediakan properti berikut:

- name: Nama variabel yang digunakan dalam kode fungsi untuk pesan teks SMS Twilio.
- type: Harus disetel ke twilioSms.
- accountSid: Nilai ini harus ditetapkan ke nama Pengaturan Aplikasi yang menyimpan Sid Akun Twilio Anda.
- authToken: Nilai ini harus ditetapkan ke nama Pengaturan Aplikasi yang menyimpan token autentikasi Twilio Anda.
- to: Nilai ini ditetapkan ke nomor telepon yang menjadi tujuan pengiriman SMS. from: Nilai ini ditetapkan ke nomor telepon yang menjadi tujuan pengiriman SMS. direction: Harus ditetapkan ke out.
- body: Nilai ini dapat digunakan untuk membuat kode keras pesan teks SMS jika Anda tidak perlu menetapkannya secara dinamis dalam kode untuk fungsi Anda.

File function.json terakhir saya terlihat seperti ini:

```
{
  "bindings": [
  {
    "authLevel": "function",
    "type": "httpTrigger",
    "direction": "in",
    "name": "req",
    "route": "orders/",
    "methods": [
    "post"
    ]
  },
  {
    "type": "http",
    "direction": "out",
    "name": "res"
  },
  {
    "type": "twilioSms",
    "name": "$return",
    "accountSid": "TwilioAccountSid",
    "authToken": "TwilioAuthToken",
    "direction": "out"
  }
  ],
  "disabled": false
}
```

Pada titik ini, kita seharusnya sudah menyelesaikan semua pengaturan sehingga kita benar-benar dapat menulis fungsi tersebut. Untuk memulai, kita perlu memerlukan Twilio

sehingga kita dapat menggunakan klien Twilio untuk mengirim pesan. Dari sana, kita cukup mengambil data yang kita terima, menguraikannya, dan mengirimkannya dalam pesan. Isi berkas index.js Anda seharusnya terlihat seperti Daftar 4.1.

Daftar 4.1. Isi file index.js untuk fungsi Twilio kita

```
var client = require('twilio')(process.env.TwilioAccountSid,
  process.env.TwilioAuthToken);

module.exports = function (context, req) {
  console.log(req.body.name);
  if(req.body.name && req.body.phoneNumber){
    client.messages.create({
      from: '+18178544390',
      to: req.body.phoneNumber,
      body: "Hello "+ req.body.name + "! Your order of " + req.body.order
      + " is on the way."
  }, function(err, message) {
    if(err) {
      console.error(err.message);
    }
  });
}

  else {
    console.error("Please include a request body with a name and a phone
      number");
  }
};
```

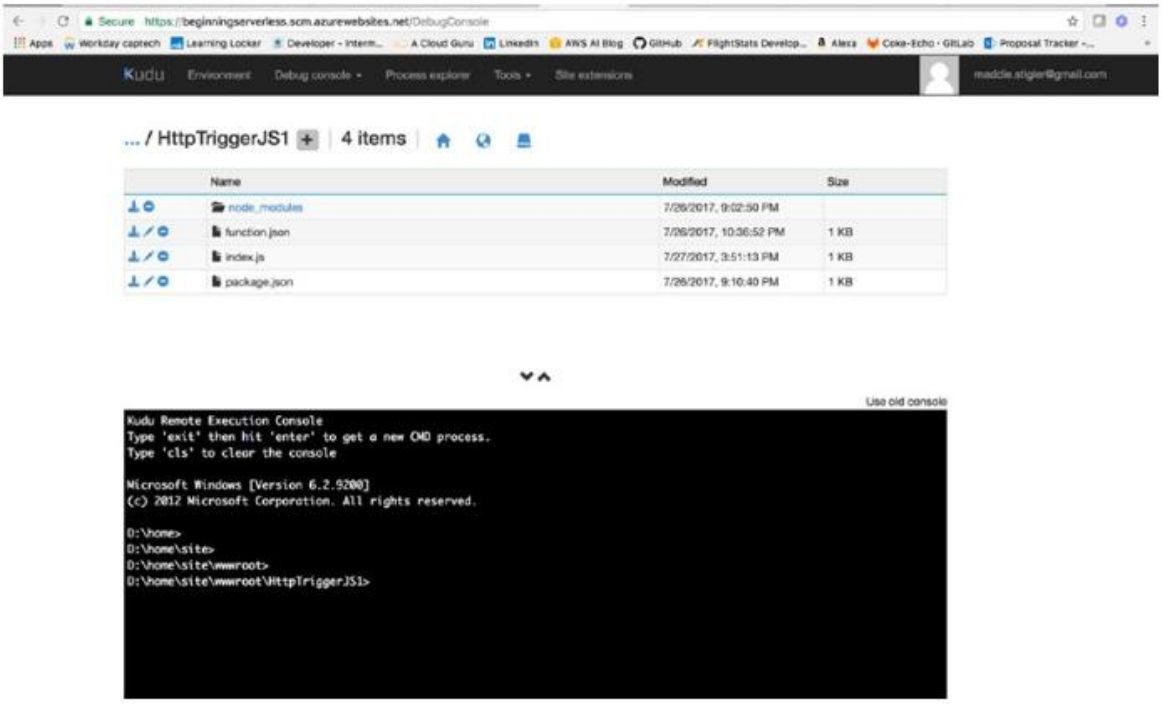
Kita juga memerlukan file package.json karena kita menggunakan modul Node. Setelah kita menyempurnakannya, bagian tersulit sebenarnya adalah mengunggahnya ke Azure. Di sinilah Azure menjadi sedikit rumit. Anda tidak dapat langsung mengunggah seluruh folder dari konsol. Ini berarti Anda harus memilih semua file satu per satu sebelum mengunggah, atau Anda menggunakan kerangka kerja seperti Serverless untuk membantu Anda mengunggah (rekomendasi saya), atau Anda menggunakan alat baris perintah daring Azure. Saya akan membahas opsi ketiga agar kita mendapatkan sedikit pemahaman tentang ini. Namun, dalam praktiknya, saya pasti akan menggunakan Serverless. Saya berharap ini adalah sesuatu yang berubah di masa mendatang. Bagian tersulit seharusnya bukan mengunggah proyek Anda, tetapi sebenarnya sedikit melelahkan.

Setelah Anda memiliki file indeks, fungsi, dan package.json di proyek Anda di konsol, navigasikan ke <functionname>.scm.azurewebsites.net. Ini adalah URL fungsi Anda dengan Kudu. Untuk memberikan sedikit latar belakang, Kudu adalah kerangka kerja penyebaran yang dapat dipicu oleh Git. Sebenarnya sangat mirip dengan Visual Studio Online. Saya menggunakan VSOnline di situs klien saya dan melihat banyak persamaan antara itu dan Kudu saat pertama kali menggunakan Kudu. Anda dapat memeriksa kode, membuat definisi build yang akan membangun dan menjalankan kode dan pengujian, lalu melanjutkan untuk mendorong konten ke situs web Azure.

Manfaat lain dari Kudu, yang akan kita lihat, adalah kemampuan untuk memiliki lebih banyak akses dan kontrol atas kode Anda secara online. Salah satu frustrasi terbesar saya saat pertama kali bekerja dengan Azure adalah tidak memiliki kontrol atas proyek saya. Saya menghabiskan banyak waktu mencari solusi untuk mengunggah modul node dan mencoba mengunggahnya secara fisik melalui Cloud Shell tetapi tidak berhasil. Saya bahkan mengirim email kepada seorang mentor dan bertanya bagaimana ia dapat menggunakan modul node dengan fungsi Azure; tanggapannya adalah bahwa ia tidak pernah mampu dan beralih ke runtime yang sama sekali berbeda hanya untuk menghindari sakit kepala.

Ketika saya akhirnya menemukan solusi online, itu adalah menggunakan Kudu untuk berinteraksi dengan kode Anda secara langsung. Kita akan melakukan ini sekarang agar kita memiliki pemahaman yang lebih baik tentang keduanya, cara menavigasi dan mengakses kode Anda di cloud dan manfaat Kudu.

Saat pertama kali masuk ke situs Anda, Anda hanya akan melihat dasbor. Kita akan masuk ke konsol Debug (Gambar 4-30) untuk dapat mengakses kode kita. Navigasi ke root aplikasi kita dengan package.json dan di konsol, dan gunakan npm install untuk menginstal paket NPM yang kita butuhkan.



Gambar 4.29. Seperti inilah tampilan dasbor Kudu saat Anda membuka konsol Debug

Setelah modul NPM diunggah, kita dapat membuka kembali dasbor dan menguji fungsi kita menggunakan badan POST yang pernah kita gunakan sebelumnya. Milik saya terlihat seperti ini:

```
{
  "name": "Maddie",
  "order": "Mac 'n Cheese",
  "phoneNumber": "1XXXXXXXXX"
```

```
}
```

Anda menggunakan nomor telepon dan perintah untuk menyusun pesan Anda. Saya mengirim teks itu kepada diri saya sendiri untuk mengujinya, tetapi jangan ragu untuk mengganggu teman dengan aplikasi baru Anda. Anda akan menerima teks dengan pesan Anda segera setelah mengujinya. Untuk memicu fungsi ini di luar lingkungan pengujian, Anda dapat menggunakan Postman atau sumber daya lain dan melakukan ping ke URL fungsi.

Untuk memicu suatu fungsi, Anda mengirim permintaan HTTP ke URL yang merupakan kombinasi dari URL aplikasi fungsi dan nama fungsi:

```
https://{function app name}.azurewebsites.net/api/{function name}
```

Jika Anda menentukan suatu metode, URL Anda tidak perlu merujuk ke fungsi tertentu. Jadi, URL fungsi kita yang sebenarnya adalah https://beginningserverless.azurewebsites.net/api/orders/. Dan voila! Anda memiliki aplikasi notifikasi yang dapat disesuaikan dan sederhana (Gambar 4.30).

Gambar 4.30. Teks dari Aplikasi Perpesanan Twilio

Pada latihan berikutnya, kita akan membangun fungsi pemicu penyimpanan yang akan terhubung dengan aplikasi ini.

Tambahkan Ke Fungsi Anda

Latihan: Bangun fungsi ini untuk menambahkan lebih banyak metode dan proksi Azure Functions, fitur pratinjau yang memungkinkan Anda meneruskan permintaan ke sumber daya

lain. Anda menetapkan titik akhir HTTP seperti halnya pemicu HTTP, tetapi alih-alih menulis kode untuk dijalankan saat titik akhir tersebut dipanggil, Anda menyediakan URL ke implementasi jarak jauh. Ini memungkinkan Anda menyusun beberapa sumber API ke dalam satu permukaan API yang mudah digunakan klien. Ini sangat berguna jika Anda ingin membangun API sebagai layanan mikro. Proksi dapat menunjuk ke sumber daya HTTP apa pun, seperti berikut:

- Azure Functions
- Aplikasi API
- Kontainer Docker
- API lain yang dihosting

Untuk mempelajari lebih lanjut, kunjungi https://docs.microsoft.com/en-us/azure/azure-functions/functions- proxies.

4.6 PERISTIWA PENYIMPANAN

Untuk peristiwa yang dipicu penyimpanan, kita akan membuat aplikasi yang dipicu oleh penambahan pada antrean. Untuk melanjutkan tema makanan, kita akan membuat antrean Pesanan yang akan menyimpan pesanan untuk pengiriman saat pesanan masuk. Saat pesanan baru ditambahkan ke antrean, kita akan memperbarui antrean dan membuat POST ke URL fungsi sebelumnya. Kita menggunakan antrean dalam aplikasi ini karena masuk akal dalam konteks pengiriman. Namun, ada beberapa pemicu penyimpanan yang disediakan oleh Azure, termasuk berikut ini:

- Azure Blob Storage
- Tabel Penyimpanan
- Tabel SQL
- Basis Data Tanpa SQL

Saya sarankan untuk melihat opsi lain ini setelah demonstrasi antrean. Meskipun opsi penyimpanan Azure mirip dengan yang ada di AWS, ada beberapa perbedaan dalam cara pengaturan dan aksesnya. Sebaiknya baca kembali Bab 3 dan coba terapkan aplikasi yang kita buat di AWS di lingkungan Azure baru. Itu akan memberi Anda tolok ukur yang baik untuk perbedaan dalam pengaturan, pengembangan, dan penerapan.

Azure Queue Storage

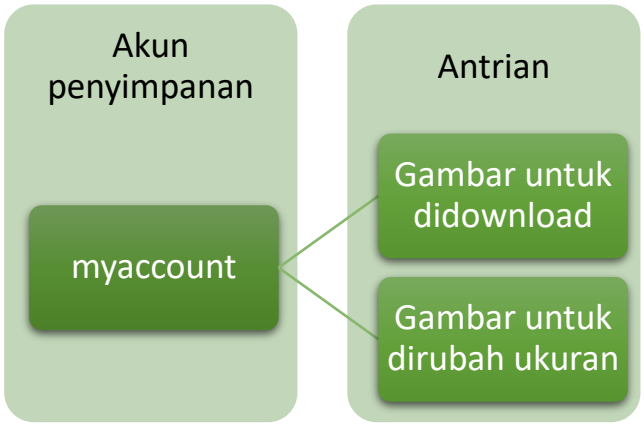
Penyimpanan Azure Queue menyediakan pengiriman pesan awan antarkomponen aplikasi. Dalam mendesain aplikasi untuk skalabilitas, komponen aplikasi sering dipisahkan, sehingga dapat diskalakan secara independen. Penyimpanan antrean memberikan pengiriman pesan asinkron untuk komunikasi antarkomponen aplikasi, baik yang berjalan di awan, di desktop, di server lokal, atau di perangkat seluler. Penyimpanan antrean juga mendukung pengelolaan tugas asinkron dan membangun alur kerja proses.

Gambar 4-31 adalah contoh yang disediakan oleh Azure untuk menjelaskan komponen antrean. Contoh yang digunakan adalah aplikasi pengubahan ukuran gambar yang bergantung pada urutan. Layanan Antrean berisi komponen-komponen berikut:

Format URL: Mirip dengan fungsi, antrean dapat dialamatkan menggunakan format URL berikut:

http://<akun penyimpanan>/queue.core.windows.net/queue

URL berikut mengalamatkan antrean pada Gambar 4-32:
http://myaccount.queue.core.windows.net/images-to-download
Akun Penyimpanan: Semua akses ke Azure Storage dilakukan melalui akun penyimpanan. Antrean: Antrean berisi sekumpulan pesan. Semua pesan harus berada dalam antrean. Perhatikan bahwa nama antrean harus semuanya huruf kecil.
Pesan: Pesan, dalam format apa pun, hingga 64 KB. Waktu maksimum pesan dapat tetap berada dalam antrean adalah 7 hari.



Gambar 4.31. Ilustrasi penyimpanan Antrean yang disediakan Microsoft Azure

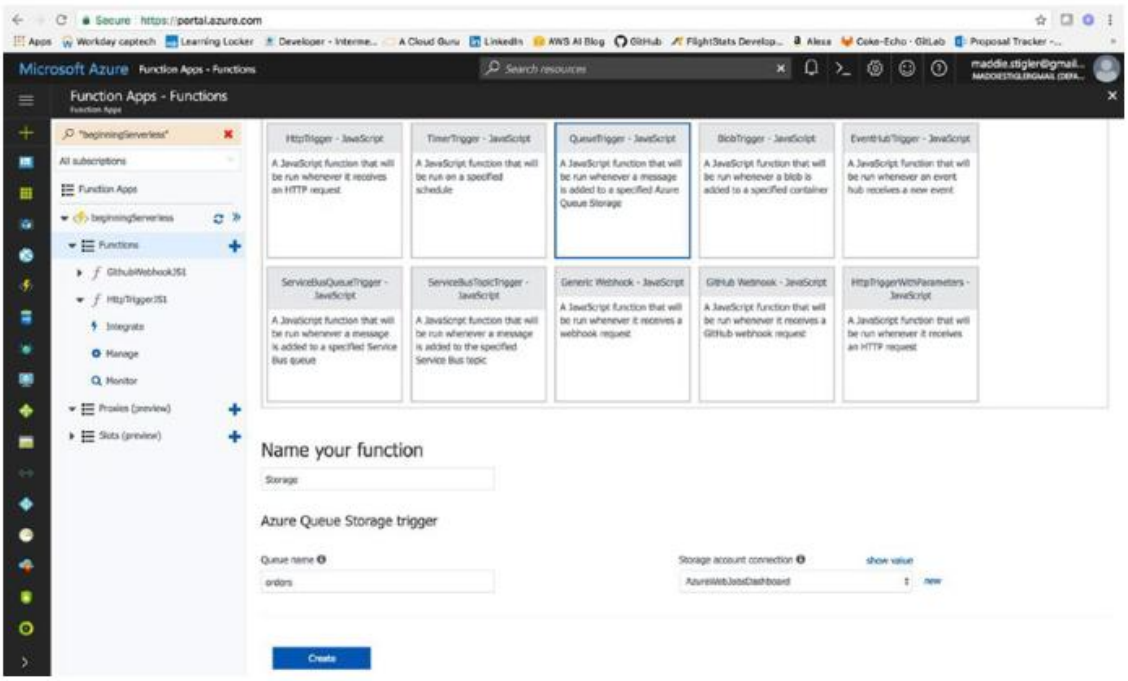
Antrean Azure cukup mudah diatur dan dipahami, jadi saya senang menggunakannya. Kelemahan antrean adalah kurangnya kontrol terhadap kolom dan baris. Baris akan meninggalkan antrean segera setelah dieksekusi (begitulah cara kerja antrean), dan Anda hanya memiliki satu kolom pesan, yaitu pesan yang sedang dikirim. Kolom yang Anda dapatkan dari Azure adalah

- Id
- Insertion Time (UTC)
- Expiration Time (UTC)
- Dequeue count
- Size (bytes)

Untuk tujuan latihan ini, kita akan dapat menyelesaikan semua yang perlu kita lakukan menggunakan antrean. Namun, jika Anda ingin menambahkan informasi tambahan, Anda mungkin ingin mencoba menggunakan solusi penyimpanan lain.

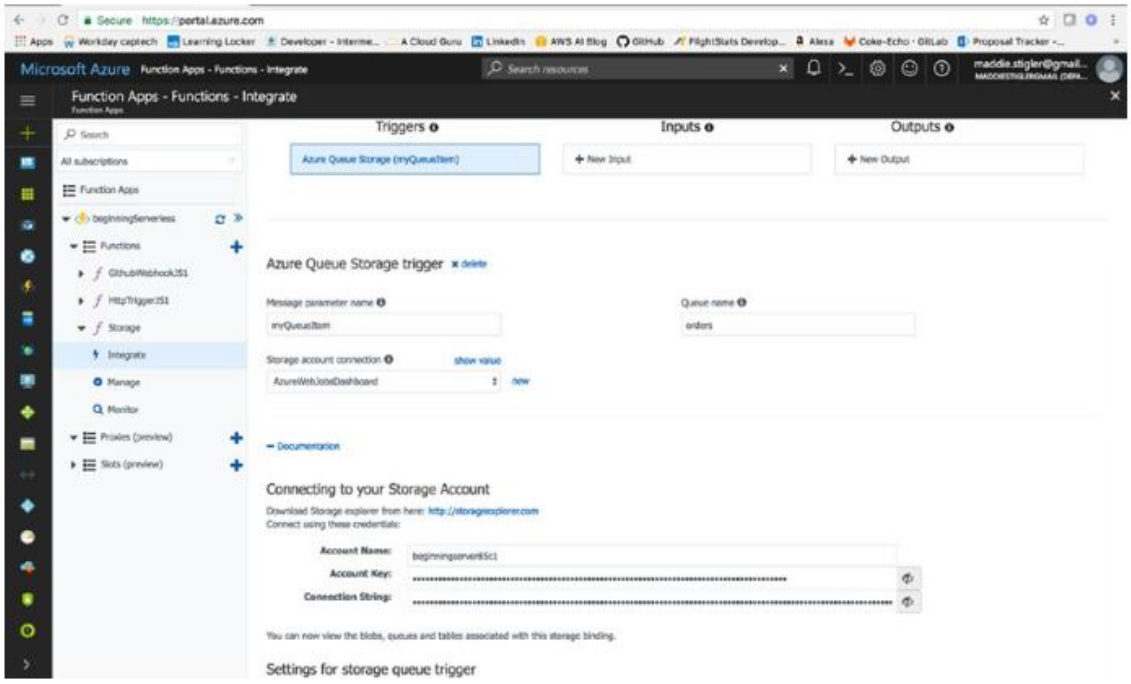
Buat Fungsi

Kita akan kembali ke portal Azure untuk membuat fungsi penyimpanan kita. Untuk membuat fungsi Storage, navigasikan kembali ke bagian fungsi baru dan pilih QueueTrigger – templat JavaScript (Gambar 4-32). Biarkan koneksi akun storage Anda apa adanya, dan ubah nama antrean menjadi apa pun yang sesuai untuk antrean Anda. Saya memilih orders karena antrean saya akan menjadi kumpulan order.



Gambar 4.32. Buat fungsi yang dipicu Antrean di Node.js.

Saya menamai fungsi saya Storage untuk memisahkannya dari fungsi lain yang telah kita buat. Dengan membuat fungsi ini, Anda akan mendapatkan fungsi antrean yang dapat Anda uji menggunakan kotak uji di sebelah kanan. Anda dapat membuat pesan uji dan menjalankannya untuk melihat apakah fungsi tersebut berfungsi. Namun, kita belum menyiapkan antrean, jadi kita perlu melakukan ini. Untuk memulai, klik tab Dokumentasi di bagian bawah bilah integrasi Anda (Gambar 4.33).

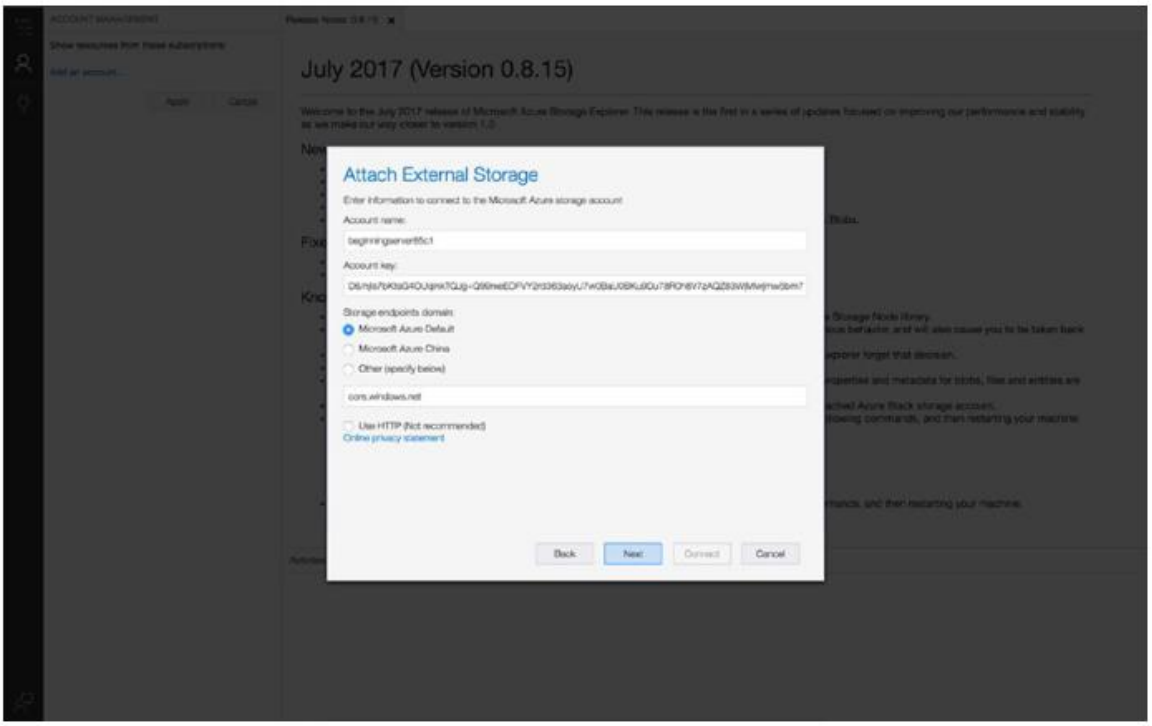


Gambar 4.33. Dapatkan Kunci Akun dan string Koneksi dari dokumentasi integrasi Anda

Tab Dokumentasi ini berisi nama Akun, Kunci Akun, dan string Koneksi untuk fungsi Anda. Anda akan menggunakan nama Akun dan kunci untuk mengatur koneksi antara Antrean dan fungsi. Jadi, salin nilai-nilai ini dan simpan. Kita akan membuat antrean menggunakan Microsoft Azure Storage Explorer. Anda perlu mengunduh alat ini agar dapat menggunakannya. Kita akan membahas lebih lanjut tentang Azure Storage Explorer sebelum menggunakannya.

Microsoft Azure Storage Explorer

Kita akan menggunakan alat Microsoft Azure Storage Explorer untuk mengakses dan memanipulasi data kita. Microsoft Azure Storage Explorer (saat ini dalam mode Pratinjau) adalah aplikasi mandiri dari Microsoft yang memungkinkan Anda bekerja dengan mudah dengan data Azure Storage di Windows, macOS, dan Linux. Untuk menginstal alat ini, buka http://storageexplorer.com/ dan klik paket unduhan yang sesuai untuk komputer Anda. Setelah Storage Explorer diunduh, Anda perlu terhubung ke penyimpanan Azure. Klik opsi ketiga yang disediakan (Gunakan Nama dan Kunci Akun Penyimpanan) dan berikan kredensial yang kami terima dari portal sebelumnya. Gambar 4-34 menunjukkan seperti apa tampilan lampiran penyimpanan Anda.

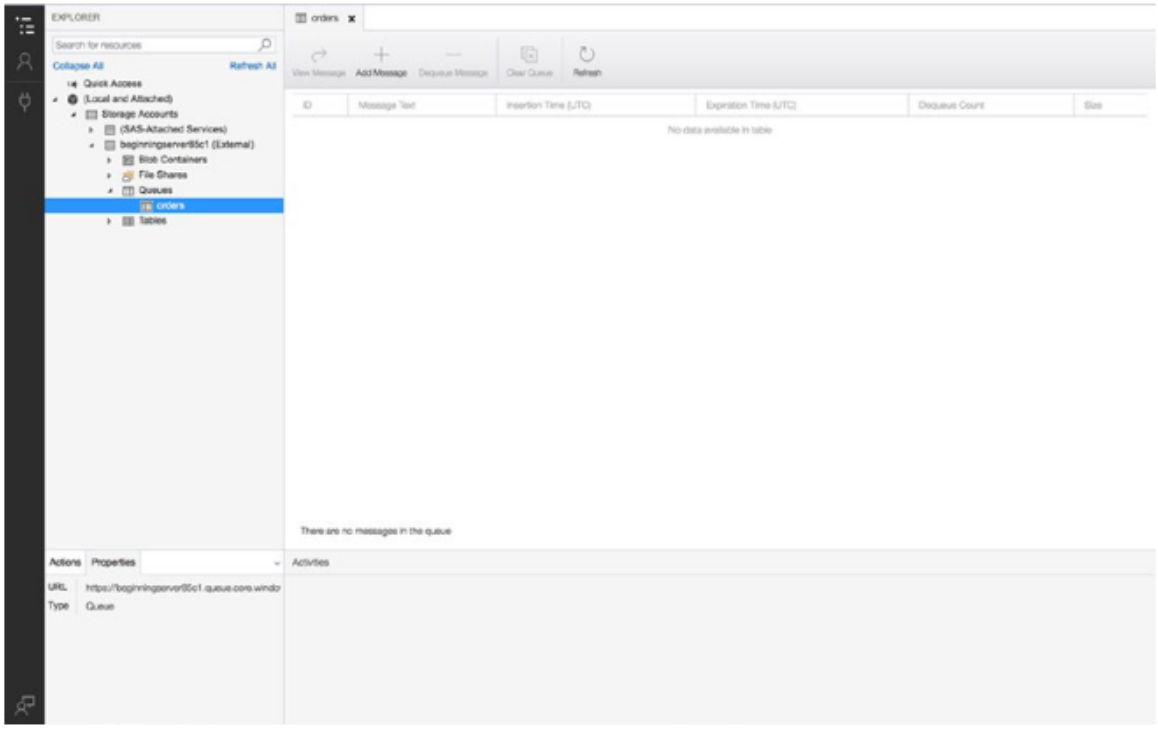


Gambar 4.34. Gunakan informasi akun untuk membuat koneksi ke Azure Storage kita

Klik melalui pengaturan lainnya, dan ini akan membuat koneksi ke akun Azure Anda dan berbagai kapabilitas Storage Anda. Saat koneksi terbentuk, Anda akan melihat aplikasi Anda tercantum di bawah Storage Accounts dan akan melihat opsi untuk Queues, Blob Containers, File Shares, dan Tables.

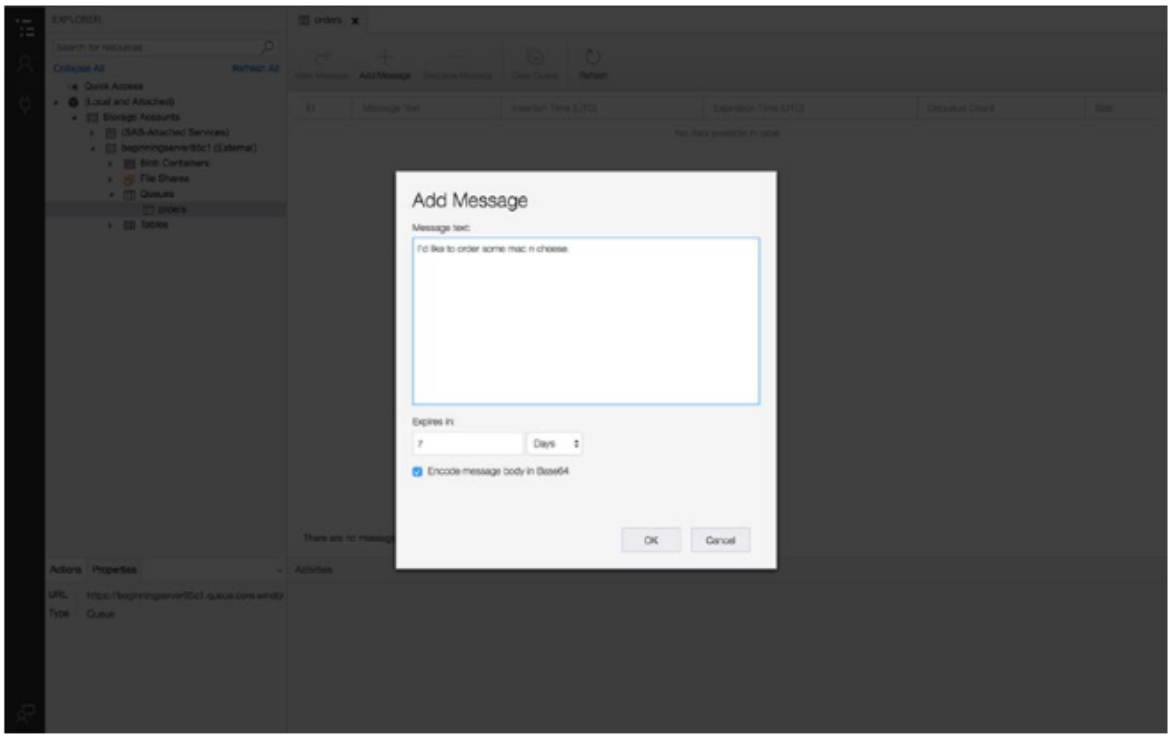
Saat pertama kali mulai bekerja, saya diberi tugas proyek Azure Machine Learning yang menggunakan data dari SQL Database. Salah satu tugas pertama saya adalah menghosting data ini di Azure dan saya melakukannya melalui SQL Server. Itu benar-benar mimpi buruk. Alat ini sebenarnya sangat canggih dan memungkinkan Anda memanipulasi data dengan mudah di satu tempat. Satu-satunya hal yang sedikit sulit adalah bahwa ini adalah alat yang sama sekali baru yang Anda perlukan secara lokal untuk mengakses kapabilitas ini. Saya lebih suka alat ini dihosting di Azure tetapi ini dapat berubah, mengingat alat ini baru dalam mode Pratinjau sekarang. Sementara itu, inilah yang saya sarankan untuk digunakan saat Anda berinteraksi dengan penyimpanan Azure.

Di bawah Queues, klik kanan dan pilih Create a Queue. Berikan nama yang sama dengan yang Anda berikan di pengaturan fungsi Azure Anda (“pesanan”). Kita dapat menguji fungsi templat dengan mengeklik Tambahkan Pesan dan menambahkan pesan ke antrean kita. Ini akan memicu fungsi kita; lalu menghapus pesan dari antrean kita. Gambar 4-35 mengilustrasikan seperti apa seharusnya antrean kita.



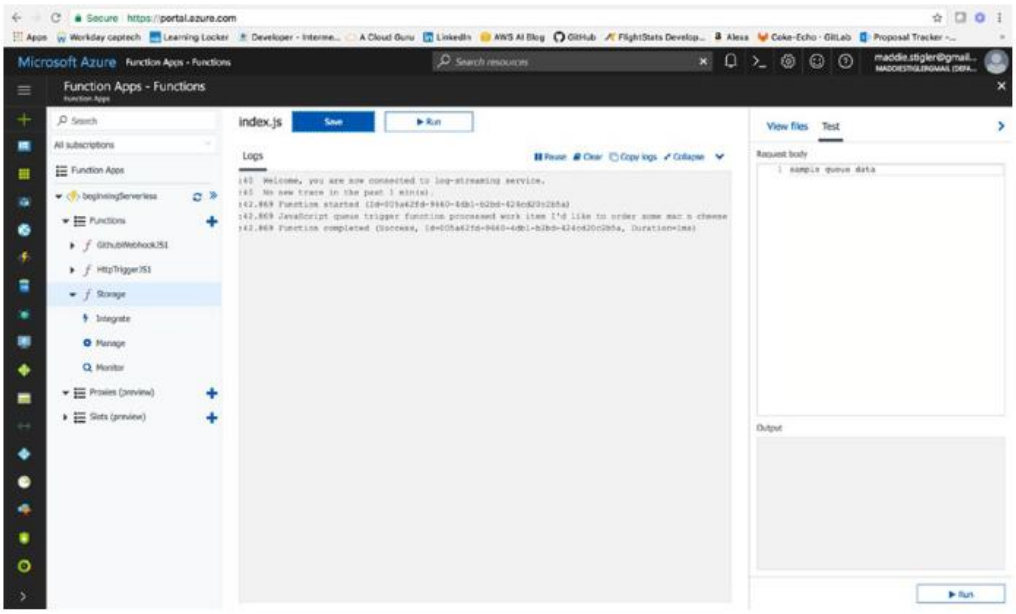
Gambar 4.35. Kita dapat menguji fungsi tersebut dengan menambahkan item ke antrean kita

Untuk pesan pengujian saya, saya hanya memberikan teks ke antrean dan mengirimkannya (Gambar 4.36).



Gambar 4.36. Tambahkan pesan ke antrean Anda

Kita dapat kembali ke portal Azure dan memeriksa log fungsi kita untuk memastikan pesan kita telah terkirim dan diterima. Jika berhasil, Anda akan melihat pesan seperti yang ditunjukkan pada Gambar 4.37.



Gambar 4.37. Log keberhasilan dari pengujian Antrean kami

Jika kami telah mengonfirmasi bahwa antrean kami terhubung ke Azure dan memicu fungsi kami, kami dapat melanjutkan untuk menyelesaikan file indeks, fungsi, dan

package.json kami. Menyelesaikan Fungsi Kami untuk menulis fungsi kami, kami akan menggunakan titik akhir HTTP POST dari latihan sebelumnya untuk mengirim perintah permintaan ke sana. Jika Anda ingat, URL ini adalah:

https://beginningserverless.azurewebsites.net/api/orders/

Kami akan menggunakan paket permintaan Node untuk melayani permintaan ini. Untuk melakukannya, kami harus membuat file package.json lain dan memasukkannya ke dalam dependensi kami. Package.json saya terlihat seperti ini:

```
{
  "name": "azure-nodejs",
  "version": "1.0.0",
  "description": "Azure Functions for Storage Trigger",
  "main": "handler.js",
  "keywords": [
  "azure",
  "serverless"
],
"dependencies": {
"request":"^2.81.0"
}
}
```

Kita perlu mengunggah berkas ini ke proyek kita di aplikasi Azure kita. Kemudian kita perlu kembali ke dasbor Kudu dan melakukan penginstalan NPM di dalam folder proyek kita untuk memastikan modul node yang tepat disertakan untuk menjalankan aplikasi kita. Setelah ini selesai, kita dapat melanjutkan dan menyelesaikan fungsi kita.

Modul node Permintaan dirancang untuk menjadi cara paling sederhana yang memungkinkan untuk melakukan panggilan http. Modul ini mendukung HTTPS dan mengikuti pengalihan secara default. Argumen pertama dapat berupa URL atau objek opsi. Satu-satunya opsi yang diperlukan adalah uri; yang lainnya bersifat opsional.

- uri || url: URI berkualifikasi lengkap atau objek URL yang diurai.
- baseUrl: string uri berkualifikasi lengkap yang digunakan sebagai url dasar. Paling berguna dengan default request., misalnya saat Anda ingin melakukan banyak permintaan ke domain yang sama. Jika baseUrl adalah https://example.com/api/, maka permintaan /end/point?test=true akan mengambil https://example.com/api/end/point?test=true. Bila baseUrl diberikan, uri juga harus berupa string.
- metode: metode HTTP (default: GET)
- headers: header HTTP (default: {})

Untuk informasi lebih lanjut tentang modul Request, kunjungi https://www.npmjs.com/package/request.

Mengikuti parameter request yang dijelaskan, kita akan membuat sebuah JSON pilihan dengan nilai kita sendiri yang diisi dan kita akan menyetel variabel JSON menjadi

myQueueItem yang masuk. Daftar 4.2 menunjukkan fungsi bawaan untuk menangani permintaan Queue dan mengirimkannya ke fungsi berikutnya.

Daftar 4-2. Sebuah fungsi yang mengambil item queue dan mengirimkannya ke fungsi HTTP kita

```
   var request=require('request');

module.exports = function (context, myQueueItem) {
     context.log('JavaScript queue trigger function processed work item',
       myQueueItem);
     if(myQueueItem.name && myQueueItem.order && myQueueItem.phoneNumber) {

       var options = {
         url: 'https://beginningServerless.azurewebsites.net/api/orders/',
         method: 'POST',
         headers: {
         'Content-Type': 'application/json'
         }

       json: myQueueItem
     };

     request(options, function(err, res, body) {
       if (res && (res.statusCode === 200 || res.statusCode === 201)) {
       console.log(body);
       }
     });
   }
     else (
       console.log("Nothing to process")
     )
     context.done();
};
```

Setelah kita menerapkan fungsi ini, kita dapat mengujinya dengan sekali lagi membuat pesan dalam antrean kita dan mengirimkannya. Jika semuanya sudah diatur dengan benar, kita seharusnya dapat masuk ke log kita untuk fungsi Storage dan fungsi HTTP dan melihat pesan kita masuk ke sana. Anda juga akan terus menerima teks untuk setiap pesan yang Anda buat.

■ Kiat Jika Anda mengalami masalah saat melihat perubahan, pastikan titik akhir Anda sudah benar dan nama antrean Anda sama di Storage Explorer dan di Azure. Konfigurasi lebih mungkin menjadi masalah Anda daripada sesuatu dalam kode Anda, dan di dua tempat inilah konfigurasi benar-benar penting.

Hal lain yang perlu diingat adalah bahwa cara kita mengatur fungsi HTTP adalah dengan menerima isi dalam permintaan POST dengan struktur tertentu. Strukturnya adalah:

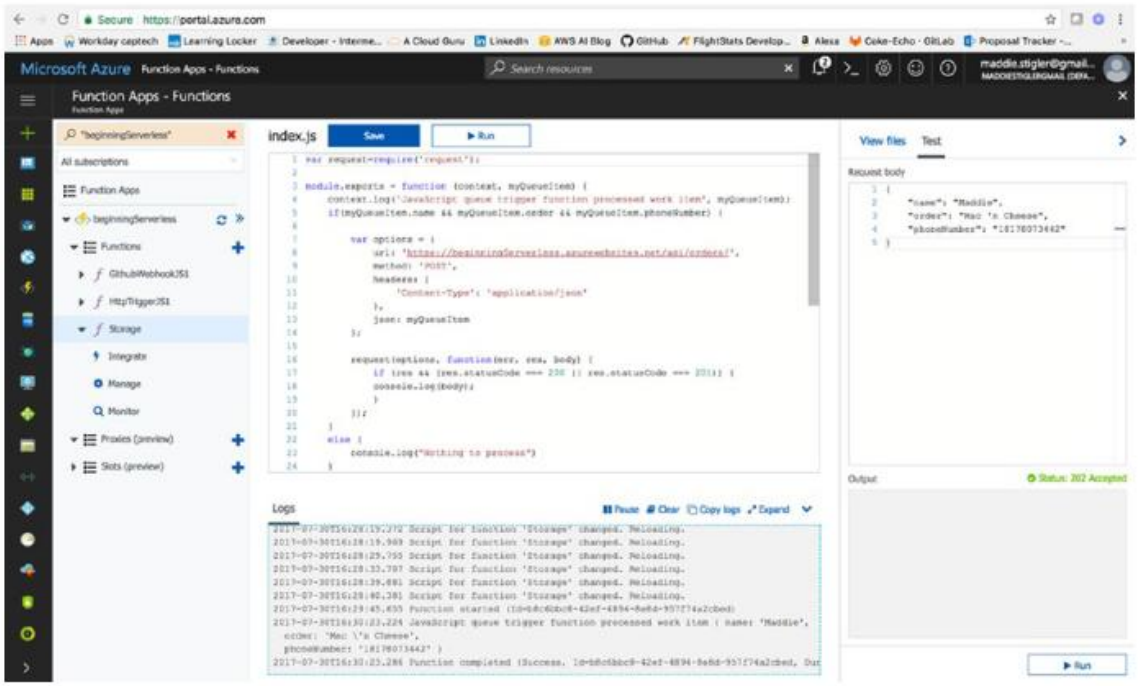
```
{
   "name": "",
   "order": "",
   "phoneNumber": "1XXXXXXXXXX"
```

```
}
```

Jadi agar fungsi HTTP kita dapat menangani pesan antrean masuk dengan benar, salah satu dari dua hal harus terjadi.

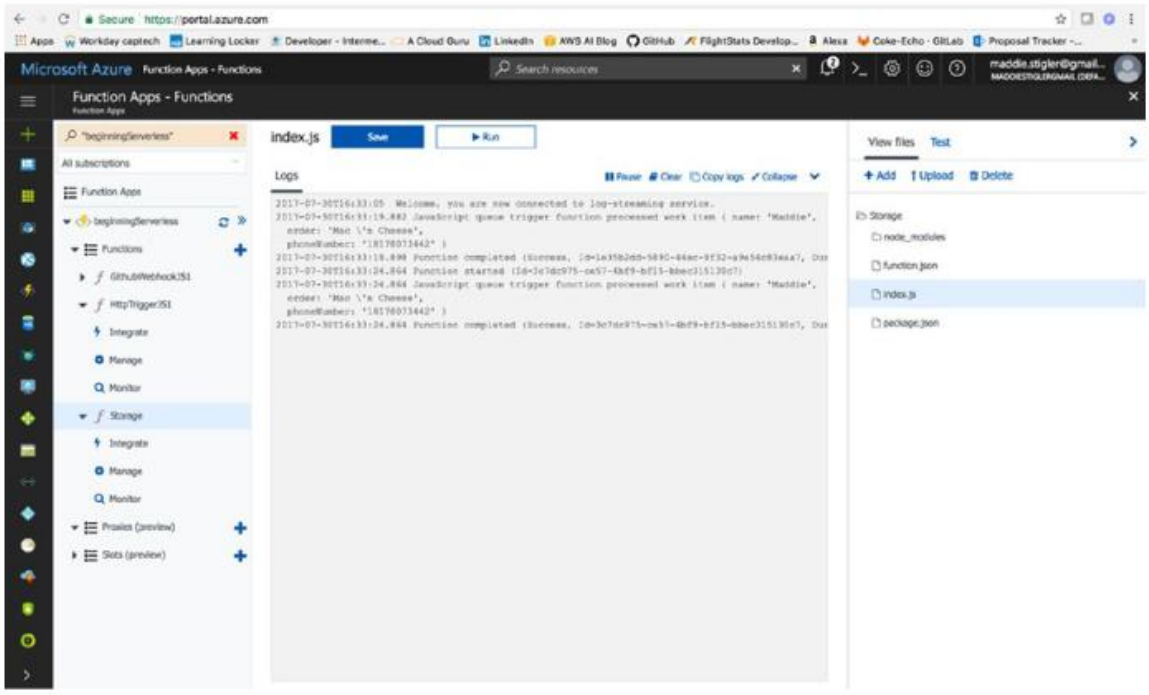
1. Kita harus menyusun semua pesan antrean seolah-olah pesan tersebut adalah objek JSON dan memastikan setiap item berisi nama, pesanan, dan nomor telepon.
2. Kita harus mengerjakan ulang fungsi HTTP kita untuk menerima pesan sebagai string dan mengirim pesan tersebut secara langsung.

Saya memilih untuk menyusun pesan antrean saya sebagai nilai JSON untuk menguji dan memastikan pesan tersebut berhasil melewati alur. Setelah menambahkan pesan awal, pertama-tama saya masuk ke fungsi penyimpanan saya dan memeriksa log untuk memastikan pesan tersebut diterima dan dikirim. Gambar 4.38 menunjukkan hasil saya.



Gambar 4.38. Fungsi penyimpanan telah dipicu oleh pesan dalam antrean kami

Setelah mengonfirmasi bahwa pesan tersebut berhasil mencapai langkah pertama, saya melihat log fungsi HTTP kami untuk memastikan pesan tersebut berhasil melewati langkah kedua juga (Gambar 4.39).



Gambar 4.39. Log fungsi HTTP menunjukkan pesan yang diterima dan dikirim melalui teks

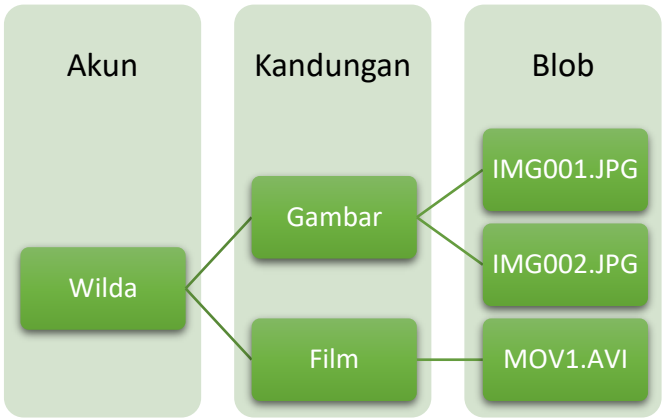
Tak lama kemudian, saya menerima pesan teks berisi pesan yang telah saya tambahkan ke antrean. Seperti yang Anda lihat, cukup cepat dan mudah untuk menyiapkan fungsi pemicu penyimpanan di Azure, terutama menggunakan penjelajah Penyimpanan. Sistem notifikasi pengiriman makanan adalah salah satu aplikasi penyimpanan sebagai pemicu, tetapi masih banyak lagi.

Tambahkan Ke Fungsi Anda

Untuk latihan ini, kita akan menyusun ulang alur proses aplikasi kita. Permintaan HTTP sekarang akan menjadi langkah pertama dan fungsi peristiwa penyimpanan akan menjadi langkah kedua. Kita akan terus menggunakan permintaan POST ke fungsi HTTP kita, tetapi alih-alih mengirim teks, kita akan memperbarui antrean kita dengan pesan yang diurai yang ingin kita kirim. Saat pesan ini ditambahkan ke antrean, fungsi Lambda kedua akan terpicu yang akan mengirim teks tersebut ke pelanggan. Ini akan memberi Anda paparan untuk memanipulasi data antrean di Node dalam fungsi Azure. Jika Anda ingin melangkah lebih jauh, Anda bahkan dapat melihat penyimpanan Blob dan menambahkan gambar ke penyimpanan untuk mengirimkannya kepada pengguna juga. Penyimpanan blob Azure sangat mirip dengan AWS S3, yang kita gunakan di bab terakhir. Penggunaan umum penyimpanan Blob meliputi hal-hal berikut:

- Menyajikan gambar atau dokumen langsung ke browser
- Menyimpan file untuk akses terdistribusi
- Streaming video dan audio
- Menyimpan data untuk pencadangan dan pemulihan, pemulihan bencana, dan pengarsipan
- Menyimpan data untuk analisis oleh layanan lokal atau yang dihosting Azure

Diagram berikut akan memberi Anda pemahaman yang lebih baik tentang cara kerja penyimpanan Blob di Azure.



Untuk mempelajari selengkapnya, kunjungi https://docs.microsoft.com/en-us/azure/storage/storage-nodejs-how-to-use-blob-storage.

4.7 KESIMPULAN

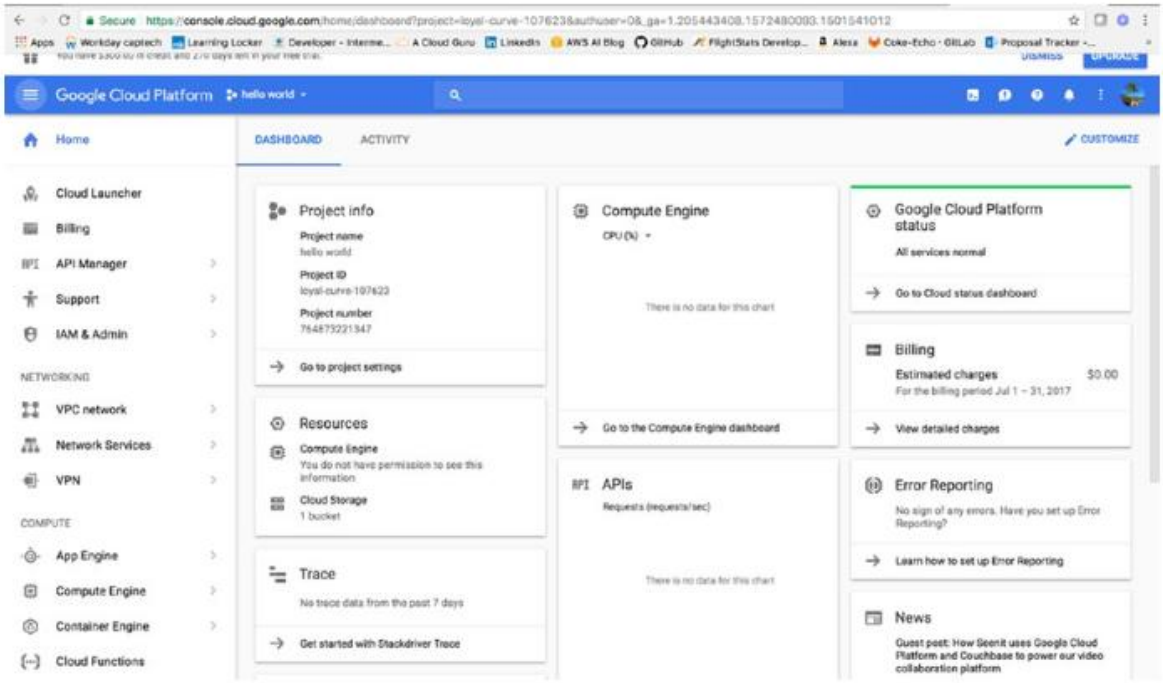
Dalam bab ini, kami menjelajahi pemicu HTTP dan pemicu penyimpanan di Azure Functions. Kami melihat beberapa perbedaan antara pengembangan di Azure dan pengembangan di AWS dan menemukan banyak perbedaan dalam proses pembuatan fungsi. Kami melihat WebHooks dan permintaan HTTP untuk pemicu API dan melihat bagaimana kami dapat menggunakan keduanya dalam aplikasi sehari-hari. Pada titik ini, Anda seharusnya merasa nyaman menulis fungsi node dan menyebarkannya di AWS atau Azure. Anda seharusnya memiliki pemahaman yang lebih baik tentang konfigurasi Azure (dan lokasi konfigurasi ini) dibandingkan dengan AWS, dan Anda seharusnya memiliki gagasan yang lebih jelas tentang mengapa Anda lebih memilih satu vendor daripada yang lain. Dalam bab berikutnya, kami akan terus membangun aplikasi Serverless. Kami akan melihat Google Cloud dan membangun beberapa fungsi Cloud, menjelajahi UI, dan terus menganalisis perbedaan dalam vendor.

BAB 5
Google Cloud

5.1 PENDAHULUAN

Dalam bab ini, kita akan menggunakan Google Cloud Platform untuk mengembangkan aplikasi tanpa server dan mengeksplorasi perbedaan antara Google sebagai penyedia cloud dan Azure serta Amazon sebagai penyedia cloud. Untuk membuat aplikasi ini, kita akan menggunakan fungsi Cloud, permintaan HTTP, bucket Google Cloud Storage, dan topik Cloud Pub/Sub.

Kita juga akan mengeksplorasi berbagai kasus penggunaan untuk fungsi HTTP dan penyimpanan yang dipicu sehingga kita memperoleh pemahaman yang lebih baik tentang luas dan dalamnya aplikasi tanpa server. Di akhir bab ini, kita akan memiliki tiga aplikasi tanpa server dan pengalaman dengan beberapa layanan Google Cloud. Kita akan mulai menyelidiki UI Google Cloud sebelum mengembangkan fungsi kita.



Gambar 5.1. Google Cloud Platform menawarkan banyak layanan mulai dari layanan komputasi hingga layanan jaringan dan penyimpanan. Kita akan menjelajahi fungsi, pengelola API, penyimpanan, dan IAM.

Saya akan merekomendasikan beberapa eksplorasi mandiri di luar bagian ini untuk benar-benar membiasakan diri dengan cara kerja Google Cloud. Azure dan AWS memiliki banyak perbedaan, tetapi bagi saya, Google jelas lebih sulit untuk dipelajari. Saya rasa hal ini banyak berkaitan dengan kenyamanan saya dengan AWS dan konsep baru yang diperkenalkan di Google Cloud, jadi kami akan memastikan untuk mempelajari perbedaan ini

seiring berjalannya waktu. Untuk memulai, Anda memerlukan akun Google untuk mengakses UI Google Cloud. Setelah masuk ke konsol di https://console.cloud.google.com/, dasbor Anda akan menampilkan daftar layanan Google yang dipisahkan berdasarkan fungsionalitas (sangat mirip dengan AWS dan Azure). Gambar 5.1 memberi Anda gambaran tentang semua layanan yang disediakan Google Cloud.

Pada titik ini, Anda telah mempelajari banyak istilah berbeda untuk konsep yang sama di Amazon dan Azure. Tabel 5.1 menyediakan peta konsep untuk ketiga platform tersebut agar terminologi tetap jelas di seluruh bab ini.

Tabel 5.1. Tabel Konsep Penyedia Cloud

Konsep	Istilah AWS	Istilah Azure	Istilah Google
Pusat Data	Wilayah	Wilayah	Wilayah
Pusat Data Abstrak	Zona Ketersediaan	Zona Ketersediaan	Zona
Caching Tepi	CloudFront		POP (beberapa layanan)
Computing: IaaS	EC2	Mesin Virtual	Mesin Komputasi Google
Computing: PaaS	Beanstalk Elastis	Layanan Aplikasi, Layanan Cloud	Mesin Aplikasi Google
Computing: Kontainer	Layanan Kontainer EC2	Layanan Kontainer Azure, Service Fabric	Mesin Kontainer Google
Jaringan: Load balancer	Load balancer Elastis	Load balancer	Load balancing
Jaringan: Mengintip	Sambungan Langsung	Rute Ekspres	Interkoneksi Awan
Jaringan: DNS	Rute 53	DNS	DNS Awan
Penyimpanan: Penyimpanan Objek	S3	Gumpal	Penyimpanan Awan
Penyimpanan: Penyimpanan File	Sistem File Elastis	Penyimpanan Berkas	Benar
Basis Data: RDBMS	RDS	Basis Data SQL	Awan SQL
Basis Data: NoSQL	DynamoDB	Penyimpanan Meja	Penyimpanan Data Cloud, Meja Besar
Pesan	SNS	Bus Layanan	Pub/Sub Awan

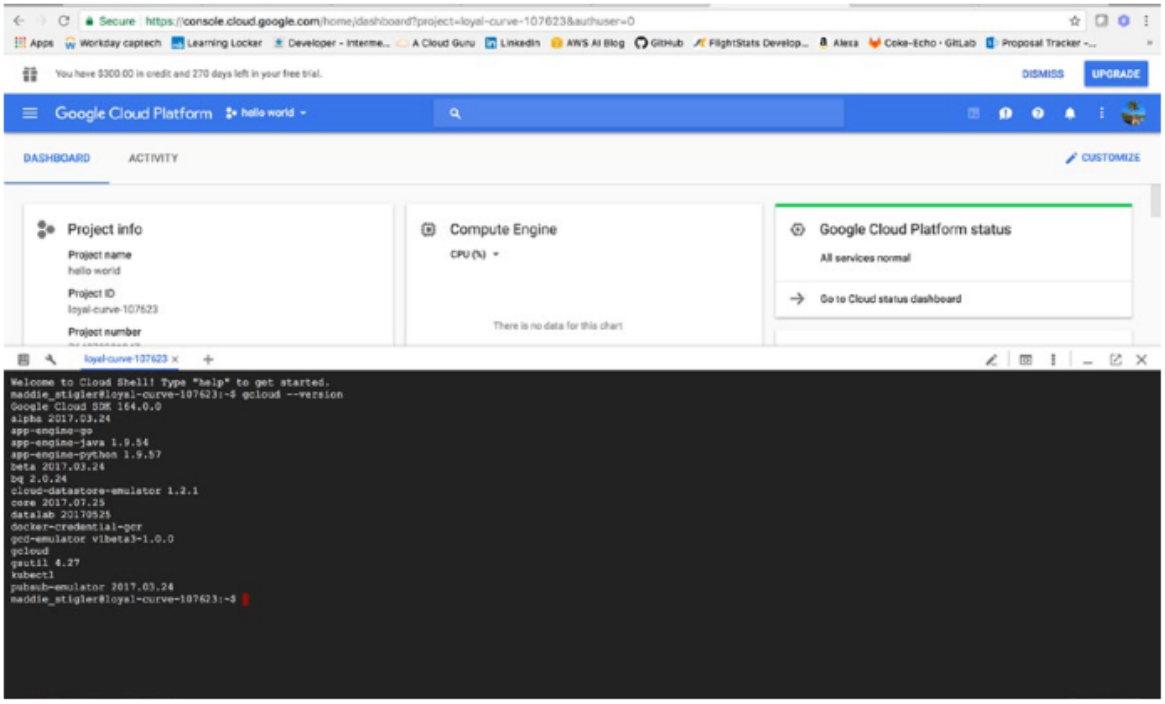
Navigasi

Dasbor untuk Google Cloud cukup mudah dan sangat mudah diakses. Seperti di Azure, Anda dapat mengonfigurasi dan menyesuaikan tata letak dasbor sesuai keinginan Anda. Mirip dengan Azure, Google Cloud menyediakan Google Cloud Shell bagi pengguna. Ini dapat ditemukan di kanan atas bilah biru di dasbor Anda. Google Cloud Shell adalah lingkungan untuk mengelola sumber daya yang dihosting di Google Cloud Platform.

Shell sudah memiliki Cloud SDK yang siap digunakan. Saya suka ini karena tidak banyak pengaturan di komputer lokal Anda dan membuatnya sangat mudah diakses melalui portal. Untuk mengujinya, Anda dapat membuka Cloud Shell dan mengetik:

```
gcloud –version
```

Ini akan memberi Anda daftar komponen Cloud SDK yang terinstal dan versinya (Gambar 5-2).



Gambar 5.2. Cloud Shell dilengkapi dengan Cloud SDK yang aktif dan berjalan, beserta beberapa komponen

Di dasbor, kita memiliki akses ke info proyek, sumber daya yang digunakan, penagihan, dan pelaporan kesalahan. Proyek spesifik yang Anda ikuti ditampilkan di kiri atas bilah biru. Saat ini, saya berada di proyek Hello World yang saya buat untuk tutorial ini. Untuk membuat proyek baru, Anda dapat mengeklik panah drop-down pada bilah biru dan modal proyek akan muncul. Jika Anda mengeklik tanda plus, Anda dapat membuat proyek baru dan menambahkannya ke dasbor Anda. Jangan ragu untuk melakukan ini untuk fungsi Hello World sekarang.

Setiap sumber daya Cloud Platform yang Anda alokasikan dan gunakan harus menjadi milik proyek. Saya ingin menganggap proyek sebagai entitas pengorganisasian untuk apa yang saya bangun. Sebuah proyek terdiri dari pengaturan, izin, dan metadata lain yang menjelaskan aplikasi Anda. Sumber daya dalam satu proyek dapat bekerja sama dengan mudah; misalnya dengan berkomunikasi melalui jaringan internal, sesuai dengan aturan wilayah dan zona. Sumber daya yang dimiliki setiap proyek tetap terpisah di seluruh batas proyek; Anda hanya dapat menghubungkannya melalui koneksi jaringan eksternal.

Dengan mengeklik kumpulan titik di samping gambar profil Anda, Anda memiliki akses ke daftar opsi lain termasuk pengaturan Proyek untuk proyek yang diberikan. Saat Anda mengeklik ini, Anda akan melihat ID proyek dan nomor proyek. Ini adalah dua cara untuk mengidentifikasi proyek Anda. Saat Anda bekerja dengan Cloud Platform, Anda akan menggunakan pengenal ini di baris perintah dan panggilan API tertentu.

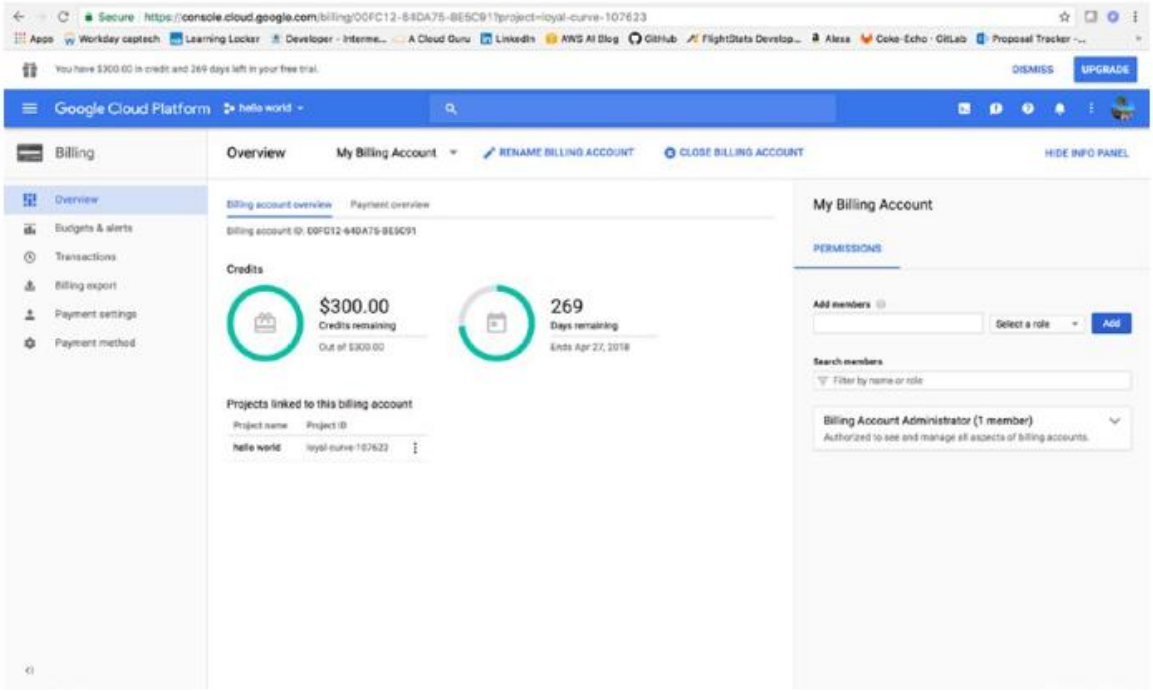
Nomor proyek ditetapkan secara otomatis saat Anda membuat proyek.

ID proyek adalah pengenal unik untuk sebuah proyek. Saat pertama kali membuat proyek, Anda dapat menerima ID proyek yang dibuat secara default atau membuatnya sendiri. ID proyek tidak dapat diubah setelah proyek dibuat, jadi jika Anda membuat proyek baru, pastikan untuk memilih ID yang akan berfungsi selama masa pakai proyek.

Di panel yang sama, Anda dapat mengakses izin dan peran IAM untuk proyek tersebut. Ini adalah fitur yang dimiliki Azure dan Google yang dimiliki AWS secara berbeda. Di Azure dan Google, layanan dan sumber daya dibagi berdasarkan proyek dari konsol ke bawah. Di AWS, Anda dapat melakukan ini, tetapi ini bukan sesuatu yang disediakan secara langsung. Ini berarti bahwa saat menavigasi melalui AWS, Anda akan melihat sumber daya untuk berbagai proyek di ruang yang sama dan harus menentukan kebijakan IAM untuk proyek tertentu. Google dan Azure memberlakukan pemisahan proyek ini. Setiap perubahan yang Anda buat pada kebijakan Google atau Azure untuk suatu proyek berlaku untuk keseluruhan proyek.

Harga

Harga dan penagihan untuk Google Cloud dapat diakses melalui portal di petak Penagihan. Dasbor Penagihan memberi Anda akses ke ikhtisar laporan penagihan, anggaran dan peringatan, transaksi, ekspor, pengaturan pembayaran, dan konfigurasi penagihan. Dalam gambar 5.3 memberikan ikhtisar portal Penagihan.

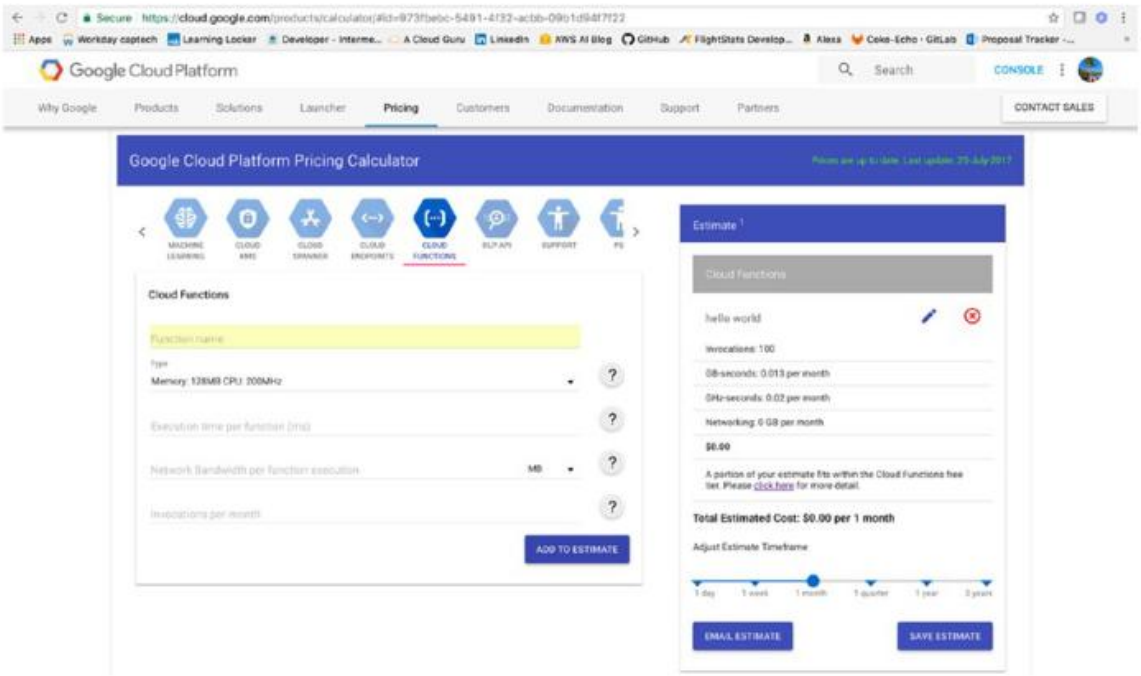


Gambar 5.3. Panel Penagihan memberi Anda ikhtisar tentang laporan penagihan Anda serta akses ke izin ke akun penagihan Anda

Anda dapat membuat anggaran yang dilampirkan ke proyek tertentu dengan jumlah tertentu untuk berbagai peringatan. Peringatan ini akan mengirimkan email kepada Anda saat nilai minimum yang dikirimkan telah tercapai. Google juga menyediakan kalkulator harga

yang intuitif dan mudah digunakan. Penyedia lain yang telah kami bahas juga menyediakan ini.

Jika Anda hanya mengerjakan proyek independen, kalkulator biaya mungkin tidak terlalu penting bagi Anda, tetapi jika Anda pernah mengerjakan arsitektur dan pengembangan Cloud untuk orang lain, kalkulator biaya menjadi sangat penting. Kalkulator Google memungkinkan Anda memilih layanan yang menurut Anda akan digunakan dan menentukan semua spesifikasi untuk masing-masing layanan sehingga Anda akan mendapatkan perhitungan harga yang lebih akurat. Gambar 5.4 menunjukkan perkiraan untuk fungsi cloud Hello World yang akan kita buat bersama.



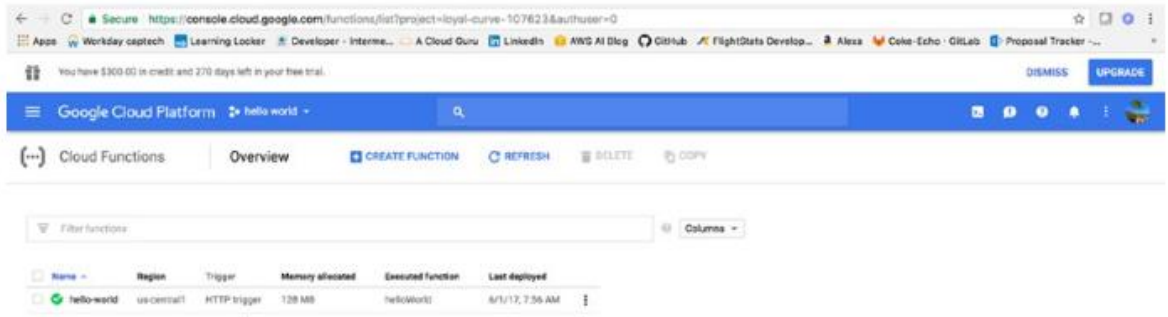
Gambar 5.4. Kalkulator Biaya memungkinkan Anda memilih layanan, informasi, dan menambahkan estimasi terkini untuk jumlah uang per bulan

Harga Cloud Functions masih sangat murah. Cloud Functions diberi harga berdasarkan lamanya fungsi Anda berjalan, seberapa sering fungsi tersebut dipanggil, dan berapa banyak sumber daya yang Anda sediakan untuk fungsi tersebut. Jika fungsi Anda membuat permintaan jaringan keluar, ada juga biaya transfer data tambahan. Cloud Functions menyertakan tingkatan gratis yang berlaku terus-menerus untuk memungkinkan Anda bereksperimen dengan platform tanpa biaya.

5.2 CLOUD FUNCTIONS

Cloud Functions dapat diakses di panel kiri di bawah Compute and Functions. Google juga memberi Anda opsi penyematan, sehingga Anda dapat menyimpan akses ke Functions di bagian atas layar di mana pun Anda berada di konsol. Saya melakukan ini untuk

memudahkan akses. Dasbor Cloud Functions memungkinkan Anda melihat fungsi yang sedang aktif, wilayah, pemicu, alokasi memori per fungsi, dan penerapan terakhir. Di sinilah Anda juga dapat membuat atau menghapus fungsi. Gambar 5-5 memberikan gambaran umum dasbor Google Cloud.



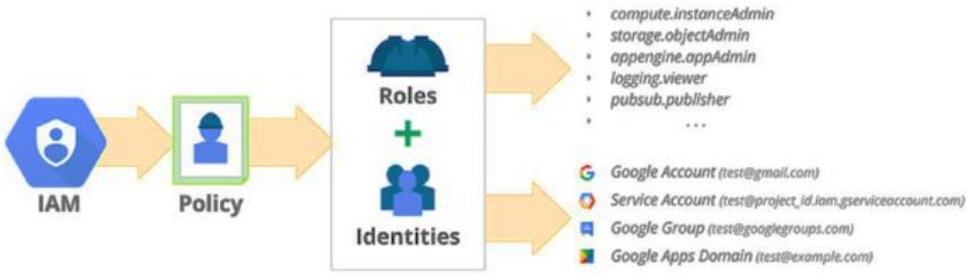
Gambar 5.5. Dasbor fungsi Cloud menunjukkan fungsi aktif dan metriknya

Kita akan membahas pembuatan fungsi saat memulai aplikasi Hello World, tetapi untuk saat ini, saya akan memberikan latar belakang singkat tentang perbedaan yang lebih spesifik dalam fungsi Google Cloud yang ditetapkan Google Cloud dalam dokumentasinya.

Keamanan IAM

Google Cloud Identity and Access Management (Cloud IAM) memungkinkan Anda membuat dan mengelola izin untuk sumber daya Google Cloud Platform. Cloud IAM menyatukan kontrol akses untuk layanan Cloud Platform ke dalam satu sistem dan menyajikan serangkaian operasi yang konsisten.

Anda dapat mengatur kontrol akses menggunakan peran di tingkat proyek. Tetapkan peran ke anggota proyek atau akun layanan untuk menentukan tingkat akses ke proyek Google Cloud Platform dan sumber dayanya. Secara default, semua proyek Google Cloud Platform dilengkapi dengan satu pengguna: pembuat proyek asli. Tidak ada pengguna lain yang memiliki akses ke proyek dan atau fungsinya, hingga pengguna ditambahkan sebagai anggota tim proyek. Gambar 5-6 menunjukkan alur IAM yang terkait dengan Google Cloud.

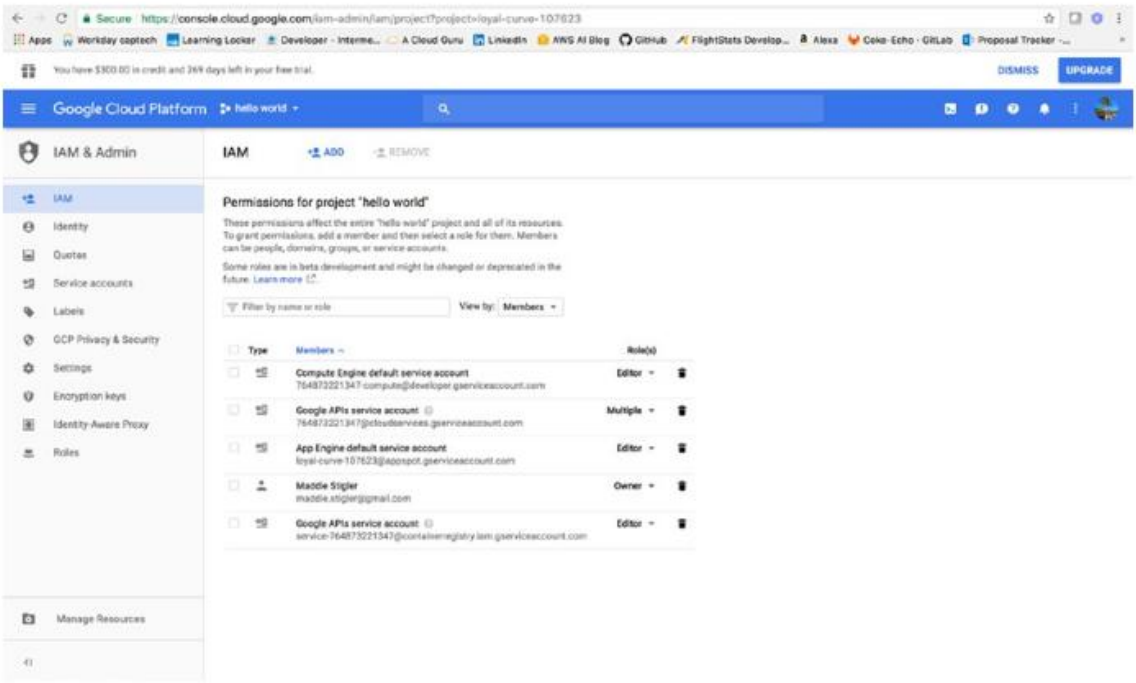


Gambar 5.6 Layanan IAM melampirkan kebijakan ke peran dan identitas untuk menyediakan akses yang aman

Konsol IAM

Konsol IAM (Gambar 5.7) ditemukan di bawah daftar produk dan layanan di bilah Dasbor sebelah kiri. Konsol tersebut memberi Anda dasbor dengan izin IAM terkini untuk

proyek, identitas, kuota, akun layanan, label, privasi GCP, setelan, kunci enkripsi, proksi, dan peran Anda.



Gambar 5.7. Dasbor IAM memberi Anda ikhtisar semua sumber daya IAM Anda beserta lima langkah keamanan yang disarankan untuk Anda selesaikan

Dasbor mencantumkan izin untuk proyek tertentu yang sedang Anda ikuti saat ini. Izin ini memengaruhi seluruh proyek Hello World dan semua sumber dayanya. Untuk memberikan izin, tambahkan anggota lalu pilih peran untuk mereka. Anggota dapat berupa orang, domain, grup, atau akun layanan.

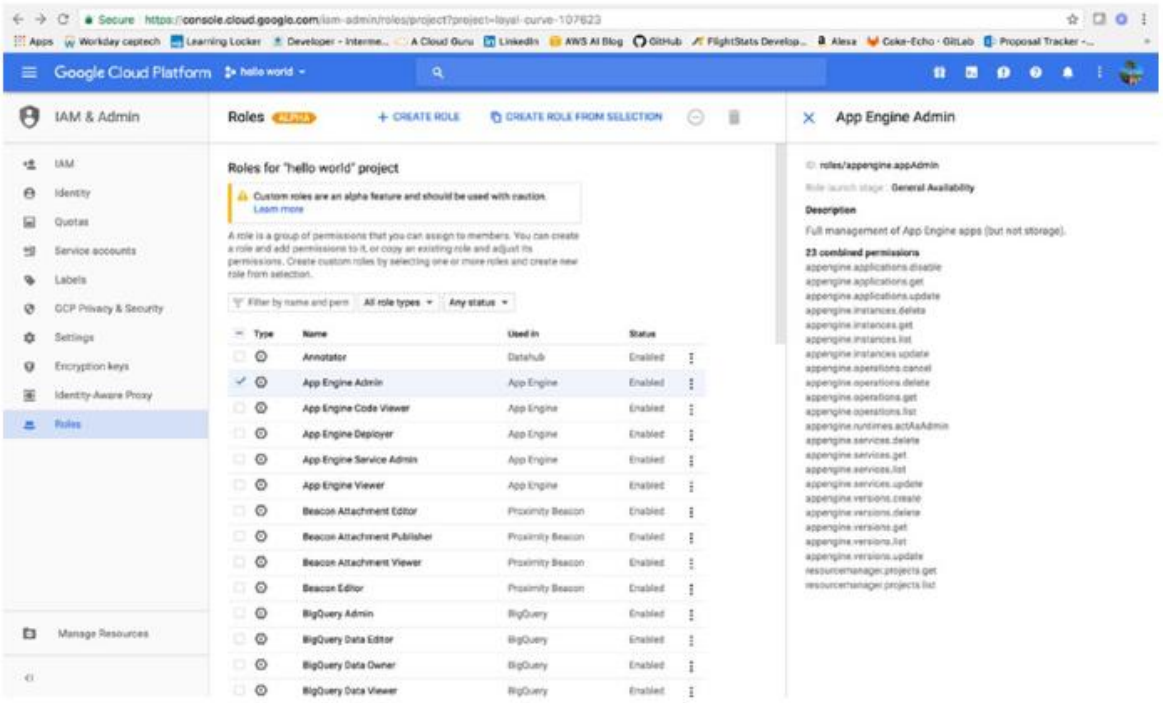
Peran

Dengan Cloud IAM, metode Cloud API mengharuskan identitas yang membuat permintaan API memiliki izin yang sesuai untuk menggunakan sumber daya. Anda dapat memberikan izin dengan memberikan peran kepada pengguna, grup, atau akun layanan. Ada tiga jenis peran yang akan kita bahas: peran primitif, peran yang telah ditetapkan sebelumnya, dan peran khusus.

Peran primitif adalah peran yang ada sebelum Cloud IAM. Pemilik, Editor, dan Penampil akan terus bekerja seperti sebelumnya. Peran-peran ini bersifat konsentris; yaitu, peran Pemilik mencakup izin dalam peran Editor, dan peran Editor mencakup izin dalam peran Penampil. Peran primitif dapat ditetapkan di tingkat proyek.

Cloud IAM menyediakan peran pradefinisi tambahan yang memberikan akses terperinci ke sumber daya Google Cloud Platform tertentu dan mencegah akses yang tidak diinginkan ke sumber daya lainnya. Anda dapat memberikan beberapa peran kepada pengguna yang sama. Misalnya, pengguna yang sama dapat memiliki peran Admin Jaringan dan Penampil Log pada suatu proyek dan juga memiliki peran Penerbit untuk topik Pub/Sub dalam proyek tersebut.

Selain peran pradefinisi, Cloud IAM juga menyediakan kemampuan untuk membuat peran yang disesuaikan. Anda dapat membuat peran IAM kustom dengan satu atau beberapa izin, lalu memberikan peran kustom tersebut kepada pengguna yang merupakan bagian dari organisasi Anda (Gambar 5.8).



Gambar 5.8. Daftar peran yang tersedia melalui Google Cloud. Anda juga dapat membuat peran.

Kebijakan

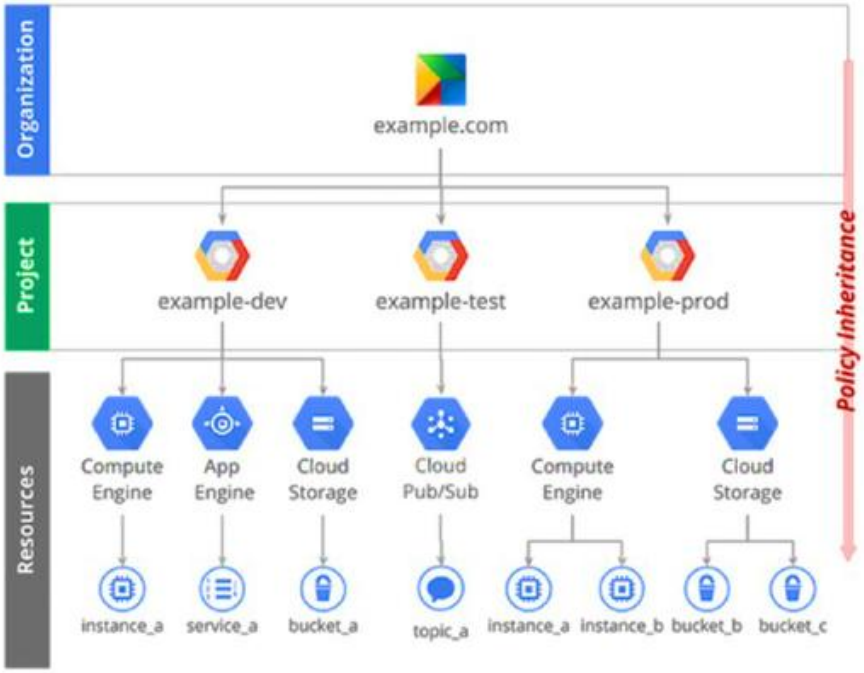
Anda dapat memberikan peran kepada pengguna dengan membuat kebijakan Cloud IAM, yang merupakan kumpulan pernyataan yang menentukan siapa yang memiliki jenis akses apa. Kebijakan dilampirkan ke sumber daya dan digunakan untuk memberlakukan kontrol akses setiap kali sumber daya tersebut diakses.

Kebijakan Cloud IAM diwakili oleh objek kebijakan. Kebijakan terdiri dari daftar pengikatan. Pengikatan mengikat daftar anggota ke peran. Cuplikan kode berikut menunjukkan struktur kebijakan Cloud IAM:

```
{
  "bindings": [
  {
    "role": "roles/owner",
    "members": [
      "user:maddie@example.com",
      "group:admins@example.com",
      "domain:google.com",
      "serviceAccount:my-other-app@appspot.gserviceaccount.com",
    ]
  },
```

```
    {
    "role": "roles/viewer",
    "members": ["user:maddie@example.com"]
    }
  ]
}
```

Hal terakhir yang perlu dipahami tentang kebijakan adalah bahwa Google Cloud mengaturnya secara hierarkis. Hirarki ini didefinisikan oleh Organization ä Projects ä Resources. Gambar 5.9 menunjukkan contoh hierarki sumber daya Cloud Platform. Node Organization adalah node akar dalam hierarki, Projects adalah turunan dari Organization, dan Resources lainnya adalah turunan dari Projects. Setiap sumber daya memiliki tepat satu induk.



Gambar 5.9. Daftar peran yang tersedia melalui Google Cloud. Anda juga dapat membuat peran.

5.3 KODE PERTAMA GOOGLE CLOUD

Sekarang setelah kita memiliki pemahaman yang lebih baik tentang Google Cloud dan kemampuannya, kita dapat mulai membuat aplikasi tanpa server menggunakan Google Functions. Jika Anda sudah menyiapkan proyek, SDK, Penagihan, dan API, Anda dapat melanjutkan ke bagian Hello World. Jika belum, luangkan waktu lima menit untuk menyiapkan semuanya.

Untuk menyiapkan proyek, buka halaman Proyek: https://console.cloud.google.com/project. Buat proyek Hello World baru di ruang ini. Kita juga perlu mengaktifkan penagihan. Lakukan langkah-langkah berikut untuk melakukannya:

1. Dari daftar proyek, pilih proyek untuk mengaktifkan kembali penagihan.
2. Buka menu sisi kiri konsol dan pilih Penagihan.

3. Klik Tautkan Akun Penagihan.
4. Klik Tetapkan Akun.

Berikutnya, kita ingin mengaktifkan Cloud Functions API:

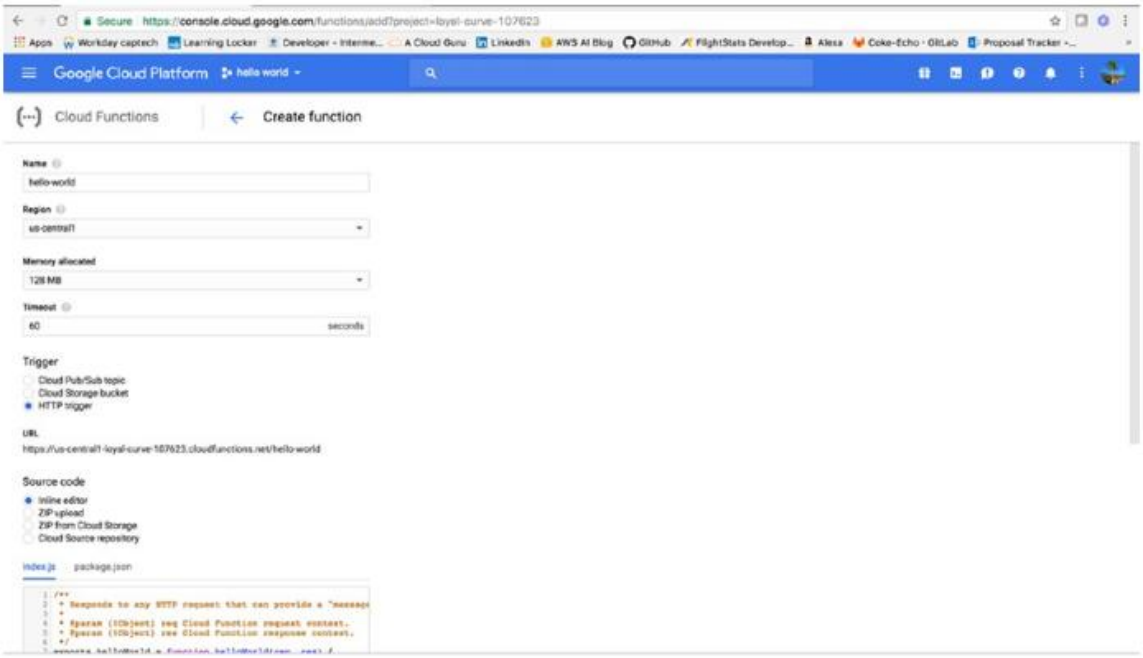
1. Dari daftar proyek, pilih proyek atau buat yang baru.
2. Jika halaman API Manager belum terbuka, buka menu sisi kiri konsol dan pilih API Manager, lalu pilih Library.
3. Klik Cloud Functions API.
4. Klik ENABLE.

Anda dapat menginstal Google Cloud SDK dengan mengunduh paket yang benar dari sini: https://cloud.google.com/sdk/docs/ Pada tahap ini, Anda seharusnya sudah memiliki semua prasyarat untuk mulai mengembangkan fungsi tanpa server.

Hello World

Kita akan mulai dengan membuat fungsi baru di konsol Cloud Functions. Untuk sampai di sini, navigasikan ke Cloud Functions dari dasbor. Seperti yang ditunjukkan pada Gambar 5-10, kita akan membuat fungsi baru, beri nama hello-world, dan berikan properti berikut:

- Nama: hello-world
- Wilayah: us-central1
- Memori yang Dialokasikan: 128 MB
- Waktu habis: 60 detik
- Pemicu: Pemicu HTTP
- Bucket: hello-world



Gambar 5.10. Google Cloud menyediakan pengaturan mudah untuk Cloud Functions

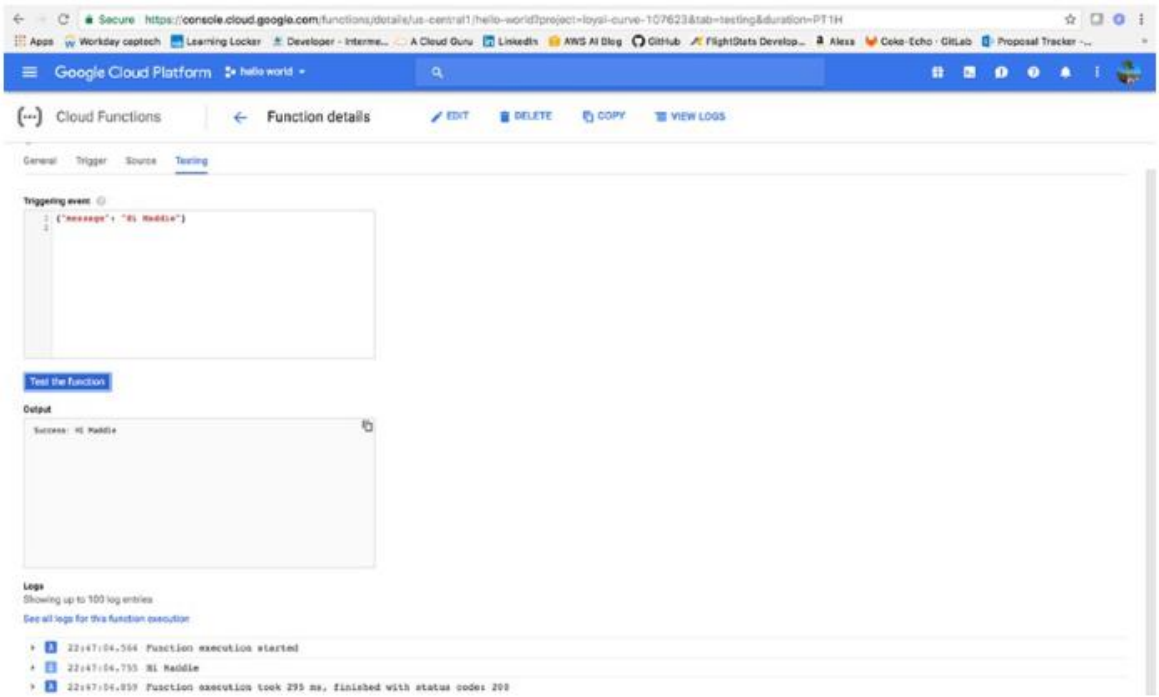
Kita akan memilih pemicu HTTP untuk menjadikannya fungsi dasar bagi fungsi pemicu HTTP kita. Google mirip dengan Azure karena menyediakan URL untuk mengakses dan

memicu fungsi HTTP Anda. Dengan ini, kita dapat menguji dan menjalankan fungsi kita tanpa harus membuat API secara keseluruhan. Konfigurasi untuk fungsi apa pun memerlukan nama, pemicu, wilayah, memori, batas waktu, kode sumber, dan bucket sumber. Nama tidak harus unik secara universal, cukup dalam fungsi Anda. Google Cloud juga memberi Anda pilihan untuk mengedit kode Anda secara inline atau mengunggah zip. Karena fungsi ini akan sederhana dan kecil, kita dapat mengeditnya secara inline.

Fungsi pemicu HTTP hello-world yang disediakan Google hanya mengambil isi permintaan yang masuk dan mengembalikannya ke konsol agar dapat dilihat pengguna. Fungsi tersebut menanggapi permintaan HTTP apa pun yang dapat menyediakan kolom "pesan" dalam isi. Untuk mengujinya dan merasa nyaman dengan lingkungan baru, kita akan tetap menggunakan fungsi HTTP hello-world yang disediakan oleh Google. Setelah dibuat, Anda akan melihatnya muncul di daftar fungsi. Jika Anda mengeklik fungsi tersebut, Anda akan memiliki akses ke pemantauan umum, konfigurasi pemicu, kode sumber, dan pengujian. Jika kita melihat pengujian, ada kotak masukan yang memungkinkan kita menentukan peristiwa pemicu. Ada juga kotak keluaran yang akan menunjukkan hasil dari log. Untuk memulai, mari buat peristiwa pengujian yang menyapa kita.

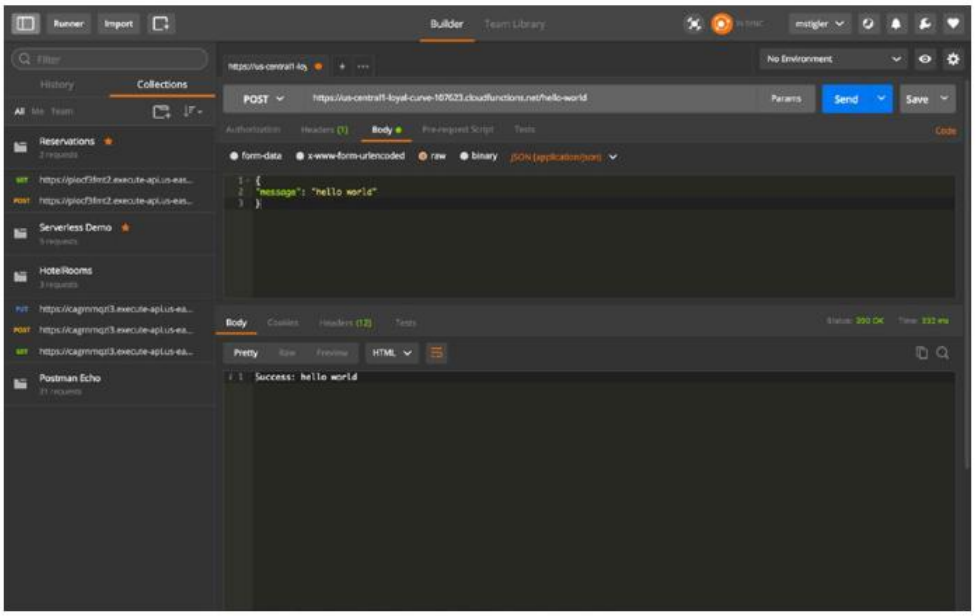
```
}
"message": "Hi Maddie"
}
```

Gambar 5-11 menunjukkan metode pengujian ini dalam pelaksanaan.



Gambar 5.11. Fungsi ini mengembalikan output yang sama dengan input dan mengirimkan log yang merinci pemicu

Fungsi tersebut harus memicu output yang sangat mirip dengan input dan juga harus memicu pencatatan log untuk fungsi tersebut. Jika Anda mengklik "Lihat semua log untuk eksekusi fungsi ini," Anda akan dibawa ke pencatatan log Stackdriver. Anda juga harus dapat mengaksesnya secara publik. Jika Anda mengambil URL untuk fungsi tersebut dan menyalinnya ke Postman dengan permintaan, Anda akan mendapatkan hasil yang sama (Gambar 5.12).



Gambar 5.12. Permintaan POST Postman ke fungsi Hello World kita

5.4 PENCATATAN STACKDRIVER

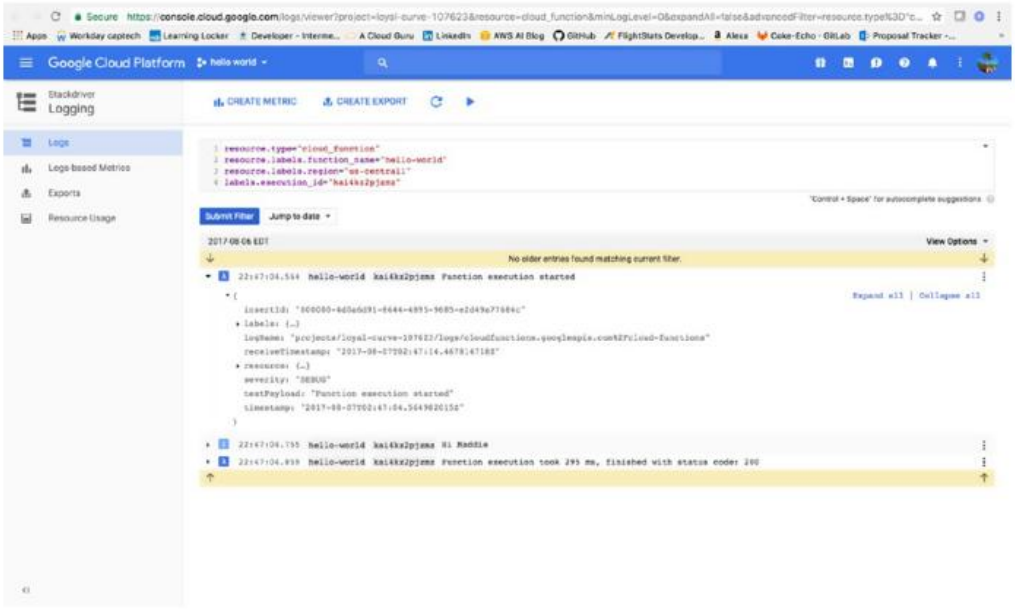
Pencatatan Stackdriver memungkinkan kita untuk menyimpan, mencari, menganalisis, memantau, dan memberi tahu tentang data log dan peristiwa dari Google Cloud Platform dan AWS. Saya sebenarnya belum menggunakan ini dengan AWS, tetapi integrasinya merupakan langkah penting. Banyak perusahaan takut beralih ke cloud karena ketergantungan pada vendor. Penyedia yang menerima ketakutan ini dan memberikan solusi untuknya mengambil langkah yang tepat dengan menjadi lebih integratif.

API Stackdriver menerima data log apa pun dari sumber mana pun. Ini hebat karena juga memungkinkan analisis data log ini secara real time. Pencatatan Stackdriver mencakup fitur-fitur berikut:

- API Log/Penyerapan Kustom: Pencatatan Stackdriver memiliki API publik yang dapat digunakan untuk menulis log kustom apa pun, dari sumber mana pun, ke dalam layanan.
- Integrasi / Agen AWS: Stackdriver Logging menggunakan agen Fluentd yang dikemas dan dikustomisasi Google yang dapat diinstal pada semua VM AWS atau Cloud Platform untuk menyerap data log dari instans Cloud Platform (Compute Engine, VM Terkelola, Kontainer) serta instans AWS EC2.
- Retensi Log: Memungkinkan Anda menyimpan log di Stackdriver Logging selama 30 hari, dan memberi Anda alat konfigurasi sekali klik untuk mengarsipkan data untuk periode yang lebih lama di Google Cloud Storage.

- Pencarian Log: Antarmuka yang canggih untuk mencari, mengiris, dan menelusuri data log.
- Metrik Berbasis Log: Stackdriver Logging memungkinkan pengguna membuat metrik dari data log yang muncul dengan lancar di Stackdriver Monitoring, tempat pengguna dapat memvisualisasikan metrik ini dan membuat dasbor.
- Pemberitahuan Log: Integrasi dengan Stackdriver Monitoring memungkinkan Anda mengatur pemberitahuan pada peristiwa log, termasuk metrik berbasis log yang telah Anda tetapkan.
- Analisis Lanjutan dengan BigQuery: Keluarkan data dengan konfigurasi sekali klik secara real time ke BigQuery untuk analisis lanjutan dan kueri seperti SQL.
- Arsipkan dengan Cloud Storage: Ekspor data log ke Google Cloud Storage untuk diarsipkan sehingga Anda dapat menyimpan data untuk periode waktu yang lebih lama dengan cara yang hemat biaya.
- Streaming Log dengan Cloud Pub/Sub: Streaming data logging Anda melalui Cloud Pub/Sub dengan solusi pihak ketiga atau titik akhir kustom pilihan Anda.
- Integrasi Splunk & Logentries - Stackdriver Logging mendukung integrasi mudah dengan Splunk dan Logentries (Rapid7).
- Audit Logging - Stackdriver Logs Viewer, API, dan gCloud CLI dapat digunakan untuk mengakses Audit Log yang merekam semua peristiwa akses data dan admin dalam Google Cloud Platform.

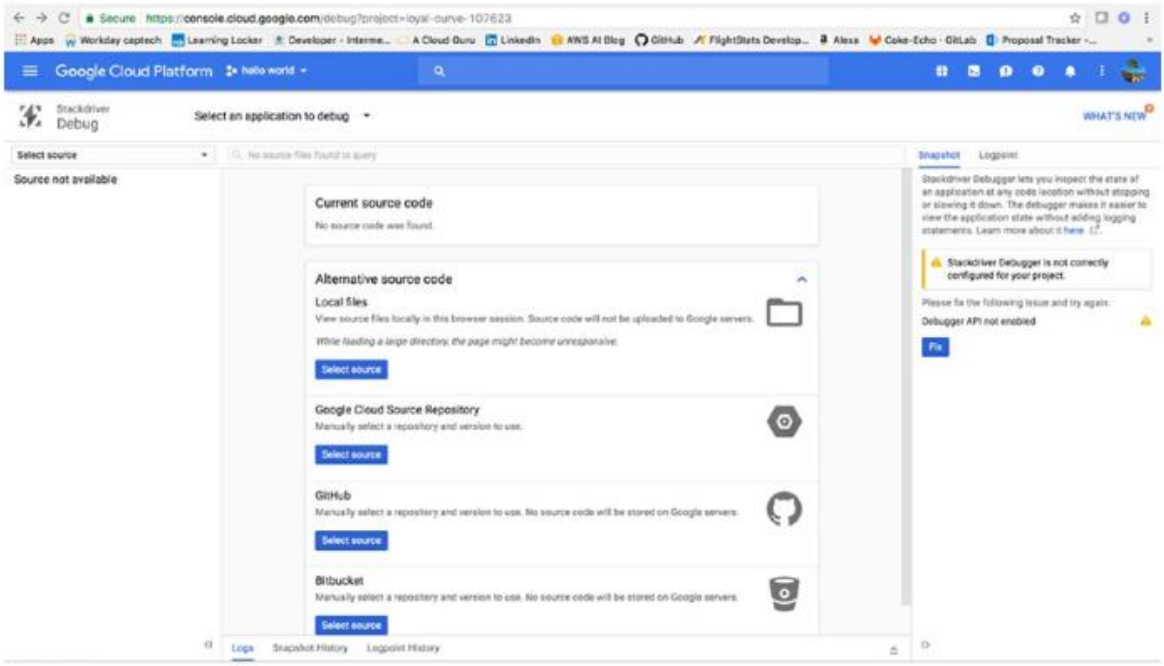
Stackdriver sangat mengingatkan saya pada CloudWatch di AWS. AWS menyediakan opsi untuk mengirim log CloudWatch ke Elasticsearch. Ini memberi Anda banyak fitur yang sama seperti yang kita lihat di Google Cloud Stackdriver. Ini adalah repositori pusat untuk semua logging. Jika Anda menavigasi ke sana setelah memicu proyek hello world kita, Anda akan melihat semua log untuk tindakan tersebut. Log tersebut diurutkan berdasarkan periode waktu dan memberi Anda permintaan yang masuk. Gambar 5.13 akan memberi Anda gambaran yang lebih baik tentang cara kerja logging Stackdriver.



Gambar 5.13. Stackdriver memberi Anda akses ke metrik, ekspor pencatatan, dan pencatatan pada suatu fungsi untuk tanggal tertentu

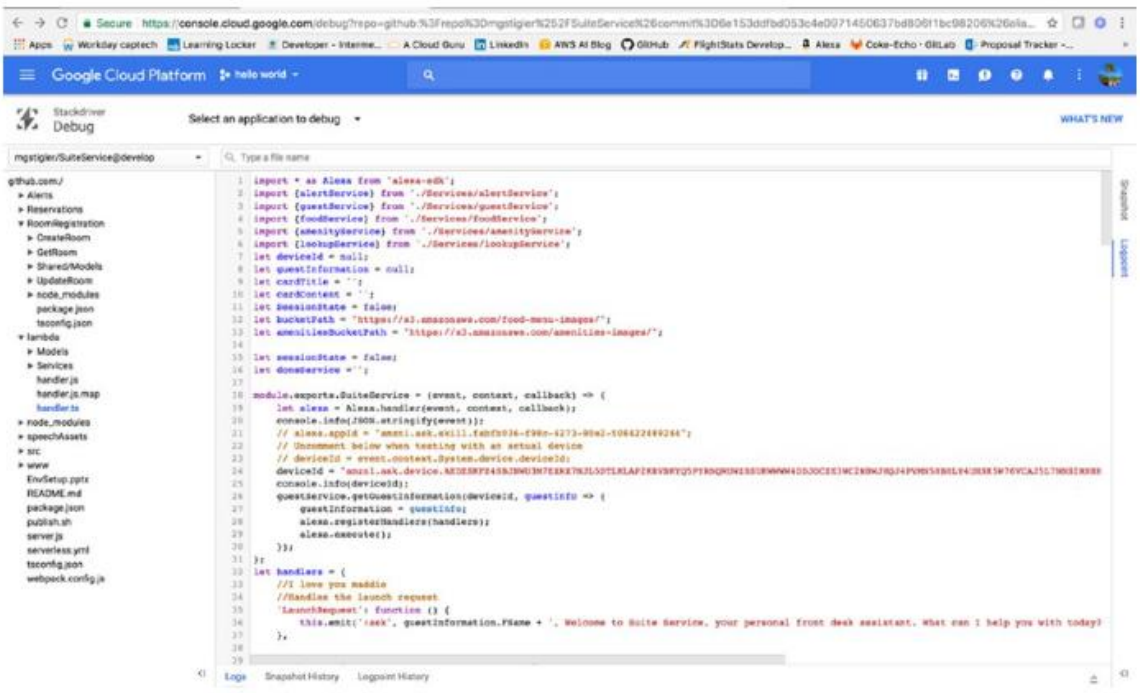
Stackdriver memberi Anda akses ke log, metrik, jejak, dan sinyal lain dari platform infrastruktur, mesin virtual, kontainer, middleware, dan tingkatan aplikasi Anda, sehingga Anda dapat melacak masalah dari pengguna akhir hingga layanan dan infrastruktur backend Anda. Dukungan asli untuk sistem terdistribusi, penskalaan otomatis, dan sumber daya sementara berarti bahwa pemantauan Anda bekerja dengan lancar dengan arsitektur modern Anda.

Jika Anda mengklik ikon hamburger Google Cloud, Anda akan melihat seluruh bagian yang dikhususkan untuk Stackdriver. Bagian ini mencakup Pemantauan, Debugging, Pencatatan, Jejak, dan pelaporan kesalahan. Kami sebagian besar akan menggunakan pencatatan, tetapi ada baiknya untuk mempelajari fungsionalitas lengkap Stackdriver. Stackdriver Debugger (Gambar 5.14) memungkinkan Anda memeriksa status aplikasi di lokasi kode mana pun tanpa menghentikan atau memperlambatnya. Debugger memudahkan untuk melihat status aplikasi tanpa menambahkan pernyataan pencatatan. Untuk memberikan contoh bagaimana kita dapat menggunakan debugger, saya akan masuk ke konsol dan mengaktifkan debugger untuk mengakses GitHub saya.



Gambar 5.14. Stackdriver memungkinkan Anda men-debug kode dan menyediakan beberapa opsi untuk kode sumber

GitHub akan meminta Anda untuk memberikan akses Google Cloud ke repo Anda. Saat Anda melakukannya, Anda dapat memilih sumber dari repo yang ingin Anda lihat.



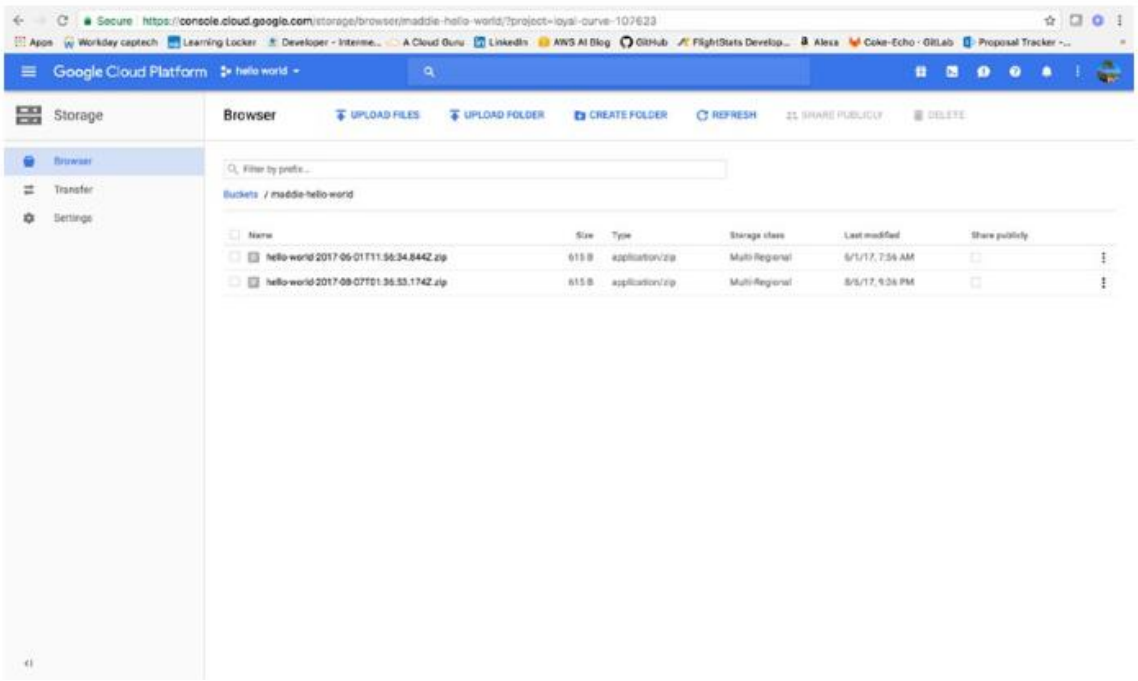
Gambar 5.15. Stackdriver memungkinkan Anda melihat kode produksi dalam mode debugging

Saya telah mengerjakan proyek Alexa Skills dengan beberapa rekan kerja, jadi saya akan membuka repo itu dan melihatnya. Jangan ragu untuk menjelajahi beberapa kode Anda sendiri. Pengguna Anda tidak terpengaruh selama debugging. Dengan menggunakan debugger produksi (Gambar 5.15), Anda dapat menangkap variabel lokal dan memanggil stack serta menautkannya kembali ke lokasi baris tertentu dalam kode sumber Anda. Anda dapat menggunakan ini untuk menganalisis status produksi aplikasi Anda dan memahami perilaku kode Anda dalam produksi.

5.5 STAGE BUCKET

Google Cloud menggunakan ide stage bucket untuk mementaskan dan menyebarkan kode Anda di Google Cloud. Kita akan melihat lebih jauh tentang bucket selama pembuatan fungsi Storage Trigger, tetapi akan berguna untuk melihat ikhtisar tingkat tinggi sekarang. Di Google, sumber daya Bucket mewakili bucket di Google Cloud Storage. Ada satu namespace global yang digunakan bersama oleh semua bucket. Ini berarti setiap nama bucket harus unik.

Bucket (Gambar 5.16) berisi objek, yang dapat diakses dengan metodenya sendiri. Selain properti ACL, bucket berisi bucketAccessControls, untuk digunakan dalam manipulasi terperinci dari kontrol akses bucket yang ada. Bucket selalu dimiliki oleh grup pemilik tim proyek.



Gambar 5.16. Bucket penyimpanan untuk Cloud Function saya

Di sinilah kode Cloud Function kami disimpan. Anda dapat mengaksesnya dari bucket. Anda juga dapat membuat, mengunggah file, dan mengunggah folder. Antarmuka ini sangat mirip dengan AWS S3 bagi saya. Sangat mudah digunakan, mudah dipahami, dan mudah diakses.

Sebelum melanjutkan ke fungsi pemicu peristiwa HTTP, kami akan menambahkan beberapa hal ke fungsi Hello World kami. Parameter permintaan yang harus kami pelajari meliputi:

- Request.method (misalnya, POST)
- Request.get('x-myheader') (misalnya, “123”)
- Request.query.foo (misalnya, “baz”)
- Request.body.text (misalnya, “something”)

Kami akan menguraikan fungsi Hello World kami sehingga dapat menangani berbagai jenis permintaan. Kami akan menggunakan parameter request.method untuk menentukan tindakan yang harus diambil dalam kasus tertentu. Kode dalam Daftar 5-1 menunjukkan cara menangani hal ini.

Daftar 5.1. Kode ini menunjukkan cara menangani beberapa permintaan HTTP dalam satu berkas.

```
function handlePUT(req, res) {
 //handle put request
 console.log(req.body.message);
 res.status(200).send('PUT Success: ' + 'Hello ${name || 'World'}!');
};

function handleGET(req, res) {
```

```
 //handle get request
 }
 console.log(req.body.message);
 res.status(200).send('GET Success:'+'Hello ${name||'World'}!');
};

/ **
* @param {Object} req Cloud Function request context.
* @param {Object} res Cloud Function response context.
*/
exports.helloWorld = function helloWorld (req, res) {
 let name =null;
 switch (req.method) {
  case 'GET':
   handleGET(req, res);
   break;
  case 'PUT':
   handlePUT(req, res);
   break;
  default:
   res.status(500).send({ error: 'Something blew up!' });
   break;
 }
};
```

Dalam kode ini, kami mengubah respons berdasarkan jenis permintaan. Anda seharusnya dapat melihat hasil ini di Postman. Anda dapat melihat bagaimana kami dapat memanfaatkannya nanti. Dalam arsitektur tanpa server, biasanya praktik yang lebih baik adalah memisahkan permintaan ini satu sama lain dalam kode. Untuk ini, Anda akan memiliki fungsi Cloud untuk setiap permintaan yang menangani satu eksekusi.
Kami masih dapat menggunakan properti request.method di sini dalam fungsi yang lebih tunggal untuk memeriksa ulang apakah permintaan yang masuk valid. Penting juga untuk mengetahui struktur permintaan yang masuk. Isi permintaan secara otomatis diurai berdasarkan jenis konten dan diisi dalam isi objek permintaan. Gambar 5-17 menunjukkan jenis permintaan yang diterima Google.

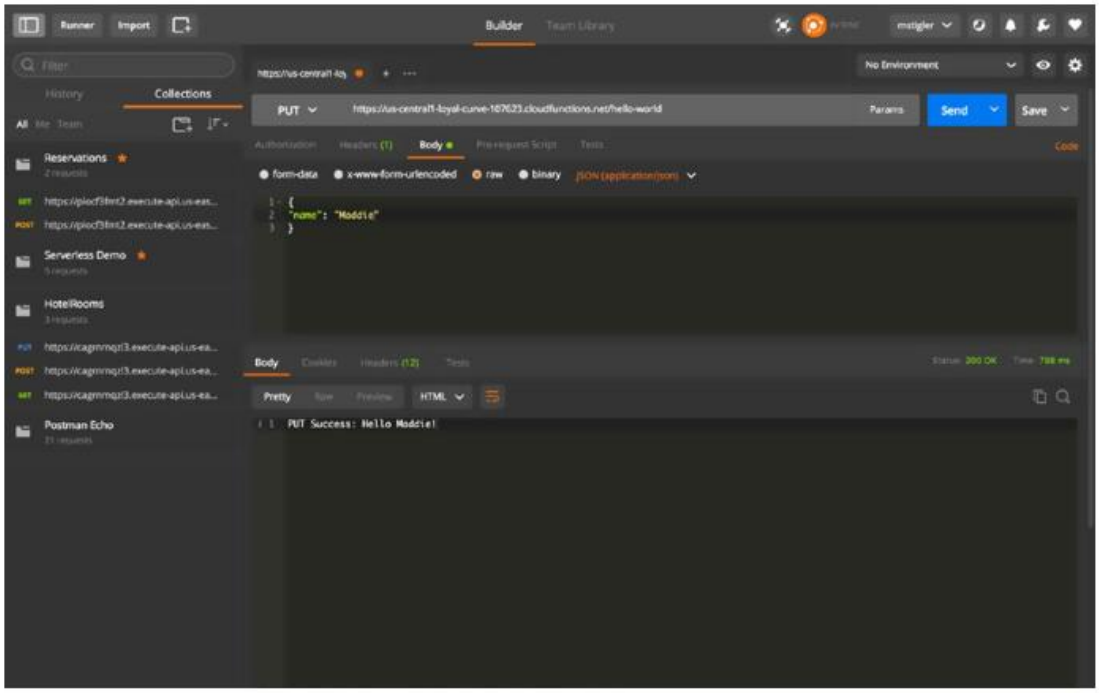
Content Type	Request Body	Behavior
`application/json`	`'{"name":"John"}'`	`request.body.name` equals 'John'
`application/octet-stream`	'my text'	`request.body` equals '6d792074657874' (see Node.js Buffer docs)
`text/plain`	'my text'	`request.body` equals 'my text'
`application/x-www-form-urlencoded`	'name=John'	`request.body.name` equals 'John'

Gambar 5.17. Contoh isi permintaan untuk berbagai jenis konten

Kita dapat menguji berbagai isi permintaan menggunakan pernyataan switch yang mirip dengan yang kita tulis untuk menentukan metode. Di dalam fungsi POST dan GET, kita akan menyertakan pernyataan switch yang mencari jenis konten dan mengembalikan respons yang sesuai. Kode berikut menunjukkan hal ini.

```
/**
 * @param {Object} req Cloud Function request context.
 * @param {Object} res Cloud Function response context.
*/
exports.helloWorld = function helloWorld (req, res) {
let name =null;
 switch (req.method) {
  case 'GET':
   handleGET(req, res);
   break;
  case 'PUT':
   handlePUT(req, res);
   break;
  default:
   res.status(500).send({ error: 'Something blew up!' });
   break;
  }
 };

 function handlePUT(req, res) {
 //handle put request
  switch (req.get('content-type')) {
  // '{"name":"Maddie"}'
   case 'application/json':
   name = req.body.name;
   break;

  // 'name=Maddie'
  case 'application/x-www-form-urlencoded':
   name = req.body.name;
   break;
  }
  console.log(req.body.message);
  res.status(200).send('PUT Success: ' + `Hello ${name || 'World'}!`);
  };

 function handleGET(req, res) {
  //handle get request
  switch (req.get('content-type')) {

   // '{"name":"Maddie"}'
   case 'application/json':
   name = req.body.name;
   break;

   // 'name=Maddie'
   case 'application/x-www-form-urlencoded':
    name = req.body.name;
    break;
```

```
  }
  console.log(req.body.message);
  res.status(200).send('GET Success: ' + `Hello ${name || 'World'}!`);
};
```



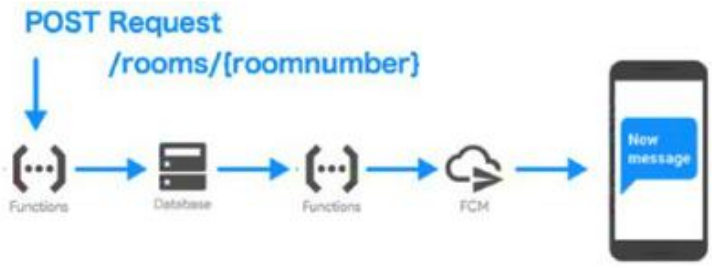
Gambar 5.18. Permintaan PUT Postman mengembalikan respons yang ditentukan di bawah fungsi PUT dengan pernyataan kasus alih aplikasi/JSON

Jika kita mengujinya di Postman menggunakan permintaan PUT dengan objek JSON, kita akan menerima respons yang ditentukan dalam handle PUT di bawah tipe konten application/json (Gambar 5.18). Sekarang setelah kita memiliki pemahaman dasar tentang penanganan permintaan dan respons dalam fungsi Google Cloud, kita dapat beralih ke pembuatan fungsi pemicu HTTP yang lebih lengkap.

5.6 PERISTIWA HTTP

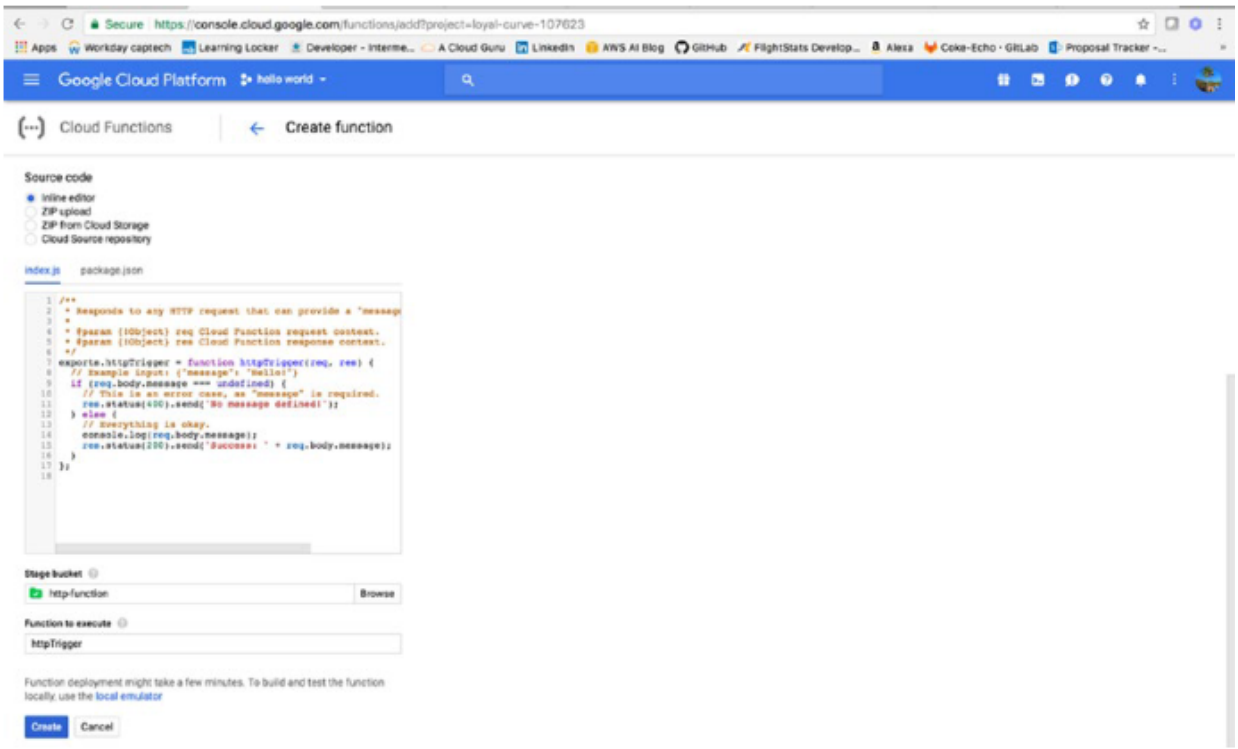
Fungsi pemicu HTTP kita akan menggunakan beberapa komponen Google Cloud untuk membuat sistem notifikasi. Untuk peristiwa HTTP, kita akan menggunakan permintaan POST dan basis data Firebase Realtime. Saat ini saya sedang mengerjakan proyek Alexa Skills yang membuat permintaan untuk tamu dari kamar hotel mereka. Saya akan mendasarkan fungsionalitas fungsi ini pada sistem tersebut.

Aplikasi kita akan menerima permintaan POST yang masuk dengan nomor kamar dan permintaan layanan, lalu mempostingnya ke basis data. Peristiwa HTTP kita akan menangani permintaan POST dengan informasi tamu baru dan akan menambahkan informasi tersebut ke keranjang penyimpanan tamu. Fungsi penyimpanan kemudian akan dipicu oleh unggahan objek ini dan akan terintegrasi dengan Firebase untuk memberi tahu tamu. Gambar 5-19 menunjukkan rencana untuk keseluruhan arsitektur dari beberapa latihan berikutnya.



Gambar 5.19. Permintaan POST akan mengambil nomor kamar dan menambahkan data tamu ke dalamnya dalam basis data

Untuk memulai, kita akan melanjutkan dan membuat fungsi pemicu HTTP lainnya. Saya memilih untuk membuat keranjang penyimpanan baru hanya untuk menyimpan semua layanan fungsi secara bersamaan. Saya juga mengubah fungsi yang akan dijalankan menjadi httpTrigger dan mengubahnya dalam kode contoh juga. Gambar 5.20 menunjukkan seperti apa konfigurasi fungsi baru saya nantinya.



Gambar 5.20. Konfigurasi untuk fungsi http

Sebelum kita melanjutkan pengembangan kerangka fungsi ini, kita akan menyiapkan lingkungan Firebase. Dalam aspek proyek ini, kita akan menggunakan Firebase untuk membuat basis data dan mengintegrasikannya dengan fungsi Cloud kita. Untuk menghubungkan fungsi ke Firebase Hosting, kita perlu menyiapkan Cloud Functions, menulis fungsi kita, membuat aturan penulisan ulang, dan menerapkan perubahan kita.

5.7 FIREBASE REALTIME DATABASE

Firebase Realtime Database adalah basis data yang dihosting di cloud. Data disimpan sebagai JSON dan disinkronkan dalam Realtime ke setiap klien yang terhubung. Ini sempurna untuk kasus penggunaan kita karena kita dapat menyimpan objek JSON tamu dan nomor kamar. Saat Anda membuat aplikasi lintas platform dengan iOS, Android, dan SDK JavaScript kita, semua klien Anda berbagi satu instans Realtime Database dan secara otomatis menerima pembaruan dengan data terbaru.

Beberapa aplikasi Google Firebase meliputi:

- Menyajikan konten dinamis: Anda dapat menjalankan logika sisi server melalui fungsi untuk mengembalikan respons yang dihasilkan secara dinamis.
- Pra-rendering untuk SPA guna meningkatkan SEO: Ini memungkinkan kita membuat tag meta dinamis untuk dibagikan di berbagai jejaring sosial.
- Aplikasi Web Ringan: Kita dapat membuat API dengan fungsi Cloud untuk situs hosting Firebase guna mengambil konten secara asinkron.

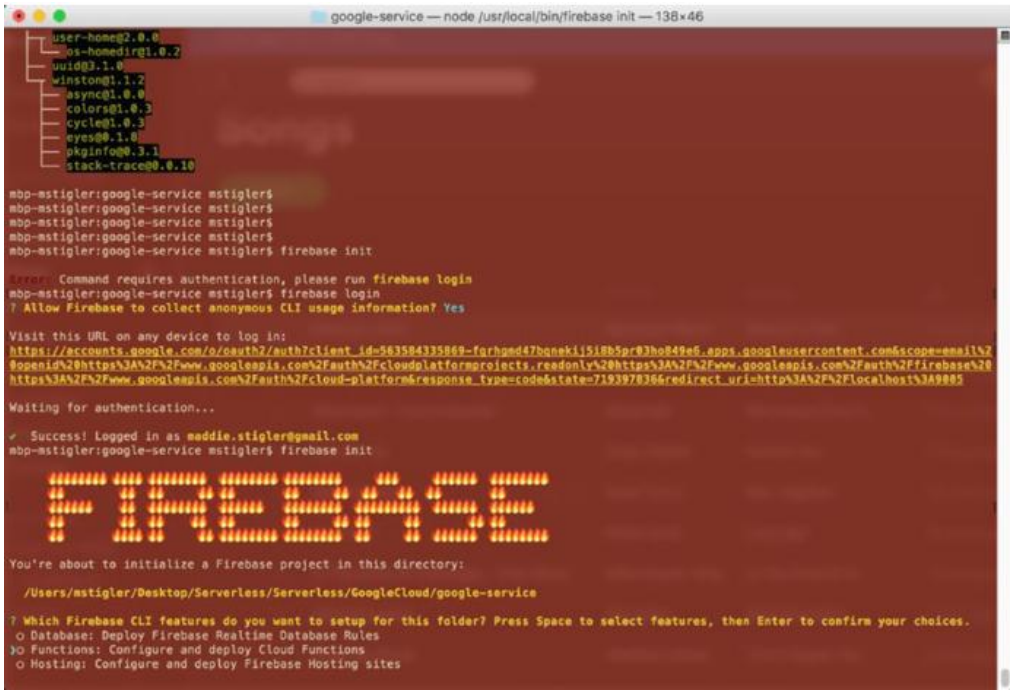
Salah satu aspek Firebase dan Google Cloud yang paling membingungkan bagi saya adalah perbedaan di antara keduanya. Keduanya cukup terintegrasi satu sama lain dan berbagi banyak sumber daya dan fungsi yang sama. Anda akan melihat ini saat menavigasi portal Firebase. Meskipun Firebase utamanya ditujukan untuk aplikasi seluler, Anda dapat menggunakan aspeknya yang dapat diskalakan dan bebas infrastruktur untuk menyiapkan dan menyebarkan aplikasi tanpa server dengan cepat. Nantinya, Anda dapat mengintegrasikan aplikasi ini dengan platform seluler, atau tetap menjalankannya sebagaimana adanya.

Saat pertama kali memulai dengan Firebase, Anda akan diminta untuk membuat proyek atau mengimpornya dari Google Cloud. Untuk aplikasi kita, kita akan membangun di dalam Firebase untuk mendapatkan paparan terhadap lingkungan ini. Namun, yang saya sarankan untuk dilakukan setelah latihan ini adalah membangun semua fungsi Anda di dalam proyek di Google Cloud dan mengimpor seluruh proyek ke Firebase. Untuk membuat aplikasi yang menggunakan Firebase, kita harus mengikuti beberapa langkah untuk menyiapkan fungsi kita.

Pertama, pastikan Anda memiliki Firebase CLI versi terbaru. Anda dapat memasang Firebase CLI versi terbaru dengan menjalankan perintah berikut di terminal Anda:

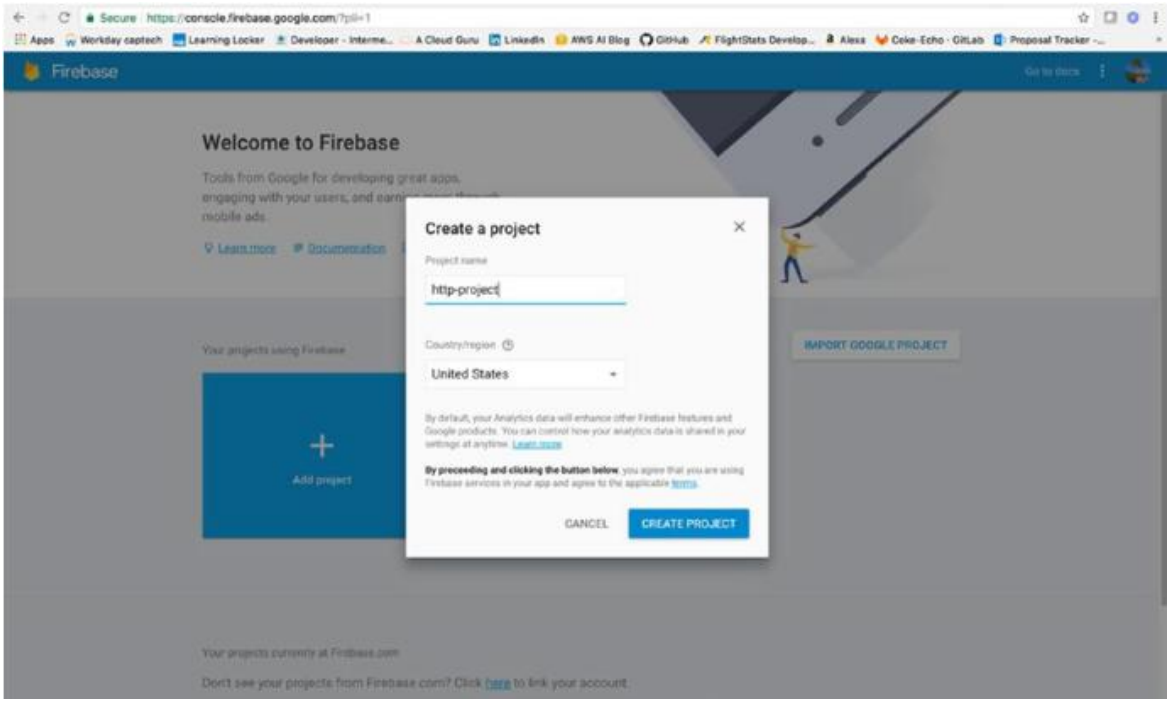
```
npm install –g firebase-tools
```

Ini akan memasang Firebase toolkit di terminal Anda. Kita akan menggunakan ini untuk mengakses proyek Firebase dan menyebarkannya menggunakan fungsi Cloud. Dalam hal ini, ia memiliki fungsi yang mirip dengan kerangka kerja tanpa server. Kita akan dapat menyebarkan proyek kita sepenuhnya menggunakan alat Firebase alih-alih harus mengompres dan mengunggah folder proyek. Setelah Anda memasang alat Firebase menggunakan NPM, terminal Anda akan terlihat seperti Gambar 5-21 jika Firebase berhasil dipasang.



Gambar 5.21. Firebase akan menginstal dan memberi Anda berbagai opsi penerapan proyek

Firebase memberi Anda URL untuk masuk ke akun Google Cloud Anda. Ini akan memungkinkan Anda untuk menerapkan langsung ke lingkungan cloud Anda. Setelah Anda diberi otorisasi, Anda dapat mulai menginisialisasi proyek Firebase di direktori saat ini. Namun, sebelum melakukannya, Anda perlu kembali ke konsol Firebase dan membuat proyek yang sudah ada dengan Hosting. Untuk melakukannya, navigasikan ke console.firebase.google.com dan klik Buat Proyek Baru. Saya menamai proyek saya http-project (Gambar 5.22).



Gambar 5.22. Buat proyek baru di Firebase untuk menghubungkan fungsi kita ke hosting Firebase

Setelah membuat proyek, kita dapat menginisialisasinya kembali di terminal kita dengan perintah:

```
firebase use –add
```

Ini akan meminta Anda untuk memilih proyek dan membuat alias untuknya. Lanjutkan dan pilih proyek yang Anda buat. Saya memilih staging sebagai alias saya, tetapi Anda bebas memberi nama proyek Anda sesuka Anda. Setelah ini, kita memiliki proyek yang sudah ada dengan hosting dan dapat menjalankan perintah berikut dalam direktori proyek kita untuk menginisialisasi fungsi cloud:

```
firebase init function
```

Perintah ini melakukan beberapa hal. Pertama, perintah ini membuat direktori fungsi dengan file index.js.

Ini adalah file fungsi cloud Anda yang akan kita edit. Kedua, perintah ini menambahkan file .firebaserc ke direktori proyek Anda. File ini adalah tempat definisi alias Anda ditulis. Kita tidak perlu mengubahnya, tetapi silakan lihat dan pahami bagaimana alias didefinisikan. Milik saya terlihat seperti berikut:

```
{
 "projects": {
  "staging": "http-project-6da2a",
  "default": "http-project-6da2a"
 }
}
```

Terakhir, perintah init membuat file firebase.json. File ini adalah file konfigurasi untuk direktori kita saat ini. Kita akan menggunakan file ini untuk menangani penulisan ulang dalam fungsi kita. Penulisan ulang sebenarnya tidak diperlukan untuk aplikasi kita, tetapi kita akan membahas contoh penggunaannya agar lebih mudah dipahami. Dengan aturan penulisan ulang, kita dapat mengarahkan permintaan yang sesuai dengan pola tertentu ke satu tujuan.

Contohnya adalah mengeksekusi fungsi tertentu saat halaman tertentu dibuka. Dalam aplikasi kita, kita dapat memikirkan beberapa situasi yang mungkin berguna. Kita dapat menulis ulang untuk halaman /guests yang memicu fungsi guest yang menampilkan daftar tamu yang saat ini menginap di hotel. Kita juga dapat memiliki halaman /alerts yang menampilkan semua peringatan yang telah dikirim tamu. Ingatlah hal ini saat Anda membangun fungsi Anda nanti.

Untuk saat ini, kita akan mengedit file index.js kita untuk menampilkan permintaan HTTP kita di browser. Kode berikut menunjukkan contoh cara melakukannya. Anda perlu melakukan instalasi NPM untuk mendapatkan modul cors.

```
'use strict';
const functions = require('firebase-functions');
const cors = require('cors')({origin: true});

exports.alerts = functions.https.onRequest((req, res) => {

 cors(req, res, () => {
  let request = req.query.alert;
  let roomnumber = req.query.roomNumber;
  console.log("alert " + alert + " room " + roomnumber);
  if (!request) {
   request = req.body.alert;
   roomnumber = req.body.roomNumber;
 }
 res.status(200).send(`<!doctype html>
 <head>
  <title>Incoming Request</title>
 </head>
 <body>
  Request: ${alert}
  </br>
  RoomNumber: ${roomnumber}
 </body>
 </html>`);
 });
 });
```

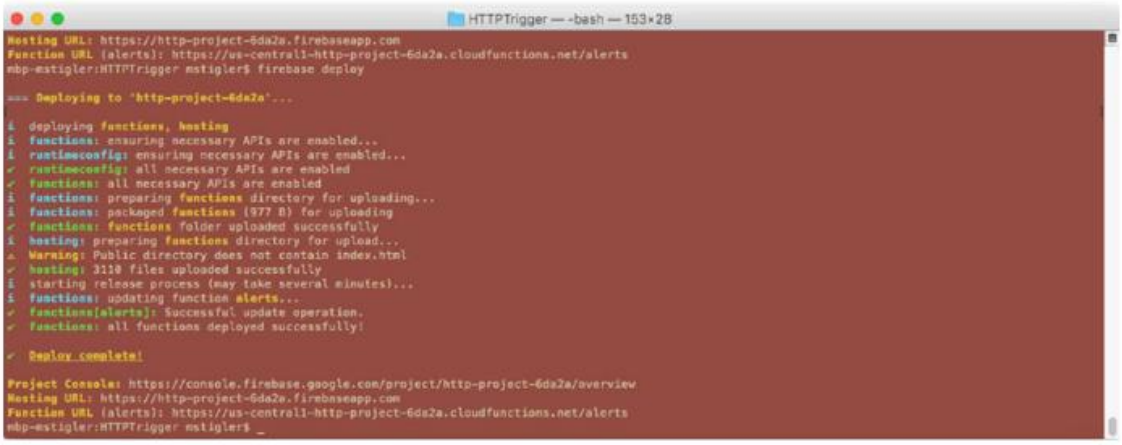
Kode tersebut mengambil permintaan yang masuk dan memisahkannya berdasarkan kuerinya. Kemudian, kode tersebut mengembalikan permintaan tersebut kepada pengguna dalam format HTML. Kita juga ingin mengedit file firebase.json untuk mengarahkan ulang permintaan berdasarkan metode. Kode berikut adalah cara yang saya pilih untuk mengarahkan ulang fungsi saya:

```
 {
  "hosting": {
   "public": "functions",
   "rewrites": [
    {
     "source": "/alerts",
     "function": "alerts"
    }
   ]
  }
 }
```

Ini menentukan folder tempat proyek kita mengambil data, sumber yang kita inginkan untuk mengarahkan permintaan kita, dan nama fungsi yang ingin kita jalankan. Untuk menyebarkan fungsi ini, kita akan menggunakan perintah firebase deploy di direktori root kita:

```
firebase deploy
```

Dalam gambar 5.23 menunjukkan hasilnya.



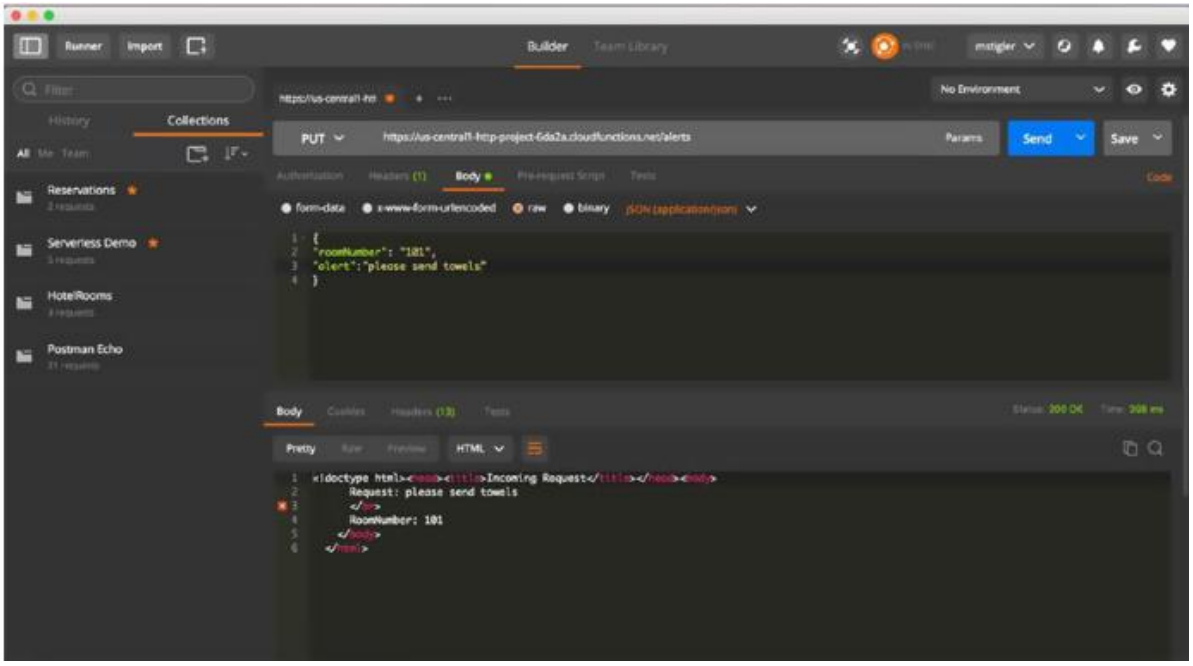
Gambar 5.23 Hasil

Setelah kode Anda berhasil diterapkan, kita dapat menguji fungsionalitasnya dengan cukup mengetik URL fungsi tersebut di browser kita. Untuk melakukannya, masukkan URL berikut dengan permintaan Anda:

```
https://us-central1-<project
name>.cloudfunctions.net/alerts?alert=<your
alert>&roomnumber=<number>
```

Ini akan mengirimkan permintaan ke fungsi yang Anda buat dan mengembalikan respons yang Anda tentukan (Gambar 5.24).

Dalam gambar 5.24. Permintaan diformat dan ditampilkan di browser Anda



Gambar 5.25. Postman menerima permintaan yang sama dan akan menerima format isi JSON

Tentu saja, ini bukan cara yang paling menarik untuk menampilkan informasi kita di browser, tetapi cara ini tetap memberi Anda gambaran tentang bagaimana Cloud Functions terintegrasi dengan Firebase dan berbagai aplikasi untuk integrasi ini. Anda juga dapat mengakses permintaan ini di Postman (Gambar 5-25). Anda dapat mengambil string kueri dan menjadikannya bagian dari isi permintaan untuk menguji berbagai cara mengakses informasi kita.

Dalam latihan ini, kami telah menggunakan Cloud Functions dengan Firebase untuk mengembangkan aplikasi yang menerima permintaan POST yang masuk, menyajikannya di browser, dan mengirimkannya ke database Firebase Realtime. Sekarang setelah kami mengetahui fungsi kami dipicu, mari lakukan sesuatu yang nyata dengan informasi yang masuk. Kami akan membangun fungsi tanpa server kami saat ini untuk memungkinkan data yang masuk disimpan dalam database Firebase Realtime. Untuk melakukannya, kami akan mengedit fungsi kami untuk terhubung ke database yang kami beri nama dan memasukkan setiap entri yang masuk ke dalam database. Kode berikut menunjukkan konsep ini dalam tindakan.

```
'use strict';
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);
const cors = require('cors')({origin: true});
const storageModule = require('./storage');

exports.alerts = functions.https.onRequest((req, res) => {

  cors(req, res, () => {
   let alert = req.query.alert;
```

```
    let roomnumber = req.query.roomNumber;
    let phoneNumber = req.query.phoneNumber;
    let name = req.query.name;
    console.log("alert " + alert + " room " + roomnumber);
    if (!alert) {
     alert = req.body.alert;
     roomnumber = req.body.roomNumber;
     phoneNumber = req.body.phoneNumber;
     name = req.body.name;
    }
admin.database().ref('/alerts').push({alert: alert,
roomnumber:roomnumber, phoneNumber:
phoneNumber, name: name}).then(snapshot => {
});
res.status(200).send(`<!doctype html>
<head>
 <title>Incoming Request</title>
</head>
<body>
 Request: ${alert}
</br>
 RoomNumber: ${roomnumber}
</body>
</html>`);
});
});

exports.sendAlert =
functions.database.ref('/alerts/{pushId}').onWrite(storageModule.
handler);
```

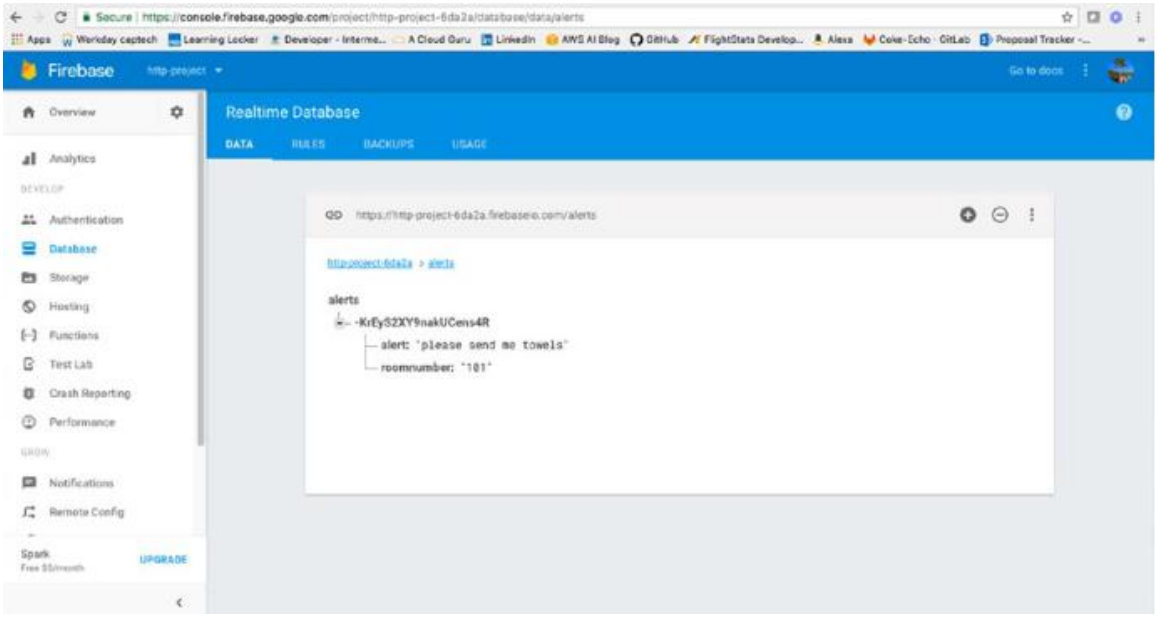
Basis data Realtime adalah basis data NoSQL dan karenanya memiliki pengoptimalan dan fungsionalitas yang berbeda dari basis data relasional. API Basis Data Realtime dirancang untuk hanya mengizinkan operasi yang dapat dieksekusi dengan cepat. Hal ini memungkinkan kami membangun pengalaman real-time yang hebat yang dapat melayani jutaan pengguna tanpa mengorbankan responsivitas.

Seperti yang dapat Anda lihat dalam kode ini, pertama-tama kami memeriksa parameter permintaan yang masuk untuk melihat apakah permintaan tersebut masuk melalui string kueri atau badan JSON. Kemudian kami mengurai permintaan dan memasukkannya ke jalur Peringatan di basis data Realtime untuk proyek kami. Setelah Anda menyempurnakan kode ini, kami dapat melanjutkan dan menerapkannya kembali ke aplikasi kami. Kami akan menggunakan perintah firebase deploy yang sama.

Saat kode diterapkan, kami dapat menguji perubahan kami dengan menekan URL yang sama di browser. Kita akan menggunakan string kueri peringatan dan nomor ruangan yang sama untuk melakukan ping pada fungsi tersebut. Kita dapat memeriksa keberhasilan fungsi kita dengan beberapa cara.

Jika kita kembali ke portal Firebase, di sisi kiri Anda akan melihat opsi Database. Dengan mengeklik opsi ini, Anda akan dibawa ke konsol Database dengan data real-time.

Anda akan melihat tabel data terstruktur di bawah nama proyek Anda. Di bawah nama proyek, Anda akan melihat jalur yang telah Anda buat dan data yang terkait dengan masing-masing jalur. Panel Database saya tampak seperti Gambar 5.26.



Gambar 5.26. Postman menerima permintaan yang sama dan akan menerima format isi JSON

Peringatan dan nomor ruangan dikaitkan dengan kunci tertentu yang dibuat berdasarkan permintaan. Anda dapat menambahkan properti sebanyak yang Anda inginkan dan properti tersebut akan berubah secara dinamis sesuai permintaan. Data basis data Firebase Realtime disimpan sebagai objek JSON. Google ingin menggambarkan struktur tersebut sebagai pohon JSON yang dihosting di cloud. Tidak seperti SQL, tidak ada tabel atau rekaman. Semua data yang ditambahkan hanya menjadi simpul baru dalam struktur JSON yang ada dengan kunci terkait.

Anda memiliki kemampuan untuk memberikan kunci Anda sendiri dalam permintaan POST. Google juga merekomendasikan beberapa praktik terbaik untuk menyusun data Anda, yang akan kita bahas selanjutnya.

Hindari Penumpukan Data

Karena Firebase Realtime Database memungkinkan penumpukan data hingga 32 level, Anda mungkin tergoda untuk berpikir bahwa ini seharusnya menjadi struktur default. Namun, saat Anda mengambil data di suatu lokasi di database, Anda juga mengambil semua node turunannya. Selain itu, saat Anda memberi seseorang akses baca atau tulis di suatu node di database, Anda juga memberi mereka akses ke semua data di bawah node tersebut. Oleh karena itu, dalam praktiknya, sebaiknya pertahankan struktur data Anda sedatar mungkin.

Meratakan Struktur Data

Jika data dibagi menjadi jalur terpisah, yang juga disebut denormalisasi, data dapat diunduh secara efisien dalam panggilan terpisah, sesuai kebutuhan.

Membuat Data yang Dapat Diskalakan

Saat membuat aplikasi, sering kali lebih baik mengunduh sebagian dari daftar. Hal ini sangat umum terjadi jika daftar tersebut berisi ribuan rekaman. Jika hubungan ini statis dan satu arah, Anda cukup menumpuk objek anak di bawah induk. Terkadang, hubungan ini lebih dinamis, atau mungkin perlu untuk mendenormalisasi data ini. Sering kali Anda dapat mendenormalisasi data dengan menggunakan kueri untuk mengambil sebagian dari data. Namun, ini pun mungkin tidak cukup. Pertimbangkan, misalnya, hubungan dua arah antara pengguna dan grup. Pengguna dapat menjadi bagian dari grup, dan grup tersebut terdiri dari daftar pengguna. Ketika tiba saatnya untuk memutuskan grup mana yang menjadi tempat pengguna, semuanya menjadi rumit.

5.7 MENINGKATKAN FUNGSI TANPA SERVER

Peningkatan dengan menerapkan variabel bertipe kuat dan otorisasi

Buat variabel bertipe kuat:

1. Gunakan TypeScript dan buat model untuk permintaan masuk kami. Ini akan menerapkan struktur dalam kode dan permintaan kami. Contoh:

```
export interface alertModel {
 alert: string,
 roomNumber: string
}
```

2. Pastikan permintaan yang masuk memenuhi model ini untuk memastikan bahwa kami tidak mendapatkan data acak dalam permintaan kami atau data yang tidak valid dalam permintaan kami.
3. Periksa metode permintaan sebelum menangani permintaan. Kami hanya ingin melihat permintaan POST dan PUT dalam kasus ini. Permintaan GET akan ditangani secara berbeda. Manfaatkan pernyataan switch yang kami buat di Hello World untuk mengimplementasikan ini.

Tambahkan bagian otorisasi:

1. Saat ini, fungsi kami terbuka lebar untuk semua permintaan yang masuk melalui titik akhir HTTP kami. Dengan aplikasi produksi, kami tidak ingin hal itu terjadi. Ada beberapa cara untuk menangani hal ini, salah satunya adalah membuat header otorisasi.
2. Gunakan token Firebase Id bersama dengan permintaan di header Authorization untuk memverifikasi permintaan menggunakan Firebase Admin SDK. Ada contoh bagus dari implementasi ini di GitHub di https://github.com/firebase/functions-samples/blob/master/authorized-https-endpoint/functions/index.js

Kode untuk kedua peningkatan pada proyek ini dapat ditemukan di sini:
https://github.com/mgstigler/Serverless/tree/master/AWS/aws-service/HTTPTrigger

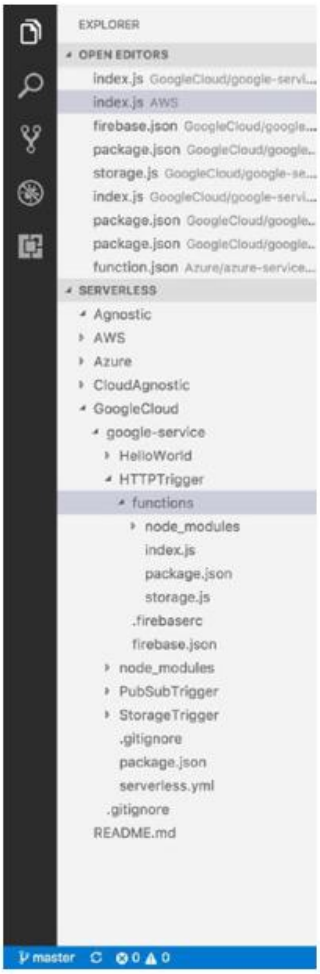
Di bagian berikutnya, kita akan menggunakan keterampilan dan alat yang kita pelajari dengan HTTP Trigger untuk membuat fungsi Google Cloud terpisah yang dipicu oleh peristiwa penyimpanan di Firebase.

Peristiwa Penyimpanan

Di bagian ini, kita akan terus mengembangkan aplikasi notifikasi untuk tamu hotel dengan menggabungkan peristiwa pemicu penyimpanan yang membuat fungsi penyimpanan kita bertindak. Kita akan menggunakan data POST dari latihan sebelumnya sebagai pemicu penyimpanan ini. Setelah fungsi kita dipicu, kita akan menggunakan Twilio lagi untuk mengirim pesan ke pelanggan. Sebelum kita mulai, kita perlu membuat beberapa perubahan pada fungsi HTTP kita saat ini. Kita ingin dapat menyimpan nama dan nomor telepon tamu selain permintaan dan nomor kamar mereka. Untuk melakukannya, kembali ke fungsi Anda dan pastikan kolom-kolom ini berjenis yang dapat dikirimkan baik melalui string kueri maupun melalui isi posting JSON. Setelah selesai, kita dapat mulai membuat fungsi kita.

Buat Fungsi yang Dipicu Penyimpanan Kita

Seperti yang saya sebutkan sebelumnya, kita akan menerapkan aplikasi ini melalui Firebase untuk mempelajari sesuatu yang baru dan mendemonstrasikan pendekatan yang berbeda. Sayangnya, kita akan mengalami salah satu kendala dari metode penerapan ini, yaitu beberapa fungsi per proyek.



Gambar 5.27. Kita akan membuat file JavaScript lain dalam struktur proyek kita saat ini untuk menunjukkan bagaimana Firebase membagi fungsi dalam lingkungannya

Agar dapat menangani perubahan ini, kita akan sedikit merestrukturisasi kode proyek kita. Gambar 5-27 menunjukkan folder fungsi dengan file JavaScript indeks kita dan file storage.js tambahan. Kita akan menempatkan fungsi kedua kita langsung di folder ini dan akan menangani permintaan terpisah dalam file index.js.

Alasan kita perlu menyertakan file ini langsung dalam folder fungsi kita dalam peristiwa pemicu HTTP adalah karena kita memerlukan akses ke basis data peristiwa HTTP Realtime. Kita tidak memiliki akses ke basis data antarproyek, jadi kita akan menerapkan kedua fungsi tersebut bersama-sama.

Nanti, kita akan melihat contoh bagaimana kita dapat menerapkan semua fungsi kita melalui lingkungan Google Cloud dan tetap menampilkannya di Firebase sebagai entitas terpisah. Untuk dapat merujuk kode yang akan kita gunakan dalam file storage.js, kita perlu menyertakannya dalam file index.js.

Kode berikut menunjukkan bagaimana saya menerapkannya.

```
'use strict'
const functions = require('firebase-functions');
const admin = require('firebase-admin');
```

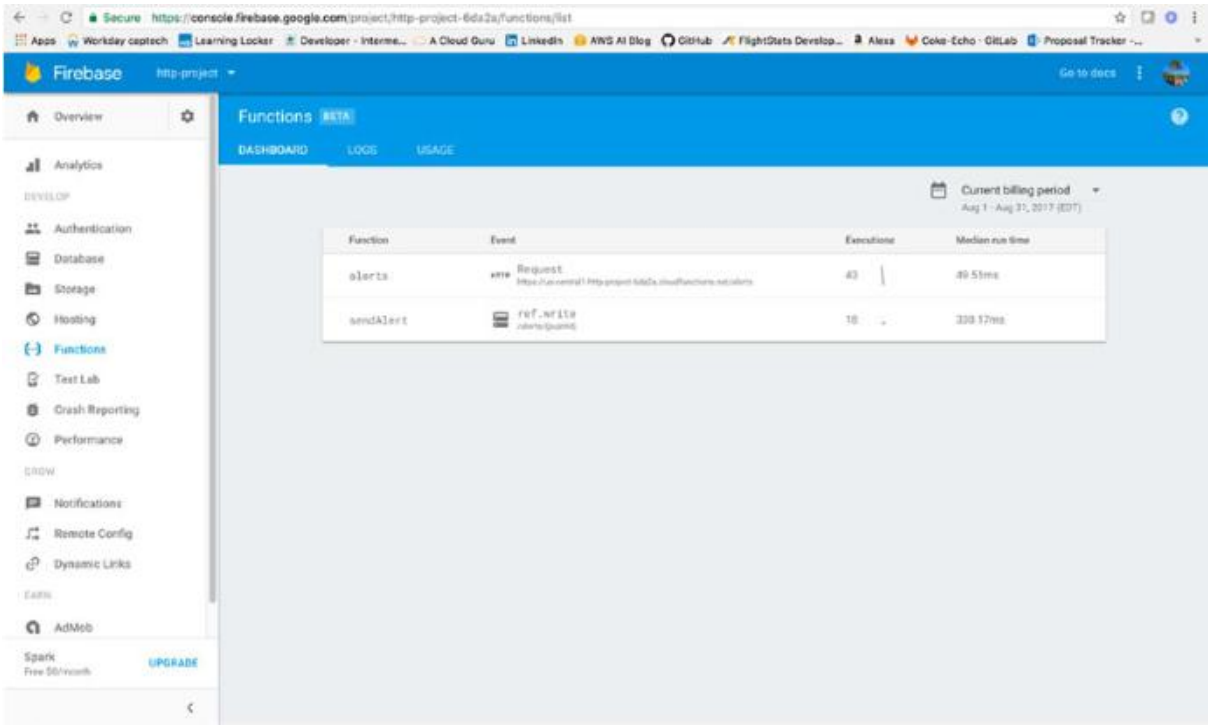
```
admin.initializeApp(functions.config().firebase);
const cors = require('cors')({ origin : true });
const storageModule = require('./storage');
```

Pertama-tama kita memerlukan file storage.js untuk digunakan dalam file indeks kita.

```
exports.sendAlert =
functions.database.ref('/alerts/{pushId}').onWrite(storageModule.ha
ndler);
```

Kemudian kita dapat menggunakan variabel ini untuk memanggil pengendali dalam file storage.js kita setiap kali item baru ditulis ke database Realtime kita. Anda dapat mengujinya dengan hanya meletakkan console.log di file penyimpanan Anda dan melihat apakah ia dipicu ketika file indeks dipicu.

Kita akan melanjutkan dan menerapkan fungsi ini untuk memastikan Firebase mengenali dua fungsi Cloud yang terpisah. Di dalam konsol Firebase, kita dapat menavigasi ke Functions dan melihat fungsi yang aktif (Gambar 5-28). Jika diterapkan dengan benar, Anda akan melihat dua fungsi terpisah: alerts dan sendAlert. Anda juga akan melihat peristiwa pemicu yang berbeda untuk masing-masing. Anda akan melihat peristiwa Write untuk fungsi sendAlert.



Gambar 5.28. Firebase memisahkan fungsi berdasarkan peristiwa pemicu. Anda akan melihat log untuk keduanya di konsol yang sama.

Kita juga harus memastikan pemicu terjadi pada penulisan yang tepat. Kita ingin dapat melacak semua penulisan ke tabel peringatan. Untuk melakukan ini, kita perlu merujuk hanya ke peringatan/{pushId}. Jika kita hanya ingin melihat penulisan yang secara khusus terkait

dengan nomor kamar, kita dapat melakukannya dengan merujuk ke peringatan/{pushId}/ nomorkamar. Ini adalah sesuatu yang baik untuk diingat untuk iterasi mendatang. Anda dapat memiliki beberapa fungsi yang semuanya membaca kunci yang berbeda di pohon JSON peringatan.

Bereaksi terhadap Peristiwa yang Dipicu

Kami ingin terus mengembangkan aplikasi kami dengan membuat fungsionalitas fungsi pemicu penyimpanan. Saat ini, kami memiliki dua kebutuhan untuk fungsi ini:

1. Kirim teks ke tamu untuk memberi tahu mereka bahwa permintaan mereka telah diterima dan layanan sedang dalam perjalanan.
2. Perbarui basis data untuk mencerminkan perubahan ini dan beri tahu fungsi bahwa peringatan telah dikirim.

Untuk menangani persyaratan pesan teks, kami akan kembali ke akun Twilio kami dan menggunakan layanan ini untuk pengiriman pesan teks. Jika Anda tidak menyiapkan akun Twilio untuk bab Azure, silakan kembali dan urus itu sekarang. Kami akan menggunakan variabel lingkungan Firebase untuk menyimpan token autentikasi dan ID akun. Firebase SDK untuk Cloud Functions menawarkan konfigurasi lingkungan bawaan untuk memudahkan penyimpanan dan pengambilan data yang aman untuk proyek kami tanpa harus melakukan penyebaran ulang.

Untuk menyimpan data Twilio kami, kami dapat menggunakan perintah Firebase berikut di Command Line kami:

```
firebase functions:config:set twilioservice.authtoken=
"XXXXXXXXXXXXXX" twilioservice.
accountsid="XXXXXXXXXXXX"
```

Untuk memeriksa apakah informasi kita disimpan dengan benar, kita dapat mengakses variabel lingkungan dengan menggunakan perintah Firebase berikut:

```
firebase functions:config:get
```

Berikut ini mendemonstrasikan seperti apa seharusnya proses ini di terminal Anda:

```
mbp-mstigler:HTTPTrigger mstigler$ firebase functions:config:set
twilioservice.authtoken="XXXXXXXXXXXX"    twilioservice.accountsid="
XXXXXXXXXXXX "
✓ Functions config updated.
```

Harap terapkan fungsi Anda agar perubahan berlaku dengan menjalankan **`firebase deploy --only function`**

```
mbp-mstigler:HTTPTrigger mstigler$ firebase functions:config:get
{
 "twilioservice": {
  "accountsid": " XXXXXXXXXXXX ",
```

```
  "authtoken": " XXXXXXXXXXXX "
 }
}
```

Sekarang kita telah menyiapkan variabel lingkungan dan dapat menggunakannya di seluruh fungsi penyimpanan. Kode berikut menetapkan klien Twilio dengan dua bagian autentikasi, menyimpan string kueri yang masuk sebagai variabel lokal, membuat pesan menggunakan variabel ini, dan mengirimkannya ke penerima.

```
'use strict'

const functions = require('firebase-functions');
const admin = require('firebase-admin');

exports.handler = (event) => {
 const accountSid = functions.config().twilioservice.accountsid;
 const authToken = functions.config().twilioservice.authtoken;
 const client = require('twilio')(accountSid, authToken);
 var alert = event.data.val();
 console.log("Alert " + JSON.stringify(alert));
 var number = alert.phoneNumber;
 var name = alert.name;
 var room = alert.roomnumber;
 client.messages.create({
   from: '+18178544390',
   to: number,
   body: "Hello " + name + "! Your request is on the way to room " +
   room +"."
}, function(err, message) {
  if(err) {
   console.error(err.message);
  }
});

return event.data.ref.parent.child('alertSent').set("alert has been
sent");

};
```

Kita dapat menguji kode ini dengan menerapkannya dan memasukkan string kueri dengan empat variabel yang diperlukan. Jika semuanya berjalan lancar, Anda akan menerima pesan teks dari akun uji coba Twilio Anda dengan isi pesan yang Anda berikan (Gambar 5-29).

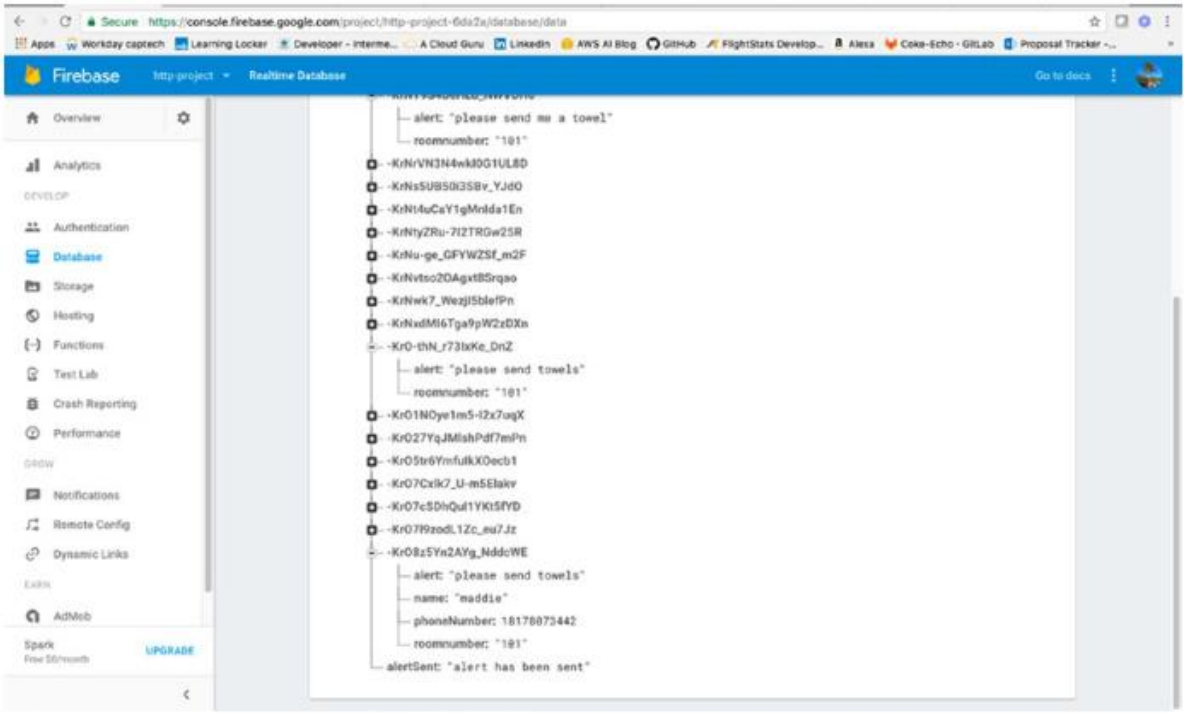
Gambar 5.29. Postman menerima permintaan yang sama dan akan menerima format isi JSON

Setelah bagian pengiriman pesan teks dari fungsi ini selesai, kita dapat melanjutkan dan menambahkan logika untuk ditambahkan ke basis data Realtime setelah pesan teks terkirim. Untuk melakukannya, kita akan menambahkan kolom alertSent dengan pesan yang menjelaskan apakah peringatan telah terkirim.

```
return event.data.ref.parent.child('alertSent').set("alert has been
sent");
```

Kami akan menulis pernyataan return ini di akhir file JavaScript penyimpanan kami. Pernyataan ini menetapkan saudara alertSent di database Alerts dan mengembalikan promise. Kami menetapkan saudara agar kami dapat menjaga struktur data peringatan kami tetap sama sambil tetap menyediakan informasi baru.

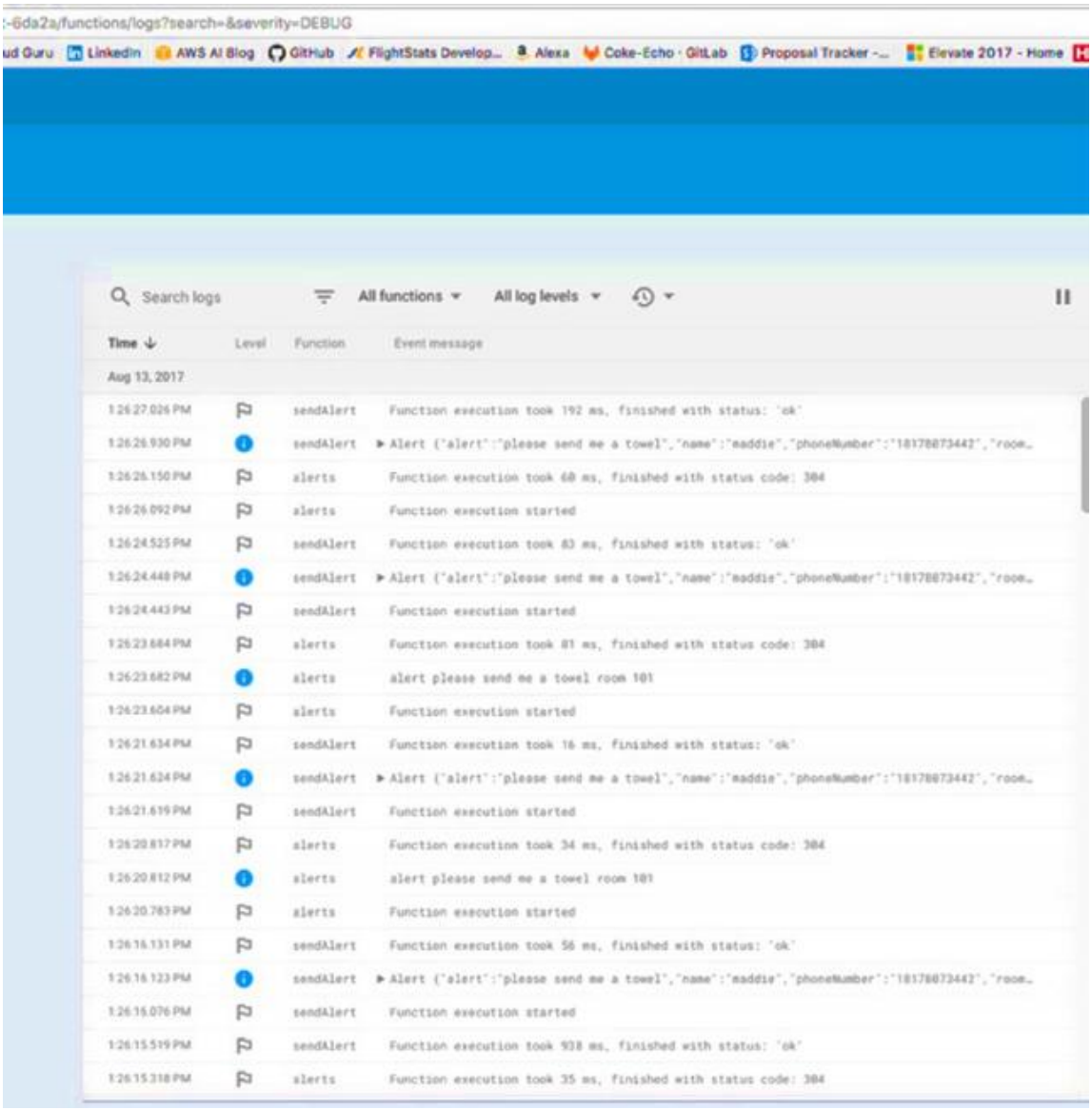
Saat kami memicunya lagi, kami sekarang akan melihat nilai baru ini di database Realtime kami (Gambar 5.30).



Gambar 5.30. Basis data Realtime kami kini telah diperbarui untuk mencerminkan alertSent saudara baru

Kami juga dapat memeriksa status fungsi kami melalui kemampuan pencatatan Firebase. Jika Anda membuka tab Functions, Anda akan melihat tab Logs di bagian atas. Ini akan menampilkan log untuk semua fungsi dalam proyek yang dipilih. Log akan diurutkan berdasarkan waktu dan akan membedakan antara pernyataan `console.log`, pernyataan function success, dan error. Anda juga akan dapat melihat fungsi mana yang terkait dengan log tersebut.

Pada Gambar 5.31, Anda dapat melihat bahwa saya memiliki log masuk untuk kedua fungsi saat salah satunya memicu yang lain.



Gambar 5.31. Pencatatan log di Firebase sangat mirip dengan pencatatan log di Cloud Functions di konsol Google

Anda juga dapat melihat log ini di konsol Google Cloud saat kita menggunakan proyek Google Cloud untuk mengimpor ke Firebase. Secara keseluruhan, Firebase memiliki banyak kemampuan yang sama dengan yang disediakan lingkungan Google Cloud. Ada baiknya untuk mengetahui hal ini karena kedua lingkungan tersebut sangat terintegrasi dan ke depannya saya prediksi mereka akan semakin terintegrasi. Di bagian berikutnya, kita akan melihat pemicu pub/sub di konsol Google Cloud dan mempelajari cara mengambil proyek Google Cloud kita dan melihatnya serta menjalankannya menggunakan Firebase.

5.8 MENINGKATKAN FUNGSI SERVERLESS

Peningkatan dengan memisahkan logika dan memanfaatkan kerangka kerja serverless Pisahkan logika AWS dari pengendali:

1. Gunakan variabel lingkungan untuk logika khusus AWS atau pindahkan logika AWS ke folder bersama.

2. Buat folder Layanan yang khusus untuk AWS dan menyajikan data DynamoDB. Manfaatkan Kerangka Kerja Serverless:
    1. Ikuti petunjuk untuk pengaturan AWS pada Kerangka Kerja Serverless.
    2. Kembangkan dan terapkan fungsi menggunakan Kerangka Kerja Serverless, bukan secara manual.

Kode untuk kedua peningkatan ini pada proyek dapat ditemukan di sini: https://github.com/mgstigler/Serverless/tree/master/AWS/aws-service/HTTPTrigger.

Di bagian berikutnya, kita akan menggunakan keterampilan dan alat yang kita pelajari dengan Pemicu HTTP untuk membuat fungsi Lambda terpisah yang dipicu oleh peristiwa penyimpanan.

Peristiwa Pub/Sub

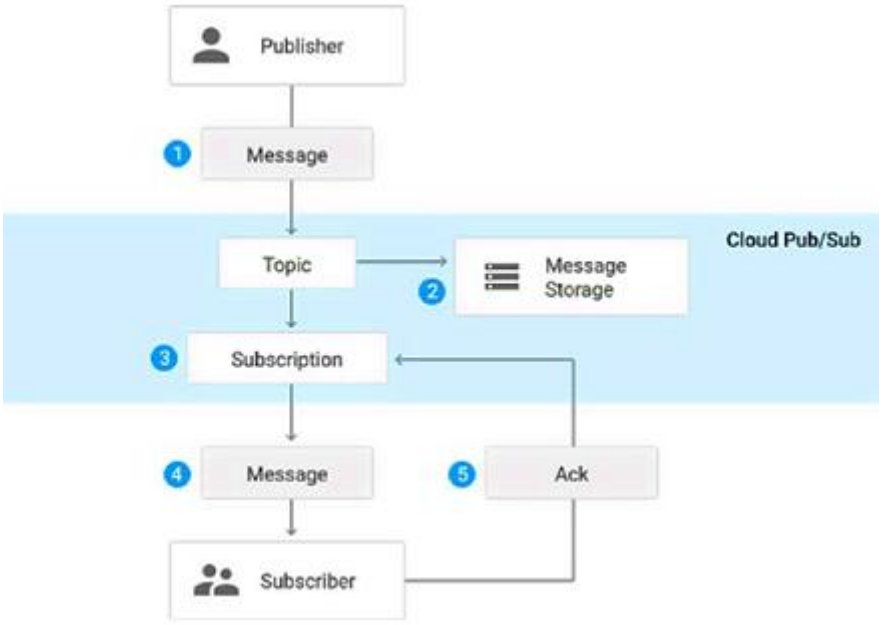
Dalam fungsi tanpa server ini, kita akan membuat fungsi yang dipicu oleh peristiwa Pub/Sub. Kita akan membuat fungsi ini di konsol Google Cloud dan kemudian mengimpor proyek ke Firebase untuk mengujinya dan melihat log.

Apa itu Google Cloud Pub/Sub?

Google Cloud Pub/Sub menyediakan middleware berorientasi pesan yang dapat diskalakan, fleksibel, dan andal. Ia menyediakan pengiriman pesan asinkron banyak-ke-banyak yang memisahkan pengirim dan penerima. Ia menawarkan komunikasi yang sangat tersedia antara aplikasi yang ditulis secara independen. Google menyediakan beberapa skenario umum yang menggambarkan kasus penggunaan untuk layanan Pub/Sub. Skenario ini meliputi:

- *Menyeimbangkan beban kerja dalam kluster jaringan:* Antrean tugas yang besar dapat didistribusikan secara efisien di antara banyak pekerja, seperti instans Google Compute Engine.
- *Menerapkan alur kerja asinkron:* Aplikasi pemrosesan pesanan dapat menempatkan pesanan pada topik, yang dapat diproses oleh satu atau beberapa pekerja.
- *Mendistribusikan pemberitahuan peristiwa:* Layanan yang menerima pendaftaran pengguna dapat mengirimkan pemberitahuan setiap kali pengguna baru mendaftar, dan layanan hilir dapat berlangganan untuk menerima pemberitahuan tentang peristiwa tersebut.
- *Menyegarkan cache terdistribusi:* Aplikasi dapat menerbitkan peristiwa pembatalan untuk memperbarui ID objek yang telah berubah.
- *Pencatatan log ke beberapa sistem*: Instans Google Compute Engine dapat menulis log ke sistem pemantauan, ke database untuk kueri selanjutnya, dan seterusnya.
- *Streaming data dari berbagai proses atau perangkat:* Sensor residensial dapat mengalirkan data ke server backend yang dihosting di cloud.
- *Peningkatan keandalan:* Layanan Compute Engine zona tunggal dapat beroperasi di zona tambahan dengan berlangganan topik umum, untuk memulihkan dari kegagalan di zona atau wilayah.

Konsep utama dalam Pub/Sub mencakup penerbit, pesan, topik, langganan, dan pelanggan. Gambar 5-32 mengilustrasikan bagaimana komponen-komponen ini bekerja bersama untuk menciptakan alur pesan yang terpadu.



Gambar 5.32. Penerbit membuat pesan yang diposting ke topik dengan serangkaian langganan yang semuanya menerima pesan

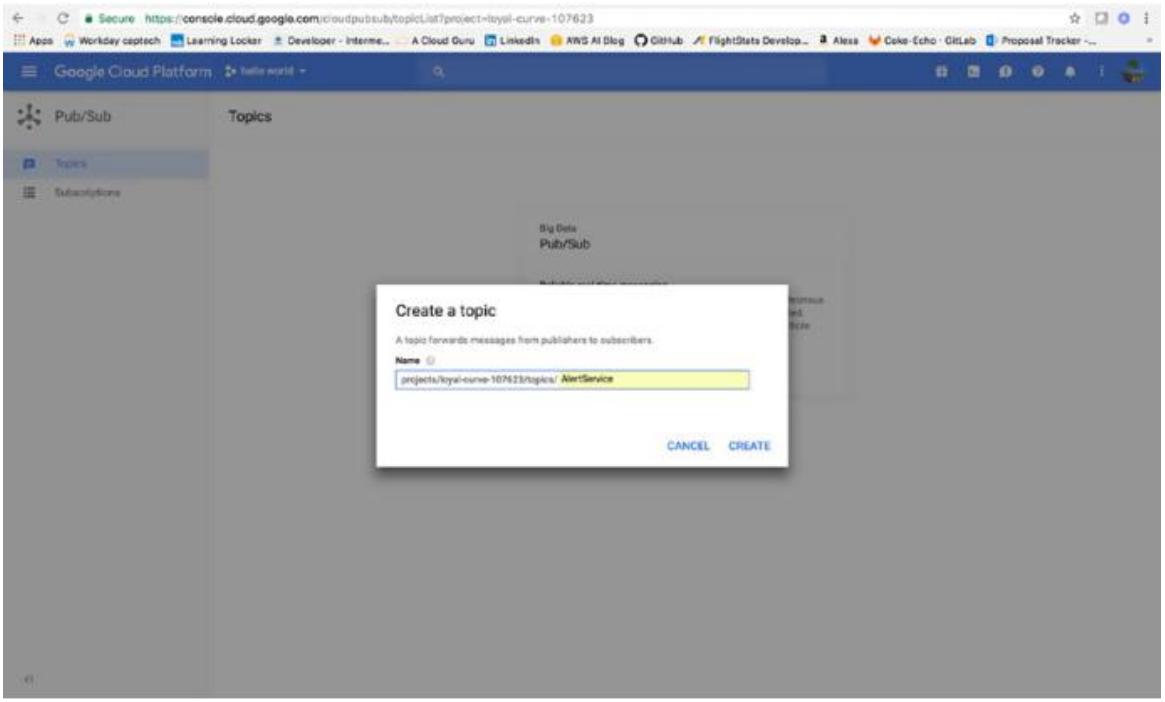
Berikut ini menjelaskan masing-masing sumber daya yang digunakan dalam model. Penting untuk memahami konsep-konsep ini sebelum memulai pembuatan fungsi tanpa server:

- Topik: Sumber daya bernama tempat pesan dikirim oleh penerbit.
- Langganan: Sumber daya bernama yang mewakili aliran pesan dari satu topik tertentu, yang akan dikirimkan ke aplikasi langganan.
- Pesan: Kombinasi data dan atribut (opsional) yang dikirim penerbit ke topik dan akhirnya dikirimkan ke pelanggan.
- Atribut pesan: Pasangan kunci-nilai yang dapat ditentukan penerbit untuk pesan.

Google Cloud memiliki konsep fungsi latar belakang, yang beroperasi secara berbeda dari fungsi HTTP. Kami menggunakan fungsi latar belakang setiap kali kami ingin fungsi kami dipanggil secara tidak langsung melalui pesan pada topik Google Cloud Pub/Sub atau perubahan dalam bucket Google Cloud Storage. Fungsi latar belakang mengambil dua parameter, peristiwa dan fungsi panggilan balik opsional.

Membuat Fungsi Pub/Sub

Kita akan melanjutkan dan membuat fungsi pemicu Pub/Sub. Pertama, kita perlu membuat topik Pub/Sub yang ingin kita picu fungsinya. Dari konsol Google Cloud, navigasikan ke layanan Pub/Sub dan klik Buat Topik (Gambar 5.33). Kita akan membuat topik yang disebut AlertService. Ini akan menjadi topik yang memicu fungsi kita.



Gambar 5.33. Buat topik untuk memicu fungsi kita. Setiap posting ke topik ini akan menyebabkan fungsi kita dijalankan.

Setelah kita membuat topik, kita akan menavigasi kembali ke panel Google Cloud Functions dan membuat fungsi baru dalam proyek kita saat ini. Kita akan menjadikan topik Pub/Sub yang baru saja kita buat sebagai pemicu. Untuk menguji fungsionalitas penggunaan topik Pub/Sub, kita akan menginisialisasi fungsi kita dengan templat Google Cloud yang disediakan. Ini cukup menuliskan peristiwa ke log.

```
/**
  * Triggered from a message on a Cloud Pub/Sub topic.
  *
  * @param {!Object} event The Cloud Functions event.
  * @param {!Function} The callback function.
  */
exports.subscribe = function subscribe(event, callback) {
  // The Cloud Pub/Sub Message object.
  const pubsubMessage = event.data;

  // We're just going to log the message to prove that
  // it worked.
  console.log(Buffer.from(pubsubMessage.data, 'base64').toString());
  // Don't forget to call the callback.
  callback();
};
```

Ketika kita kembali ke topik Pub/Sub, kita dapat mengeklik Publikasikan Pesan, menulis pesan, dan mengirimkannya ke topik kita. Jika semuanya berjalan sesuai rencana, kita dapat melihat pesan kita di log di Google Cloud. Ketika Anda mendapatkan respons berhasil, impor proyek kita ke konsol Firebase. Untuk melakukannya, navigasikan ke

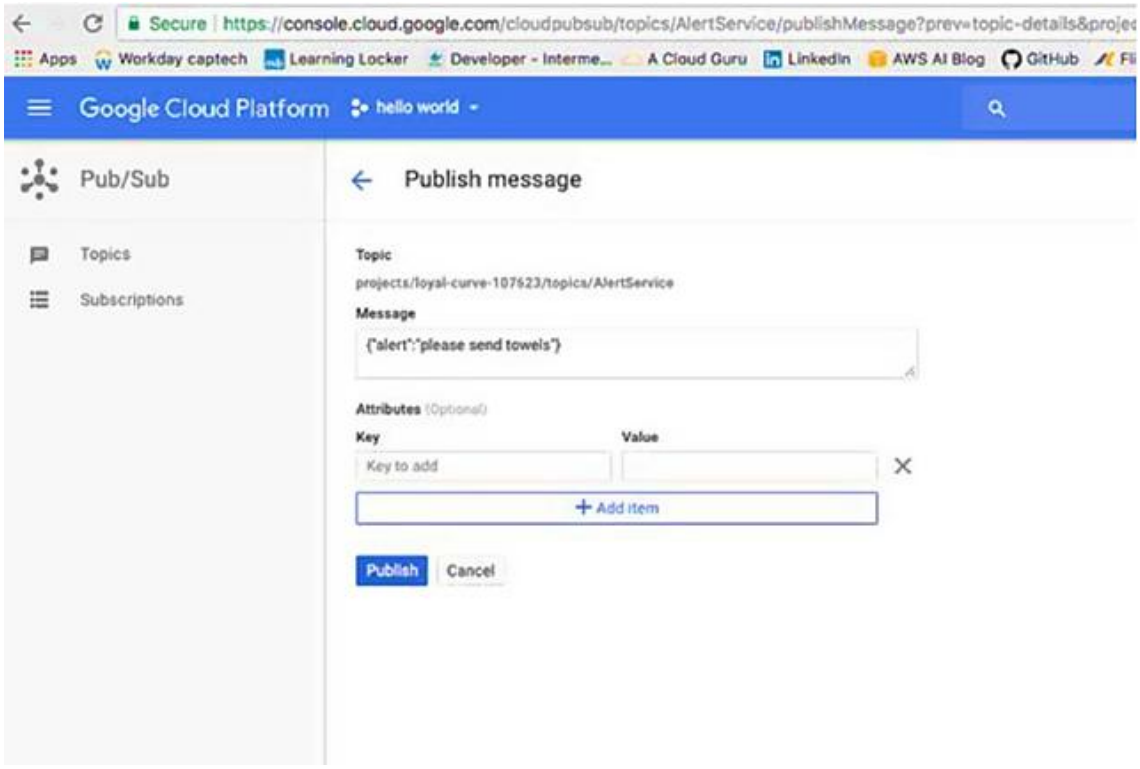
console.firebase.google.com dan klik Impor Proyek Google. Impor keseluruhan proyek. Nama proyek saya masih Hello World.

Anda kemudian dapat melihat fungsi Pub/Sub di tab Fungsi. Anda akan melihat topik AlertService sebagai pemicu dan eksekusi yang baru saja Anda panggil. Jika Anda mengeklik Log, Anda juga akan melihat log untuk pemanggilan Pub/Sub yang baru saja kita buat. Sekarang kita akan menambahkan fungsi kita sehingga ia melakukan lebih dari sekadar pencatatan ke konsol. Lanjutkan dan lakukan firebase init lain di folder proyek tempat file `index.js` Anda disimpan. Saat diminta untuk memilih proyek, pilih proyek Google Cloud yang sedang kita kembangkan. Sekarang, saat kita menerima permintaan melalui topik kita, kita ingin menyimpannya di tabel peringatan di basis data Realtime kita.

Kode berikut menunjukkan cara kita menangani permintaan yang masuk.

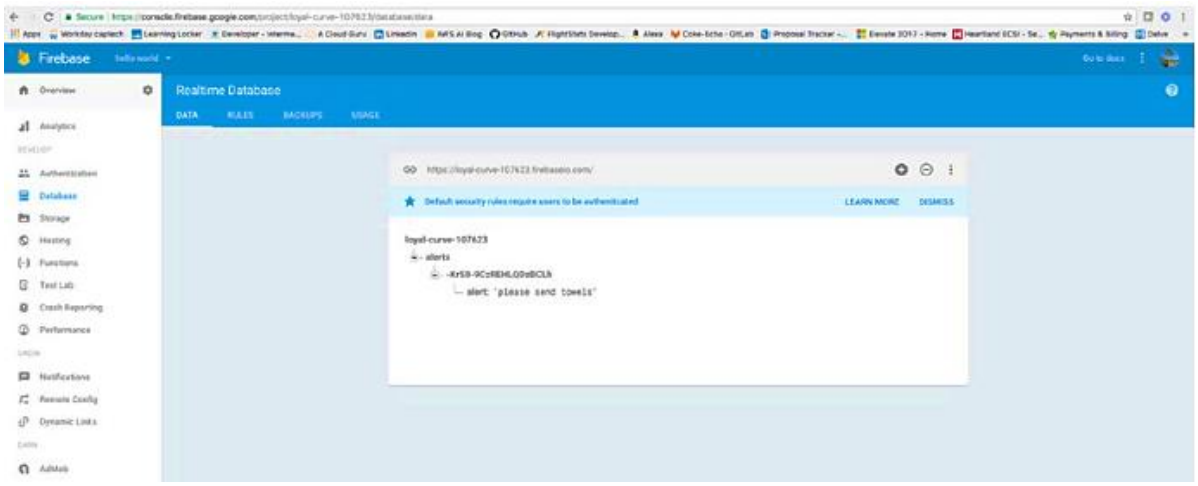
```
'use strict';
const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.subscribe = functions.pubsub.topic('AlertService')
.onPublish(event => {
  const pubSubMessage = event.data;
  // Get the `name` attribute of the PubSub message JSON body.
  let alert = null;
  try {
     alert = pubSubMessage.json.alert;
     console.log("Alert: " + alert);
     admin.database().ref('/alerts').push({alert:
     alert}).then(snapshot => {
       console.log("success!");
     });

  } catch (e) {
    console.error('PubSub message was not JSON', e);
  }
});
```

Terapkan kode ini menggunakan perintah firebase deploy di terminal Anda. Kita kemudian dapat menguji fungsionalitas ini dengan kembali ke topik Pub/Sub dan mengirim pesan. Fungsi kita mencari badan JSON dengan kolom peringatan. Jadi permintaan akan terlihat seperti Gambar 5.34.



Gambar 5.34. Kirim pesan ke topik kita untuk memicu fungsi yang baru saja kita terapkan

Anda seharusnya dapat melacak kemajuan fungsi dengan melihat log di lingkungan Firebase. Anda juga dapat melacaknya di lingkungan fungsi Google Cloud. Fungsi kita diubah namanya menjadi subscribe dan Anda dapat melihat pemanggilannya di bawah Function Details (Gambar 5.35). Setelah berhasil, peringatan yang kita kirim di sepanjang Pub/Sub akan muncul di bawah alerts/ dalam basis data Realtime kita.



Gambar 5.35. Pesan kita sekarang tersimpan dalam basis data Realtime proyek kita

Kita sekarang memiliki fungsi yang dihosting di Google Cloud dan juga diakses oleh Firebase. Anda dapat melihat pencatatan log dan pemicu di kedua lingkungan dan dapat dengan mudah memperbarui dan menerapkan fungsi tersebut sehingga dapat diakses oleh

kedua konsol. Tujuan dari latihan ini adalah untuk menunjukkan komunikasi antara fungsi dan untuk menunjukkan cara mengakses pemicu Cloud dan memengaruhi basis data Firebase.

5.9 MENINGKATKAN FUNGSI TANPA SERVER

Peningkatan dengan mengintegrasikan dengan aplikasi kami sebelumnya. Tambahkan fungsi Pub/Sub ke file `index.js` asli kami yang terkait dengan Firebase.

1. Gunakan logika yang sama yang kami gunakan untuk pemicu penyimpanan kami dan tambahkan fungsi Pub/Sub kami ke index.js dalam fungsi HTTP kami untuk disebarkan bersama.
2. Ini akan menyatukan semua fungsi kami sebagai satu aplikasi. Sebarkan proyek sebagai proyek Google Cloud.
3. Hosting semua proyek di bawah Google Cloud dan impor seluruh proyek ke Firebase alih-alih menyimpannya secara terpisah.

5.10 KESIMPULAN

Dalam bab ini kami menjelajahi beberapa aplikasi tanpa server dengan fungsi Google Cloud. Kami melihat perbedaan antara Google Cloud dan AWS dan Azure. Kami juga mempelajari dasar-dasar Google Cloud termasuk menavigasi konsol, mengonfigurasi fungsi, menetapkan pemicu seperti HTTP dan pemicu latar belakang, serta beberapa layanan terintegrasi populer lainnya seperti Firebase. Pada titik ini, Anda memiliki pengalaman membangun fungsi dengan tiga penyedia cloud yang berbeda. Anda harus dapat membedakan antara penyedia dan memiliki pemahaman yang lebih baik tentang kelebihan dan kekurangan masing-masing. Kami juga telah menjajaki beberapa cara untuk menerapkan arsitektur tanpa server. Beberapa aplikasi lebih bersifat personal dan yang lainnya melayani kasus bisnis yang sebenarnya.

BAB 6
PENDEKATAN AGNOSTIK

6.1 PENDAHULUAN

Pada titik ini, kita telah menjelajahi arsitektur tanpa server menggunakan tiga penyedia cloud: Amazon Web Services, Microsoft Azure, dan Google Cloud. Kita telah membuat aplikasi yang menggunakan pemicu HTTP dan pemicu penyimpanan serta menanggapinya dengan membuat perubahan pada layanan khusus penyedia. Melalui latihan ini, kita telah melihat banyak kasus penggunaan untuk jenis aplikasi ini dan kasus penggunaan khusus penyedia.

Dalam bab ini, kita akan mundur selangkah dan melihat cara membuat solusi yang tidak bergantung pada penyedia tertentu. Kita akan menjelajahi kasus penggunaan dan contoh arsitektur aplikasi yang menghapus logika khusus penyedia dari fungsi dan memanfaatkan universalitas kode tanpa server untuk menyediakan solusi yang sepenuhnya agnostik.

> *Catatan: Anda akan memerlukan pemahaman dasar tentang cara kerja masing-masing layanan penyedia untuk menyelesaikan latihan dalam bab ini. Jika Anda belum mempelajari contoh untuk ketiga penyedia, silakan pelajari.*

Dalam bab ini, kita akan membahas kebutuhan akan solusi agnostik, apa artinya bagi aplikasi nirserver di masa mendatang, pendekatan yang akan kita gunakan untuk menyajikan solusi, kode yang akan kita gunakan untuk membuat solusi, dan contoh nirserver yang menggunakan pemicu basis data.

6.2 KEBUTUHAN AKAN SOLUSI AGNOSTIK

Sebelum kita mulai mengembangkan solusi agnostik, saya pikir penting untuk mengidentifikasi mengapa ini merupakan konsep yang sangat penting untuk ditangani dan mengapa hal ini layak untuk didekati. Saya akan mulai dengan mengidentifikasi studi kasus terkini yang telah saya kembangkan untuk klien saya, sebuah perusahaan asuransi global.

Kondisi Terkini

Klien saya adalah pemimpin dunia dalam industri asuransi dan bantuan perjalanan. Mereka membantu orang kapan saja, di mana saja untuk menemukan solusi bagi masalah terkait perjalanan apa pun. Mitra mereka berjumlah ribuan dan meliputi agen perjalanan, maskapai penerbangan, resor, situs web, pialang tiket acara, perusahaan, universitas, dan perusahaan kartu kredit.

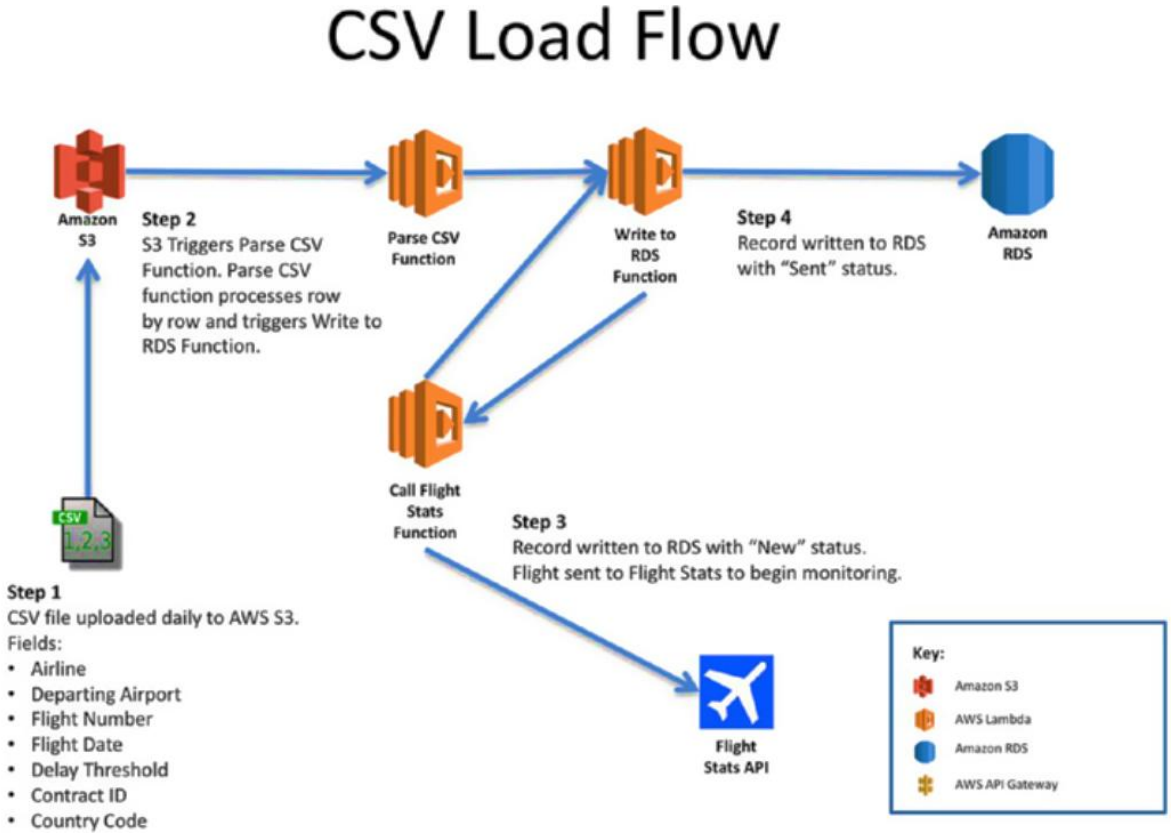
Perusahaan ini dimiliki oleh perusahaan asuransi terdiversifikasi terbesar di dunia. Berkat skala perusahaan induknya, perusahaan ini mampu menyediakan produk inovatif dengan cakupan di seluruh dunia dengan harga yang kompetitif.

Lebih dari 25 juta pelancong bergantung pada perusahaan ini setiap tahun untuk melindungi mereka saat mereka jauh dari rumah. Perusahaan ini menyediakan berbagai layanan bantuan dan layanan pramutamu yang dapat membantu pelanggannya mendapatkan

hasil maksimal dari perjalanan mereka. Perusahaan ini telah membantu melindungi tatanan Amerika selama lebih dari 100 tahun. Bahkan, klien ini mengasuransikan penerbangan pertama Wright Bersaudara, pembangunan Jembatan Golden Gate, dan banyak film Hollywood.

Dengan reputasi dan jumlah layanan serta klien yang terlibat, klien ini sangat mementingkan tumpukan teknis dan arsitekturnya. Dengan kondisi saat ini, perusahaan saat ini terlibat dengan satu penyedia cloud tetapi sebagian besar infrastrukturnya bersifat internal. Manajemen ingin memindahkan seluruh infrastruktur ke cloud, tetapi dengan begitu banyak ketergantungan dan kantor pusat di luar negeri, hal itu jelas lebih mudah diucapkan daripada dilakukan.

Langkah pertama perusahaan menuju transisi ke cloud melibatkan aplikasi awal yang mengotomatiskan pembayaran klaim untuk penundaan dan pembatalan perjalanan. Arsitektur yang diusulkan untuk solusi ini melibatkan beberapa fungsi AWS Lambda dan banyak layanan di AWS. Ada empat bagian utama dalam pelaksanaan ini yang dirinci dalam Gambar 6.1.

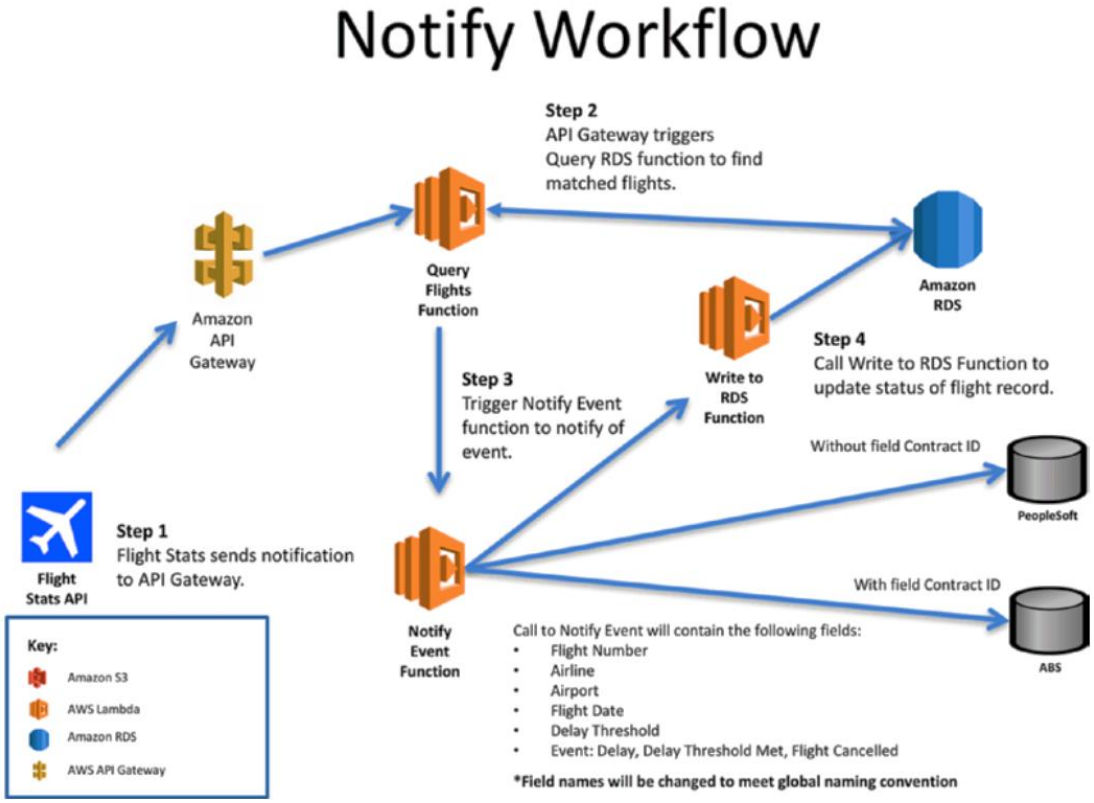


Gambar 6.1. Alur beban CSV menunjukkan transfer data dari lokal ke fungsi AWS Lambda

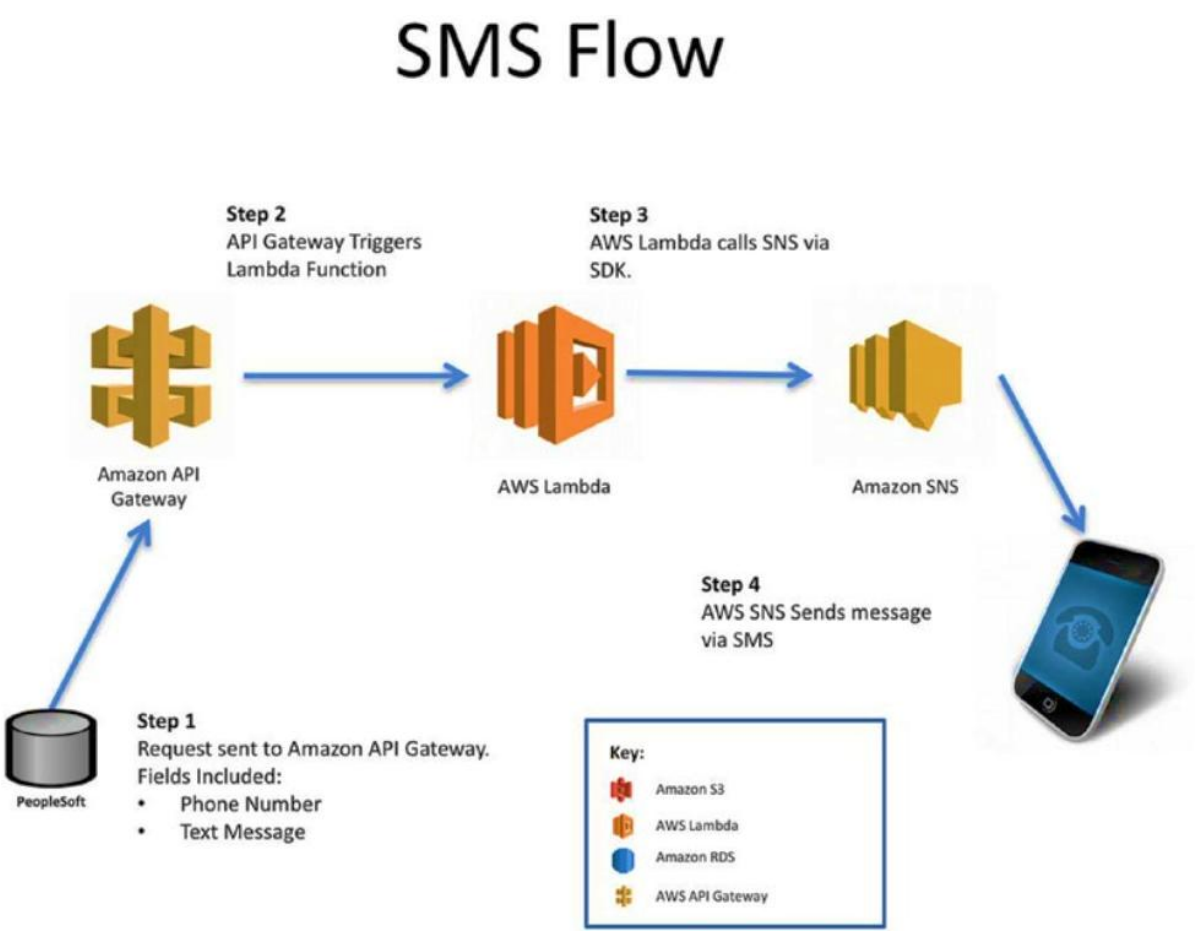
Arsitektur terkini dengan solusi cloud ini melibatkan komunikasi antara basis data lokal dan basis data AWS. Situs lokal masih menyimpan semua data pelanggan dan penerbangan yang diperlukan agar aplikasi ini berfungsi. Selain itu, perubahan yang diperlukan untuk melakukan transisi ini bukanlah perubahan sederhana. Perubahan tersebut merupakan perubahan di seluruh perusahaan yang memerlukan banyak waktu, sumber daya, dan

pelatihan proses. Bahkan menyediakan file CSV sederhana memerlukan persetujuan dari kantor pusat, keamanan, dan seluruh departemen TI. Hal ini sangat standar di seluruh industri.

Gambar 6.2 menunjukkan alur kerja notifikasi, yang menerima permintaan flightstats yang masuk (seperti penundaan atau pembatalan).

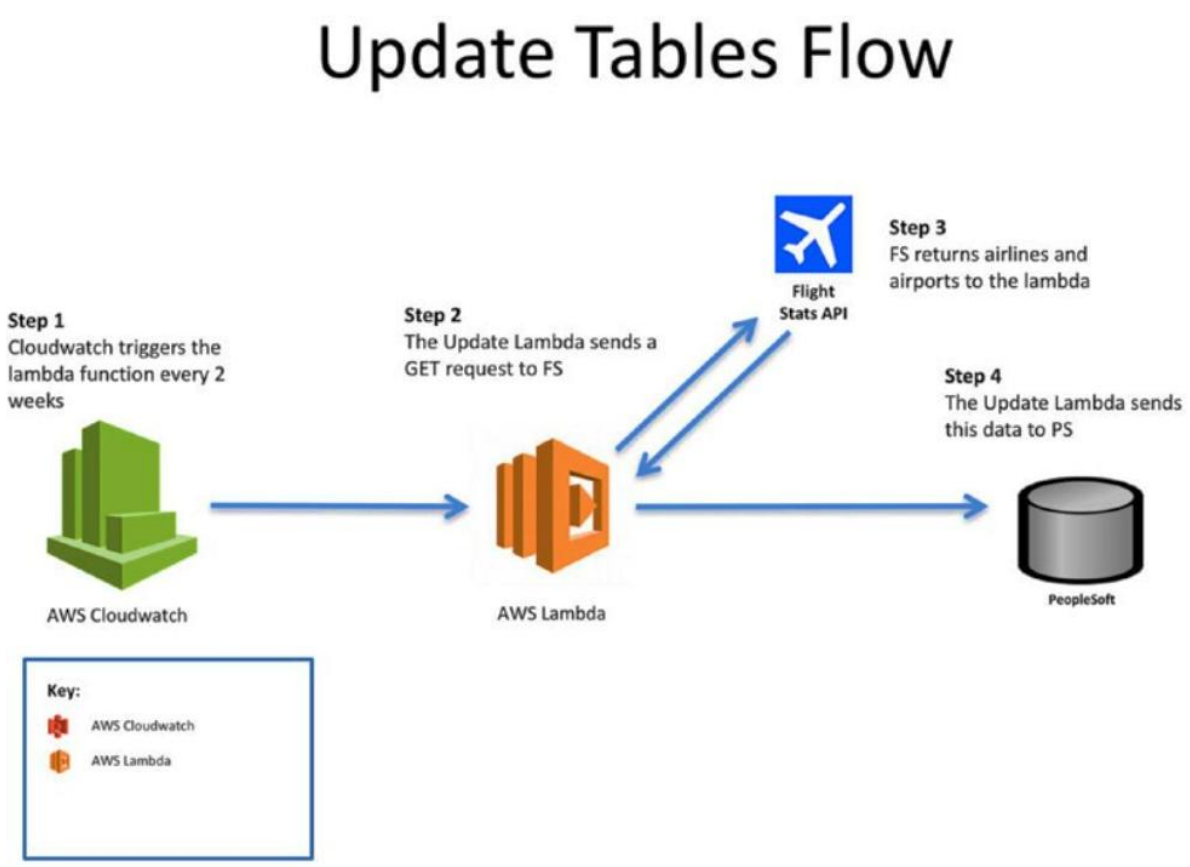


Gambar 6.2. Alur kerja Notify menunjukkan aliran data dari FlightStats API kembali ke fungsi Lambda yang menangani data penerbangan



Gambar 6.3. Alur SMS menunjukkan kemampuan aplikasi untuk memberi tahu pelanggan saat penerbangan ditunda atau dibatalkan sehingga klaim dapat diajukan secara otomatis Alur kerja Notify, mirip dengan alur CSV, bergantung pada komunikasi dengan basis data lokal agar berhasil. Alur kerja ini juga bergantung pada transfer dan pembaruan data di Amazon RDS untuk mengirim data yang sesuai kembali ke basis data lokal.

Gambar 6.3 menunjukkan alur kerja SMS, yang mengambil nomor dan pesan teks dari basis data lokal dan mengirimkan teks ke pelanggan. Sekali lagi, alur bergantung pada data internal yang mengalir masuk dan keluar untuk dapat mengirim SMS kepada pelanggan dengan informasi penundaan dan pembatalan penerbangan mereka. Alur berikutnya memperbarui informasi tabel dalam basis data di tempat untuk dihapus setiap dua minggu. Hal ini dipicu oleh pengatur waktu cloud watch yang menjalankan fungsi Lambda yang menjangkau sistem di tempat untuk menghapus data penerbangan lama dan usang. Gambar 6-4 menunjukkan proses ini.



Gambar 6.4. Alur Tabel Pembaruan menunjukkan alur fungsi Lambda yang memperbarui basis data cloud serta basis data lokal

Inti dari semua alur dan proses ini dalam arsitektur baru adalah bahwa semuanya bergantung pada infrastruktur lokal dan infrastruktur AWS. Meskipun ini adalah sistem yang dirancang dengan sangat baik dan merupakan ide yang bagus dari sudut pandang pelanggan, ada banyak masalah dari sudut pandang bisnis dan arsitektur secara keseluruhan.

Masalah Bisnis

Mengingat arsitektur dan diagram alur yang disediakan, ada beberapa masalah bisnis yang muncul. Salah satu masalah utama adalah ketergantungan pada fungsi AWS Lambda. Salah satu solusi yang diusulkan adalah menghosting aplikasi pada instans EC2, tempat lingkungan dapat dikontrol dan didokerisasi. Namun, solusi ini juga melibatkan penulisan semua komponen SNS secara independen dari layanan karena kami ingin sepenuhnya dihapus dari lingkungan AWS.

Klien saat ini bergantung pada satu penyedia cloud untuk sebagian layanannya tetapi tidak ingin sepenuhnya bergantung pada penyedia ini, juga tidak ingin bergantung sebagian pada banyak penyedia cloud. Ketakutan saat menggunakan AWS adalah ketergantungan pada penyedia ini secara keseluruhan. Dan meskipun klien berencana mengalihkan seluruh infrastrukturnya ke AWS, manajemen masih memiliki ketakutan yang sangat rasional untuk terikat pada AWS sebagai vendor.

Menurut saya, ada beberapa alasan di balik ketakutan akan ketergantungan pada vendor. Salah satunya adalah kebutuhan untuk menghindari upaya rekayasa ulang yang panjang jika klien memutuskan untuk mengganti penyedia. Hal ini masuk akal karena hal ini

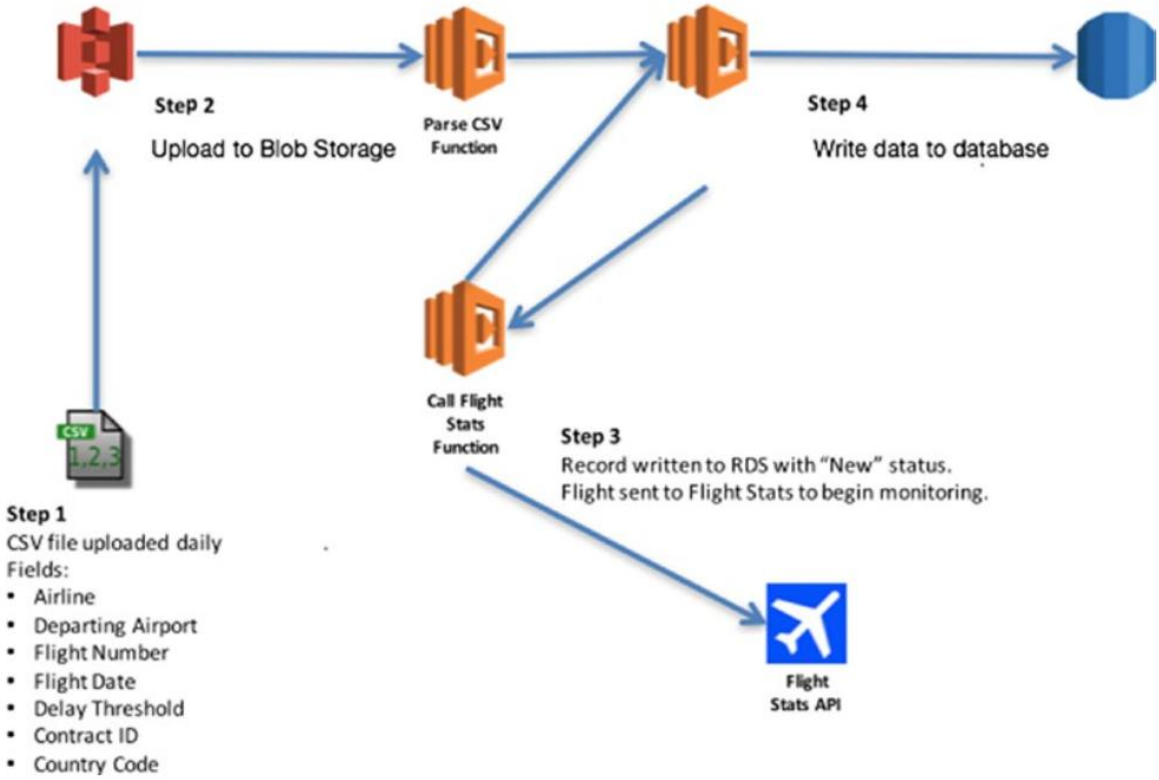
merupakan sesuatu yang dihadapi perusahaan setiap hari. Jika kita tinjau lebih jauh, perusahaan yang beralih dari layanan lokal ke layanan cloud sudah melakukan langkah besar yang membutuhkan banyak upaya dan banyak rekayasa proses. Selain itu, mereka menghadapi ketakutan untuk beralih penyedia, yang membutuhkan upaya yang sama lagi.

Alasan lain untuk ketakutan ini terletak pada gagasan bahwa satu vendor komputasi cloud akan mengambil alih pasar dan membuatnya kurang menjadi pilihan. Dalam kasus ini, satu-satunya solusi adalah dengan menggunakan kembali dan merancang arsitektur untuk vendor yang berbeda. Ketakutan ini sangat valid dan umum di antara perusahaan besar dan kecil. Ketakutan kehilangan semua data karyawan dan fungsionalitas aplikasi inti sangat nyata dan merupakan kekhawatiran yang baik bagi perusahaan. Sementara klien saya dijual di AWS sebagai vendor, manajemen tahu ini adalah permainan akhir yang potensial untuk solusi cloud dan menyampaikan kekhawatiran mereka kepada kami dengan cara yang sangat wajar. Salah satu pilihannya adalah memiliki solusi lintas penyedia. Sementara ini akan memungkinkan perusahaan untuk menggunakan layanan dan produk dalam lingkungan lintas penyedia, perusahaan masih terkunci pada vendor mana pun yang dipilihnya untuk layanan apa pun yang disediakannya.

Dari perspektif bisnis, kekhawatiran tentang keterikatan vendor adalah masalah serius. Selain itu, transformasi data historis yang besar ke cloud juga membebani bisnis. Biayanya mungkin lebih baik dalam jangka panjang, tetapi melihatnya dalam fase penerapan yang singkat dan cepat, cenderung tampak mahal dan berisiko. Perusahaan ingin dapat menggunakan sumber daya penyedia cloud yang dapat diskalakan dan fleksibel, tetapi berjuang untuk membuat perubahan yang dapat memengaruhi masa depan infrastruktur perusahaan serta responsivitas terhadap kliennya.

6.3 SOLUSI YANG DIREKOMENDASIKAN

Untuk dapat mengatasi semua kekhawatiran klien sambil tetap memberikan solusi terbaik dan solusi yang paling hemat biaya, solusi terbaik adalah mengembangkan aplikasi yang independen dari penyedia. Dalam situasi ini, perusahaan tetap mendapatkan semua keuntungan karena mengandalkan penyedia cloud tanpa mengalami potensi kerugian karena terjebak dalam ketergantungan pada vendor. Bagian yang sulit dari ini adalah menerapkan solusi dengan benar. Kami telah melihat skenario di mana kami menghapus logika cloud dari fungsionalitas. Ini akan dibangun dari ide tersebut dan menghapus logika penyedia cloud sepenuhnya dan menyediakannya dalam berbagai bentuk. Misalnya, kami akan memiliki satu solusi yang dapat beralih antara AWS, Azure, dan Google Cloud secara bersamaan. Gambar 6.5 menunjukkan eksekusi kasar solusi ini dengan langkah pertama dalam aliran beban CSV keseluruhan yang diperbarui.



Gambar 6.5. Solusi diimplementasikan tanpa mengetahui siapa penyedianya

Dalam alur ini, kami membuat interaksi yang sama dengan yang terjadi selama alur CSV, kecuali bahwa penyedia bukan bagian darinya. Ini akan menarik bagi banyak bisnis karena mereka akan dapat mengimplementasikan solusi cloud tanpa takut akan ketergantungan vendor, tetapi juga dengan rasa takut akan ketergantungan vendor.

Dengan cara kerja solusi ini, Anda akan dapat eksis sebagai perusahaan dengan ketergantungan vendor dan kemudian beralih vendor jika perlu. Untuk solusi yang kami rekomendasikan, kami akan melihat solusi yang mengandalkan ketergantungan vendor ini tetapi tahu bahwa itu adalah kelemahan yang cukup untuk membuat kode langsung untuk kelemahan ini.

Menetapkan Pendekatan

Meskipun kami dapat mengidentifikasi kebutuhan bisnis yang jelas untuk pendekatan agnostik, juga tepat untuk menilai situasi ini pada pendekatan aplikasi per aplikasi. Misalnya, saya mungkin mengembangkan aplikasi Alexa Skills yang saya ingin agar dapat diakses hanya oleh fungsi Amazon Lambda. Dalam kejadian lain, misalnya dengan situasi klien, saya ingin aplikasi tersebut dapat diakses oleh semua fungsi klien. Atau, misalkan saya ingin kode untuk Alexa Skill saya diterjemahkan ke Google Home, Siri milik Apple, atau Cortana milik Microsoft. Ini adalah kebutuhan yang sangat wajar. Perusahaan ingin produk mereka dapat diakses di berbagai platform dan produk untuk menjangkau audiens yang lebih luas. Arsitektur tanpa server membuat lompatan ini lebih memungkinkan daripada yang Anda kira. Jika kita mencoba menerapkan logika ini dalam mesin virtual atau aplikasi dengan infrastruktur, kita akan mengalami kesulitan. Kita akan diminta untuk menyiapkan layanan yang sama sekali berbeda dalam lingkungan ini dan membuatnya dapat diakses publik. Ketika Anda memikirkan berbagai kasus penggunaan untuk pendekatan agnostik, Anda dapat melihat bagaimana kode

itu sendiri dengan logika yang mendasarinya tidak perlu diubah. Satu-satunya hal yang memisahkan lingkungan aplikasi ini adalah lingkungan itu sendiri.

Kita akan mengeksplorasi bagaimana pendekatan agnostik dapat bekerja. Untuk melakukan ini, saya ingin memulai dari yang kecil lalu berkembang seiring kita mengembangkan aplikasi yang lebih besar. Untuk mencapai pendekatan yang dapat diskalakan dan diakses untuk masalah ini, menurut saya cara terbaik untuk mengatasinya adalah melalui langkah-langkah kecil. Serverless Framework memberi kita beberapa contoh tentang cara membuat fungsi tanpa server dengan benar dan agnostik. Cuplikan berikut menunjukkan contoh kode agnostik yang buruk.

```
const db = require('db').connect();
const mailer = require('mailer');

module.exports.saveUser = (event, context, callback) => {
  const user = {
    email: event.email,
    created_at: Date.now()
  }

db.saveUser(user, function (err) {
  if (err) {
    callback(err);
  } else {
    mailer.sendWelcomeEmail(event.email);
    callback();
  }
  });
};
```

Kode ini merupakan contoh yang buruk karena beberapa alasan:

- Logika bisnis tidak terpisah dari penyedia FaaS. Logika ini terikat pada cara AWS Lambda meneruskan data masuk (objek peristiwa Lambda).
- Pengujian fungsi ini akan bergantung pada layanan terpisah. Secara khusus, menjalankan instans basis data dan server email.

Bagaimana kita dapat melakukan refaktor ini dan membuatnya dapat dieksekusi tetapi tidak bergantung pada fungsi khusus AWS? Kode berikut menunjukkan bagaimana kita dapat memisahkan logika dari eksekusi.

```
class Users {
  constructor(db, mailer) {
    this.db = db;
    this.mailer = mailer;
  }

save(email, callback) {
  const user = {
    email: email,
    created_at: Date.now()
```

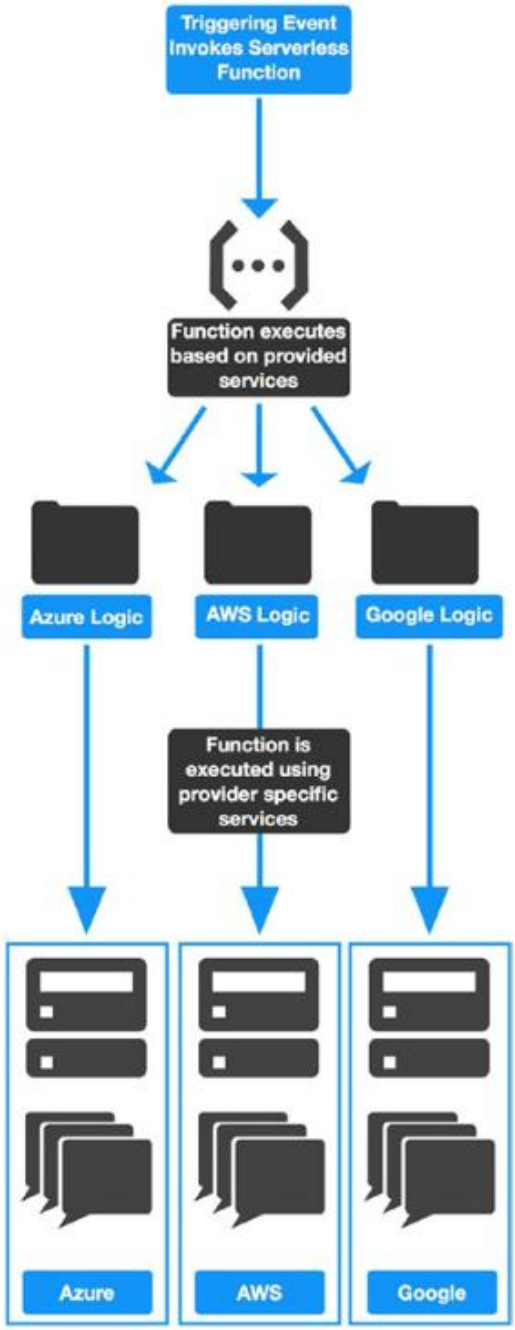
```
    }

  this.db.saveUser(user, function (err) {
    if (err) {
      callback(err);
    } else {
      this.mailer.sendWelcomeEmail(email);
      callback();
      }
      });
    }
  }
  module.exports = Users;
```

Kelas pengguna ini menangani permintaan masuk secara agnostik. Basis data dan mailer keduanya disediakan secara eksternal. Dalam kasus ini, kita dapat menyediakan kelas dengan instans db dan mailer Azure, instans db dan mailer Google, atau instans db dan mailer AWS.

Kode berikut menunjukkan modul indeks yang memanggil kelas Pengguna.

```
const db = require('db').connect();
const mailer = require('mailer');
const Users = require('users');

let users = new Users(db, mailer);

module.exports.saveUser = (event, context, callback) => {
  users.save(event.email, callback);
};
```

Sekarang, kelas yang telah kita definisikan di sini menjaga logika bisnis tetap terpisah. Lebih jauh, kode yang bertanggung jawab untuk menyiapkan dependensi, menyuntikkannya, memanggil fungsi logika bisnis, dan berinteraksi dengan AWS Lambda ada dalam berkasnya sendiri, yang akan lebih jarang diubah. Dengan cara ini, logika bisnis tidak bergantung pada penyedia, dan lebih mudah digunakan kembali dan diuji.



Gambar 6.6. Solusi yang diusulkan menjaga logika penyedia tetap terpisah dari operasi normal aplikasi. Saat fungsi diterapkan, kami memilih penyedia yang ingin kami terapkan dan hanya menggunakan logika dan layanannya.

Lebih jauh, kode ini tidak memerlukan menjalankan layanan eksternal apa pun. Alih-alih layanan db dan mailer yang sebenarnya, kita dapat meneruskan tiruan dan menegaskan jika `saveUser` dan `sendWelcomeEmail` telah dipanggil dengan argumen yang tepat.

Pengujian unit dapat dengan mudah ditulis untuk mencakup kelas ini. Pengujian integrasi dapat ditambahkan dengan memanggil fungsi (pemanggilan tanpa server) dengan alamat email tetap. Kemudian akan memeriksa apakah pengguna benar-benar disimpan ke DB dan apakah email diterima untuk melihat apakah semuanya berfungsi bersama.

Pendekatan kita untuk memecahkan skenario agnostik akan mengikuti jalur yang sama. Berkas indeks atau berkas utama yang mengeksekusi kode akan buta terhadap penyedia dan logika serta layanan yang terkait dengan penyedia tersebut. Gambar 6-6 mengilustrasikan kelayakan pendekatan ini.

Dengan solusi yang diusulkan ini, kode khusus penyedia dipisahkan dari berkas indeks. Saat waktunya untuk menerapkan, Anda dapat memilih kode logika penyedia yang sesuai dan menerapkannya ke lingkungan penyedia tersebut. Untuk membuatnya lebih sederhana, kami akan menggunakan Serverless Framework untuk menerapkan kode tersebut.

Berkas `serverless.yml` menyertakan kolom input penyedia. Kami perlu mengubahnya tergantung pada penyedia dan penerapannya, memastikan pemicu dan layanan semuanya dinyatakan dengan benar per penyedia di dalam berkas ini. Kami bahkan dapat memiliki berkas serverless untuk setiap penyedia, lalu memilih penyedia mana yang akan diterapkan saat kami memutuskan penyedia. Kode berikut menunjukkan kepada kami di mana kami dapat mengubah penyedia penerapan saat kami perlu:

```
# serverless.yml

service: azfx-node-http

provider:
  name: azure
  location: West US

plugins:
  - serverless-azure-functions

functions:
  hello:
     handler: handler.hello
     events:
       - http: true
         x-azure-settings:
           authLevel : anonymous
```

Sekarang setelah kita memiliki pemahaman yang lebih baik tentang pendekatan desain kita, kita dapat menjelajahi kode Hello World yang akan kita gunakan untuk menguji bukti konsep dalam skala besar. Kita akan tetap menggunakan tiga penyedia yang telah kita gunakan (Amazon, Azure, dan Google) dan akan menerapkannya dengan setiap penyedia dan memastikan bahwa fungsinya berfungsi sama.

6.4 JELAJAHI KODE

Kita akan membuat struktur proyek umum untuk solusi agnostik kita berdasarkan pendekatan bukti konsep yang dijelaskan sebelumnya. Hal pertama yang akan kita lakukan

adalah membuat struktur proyek di mana file indeks berada di akar folder dan ada folder penyimpanan yang berisi folder AWS, Google, dan Azure yang terpisah (Gambar 6-7).

Gambar 6.7. Struktur proyek menyimpan pengetahuan khusus penyedia cloud di folder yang berbeda. Ini membuatnya terpisah dari berkas indeks dan juga tidak tertukar dengan logika penyedia lainnya.

Untuk demonstrasi pertama bukti konsep kami, kami akan membuat berkas indeks mencatat lingkungan penyedia tempatnya berada. Kami juga akan menggunakan Serverless Framework untuk menyebarkan ke lingkungan yang berbeda. Kami akan mulai dengan melihat berkas indeks kami.

```
// dependencies
var provider = 'aws';
var Provider = require('./storage/' + provider + '/provider')

exports.handler = function (event, context, callback) {
  Provider.printProvider("Hello World");
}
```

File indeks akan memiliki satu nilai yang harus Anda perbarui di antara penerapan, yaitu variabel penyedia. Kami akan menguji kemampuan untuk beralih dengan mudah di antara penyedia dan meneruskan variabel indeks ke lingkungan yang berbeda. Di dalam setiap folder penyedia, kami akan menambahkan file JavaScript penyedia.

Kode berikut menunjukkan apa yang saya miliki di file penyedia saya. Anda perlu mengubah respons per file penyedia.

```
// dependencies

module.exports = {
  printProvider: function(message) {
    console.log('Message: ' + message + ' from AWS!');
  }
}
```

Fungsi provider.printProvider hanya mencatat pesan yang diteruskan dari file indeks dengan lingkungan penyedia. Kita juga ingin menginisialisasi file `serverless.yml` di dalam setiap folder penyedia. Setiap file serverless akan spesifik untuk penyedia.

Kode berikut menunjukkan file AWS `serverless.yml`.

```
  service: aws-nodejs

  package:
    individually: true
    exclude:
    - ./**

  provider:
    name: aws

  functions:
    helloworld:
      handler: index.handler
      package:
        include:
        - ../../index.js
        - ../../storage/aws/**
        - node_modules/**
```

Untuk mengganti penyedia, cukup ubah aws ke azure atau google. Anda juga perlu memperbarui jalur paket untuk mencerminkan jalur yang benar untuk logika penyedia. Dalam kasus ini, kita hanya perlu mengubah jalur folder setelah ../../storage/.

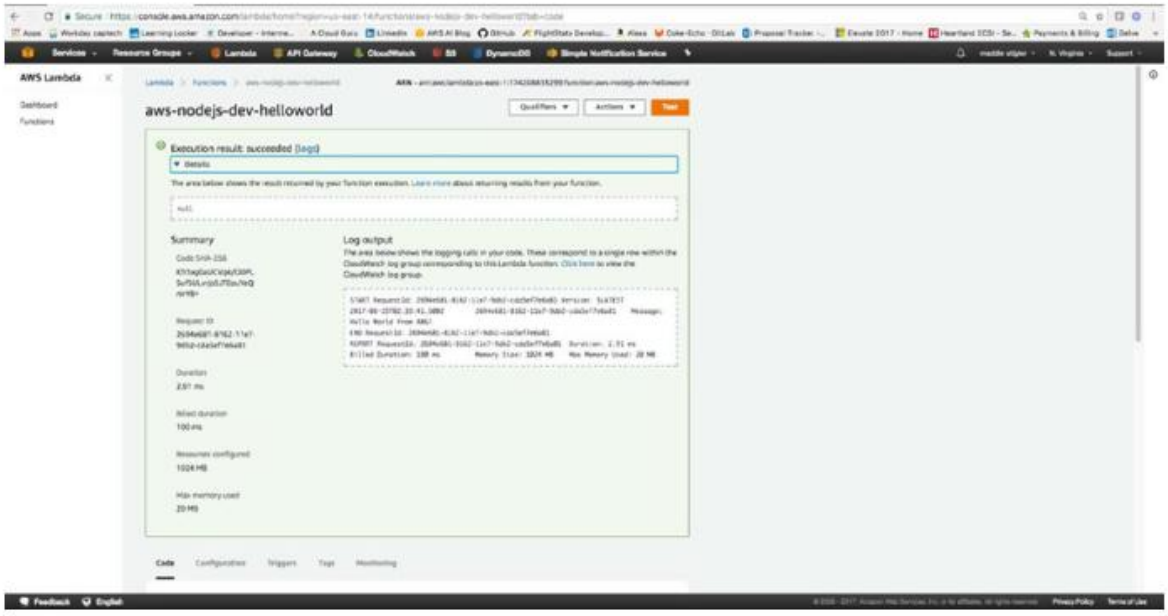
Akhirnya, kita siap untuk menyebarkan fungsi kita di lingkungan masing-masing. Untuk melakukannya, cukup navigasi ke direktori penyedia tempat Anda ingin menyebarkan dan masukkan perintah berikut ke terminal Anda:

```
serverless deploy
```

Ini akan menyebarkan fungsi yang disarankan ke lingkungan penyedia yang benar. Lakukan ini dengan ketiga penyedia. Setelah semuanya disebarkan, kita dapat masuk ke konsol masing-masing dan melihat efek perubahan kita.

> *Catatan: Saya menyadari ini terasa seperti tugas yang sangat berulang dengan banyak overhead, tetapi perlu diingat bahwa di dunia nyata, Anda mungkin tidak akan menyebarkan aplikasi Anda secara konsisten di lingkungan yang berbeda. Sebaliknya, Anda akan menyebarkannya saat Anda membutuhkannya dan saat Anda siap untuk mengganti penyedia.*

Kami tidak menetapkan pemicu untuk fungsi tanpa server kami, karena kami hanya ingin menguji struktur proyek sebagaimana adanya, sebagai bukti konsep. Jadi untuk menguji fungsi kami, di setiap lingkungan Anda dapat memicunya dengan permintaan apa pun. Jika implementasi konsep kami berhasil, Anda akan melihat pesan respons yang benar dengan penyedia yang benar di log (Gambar 6-8).



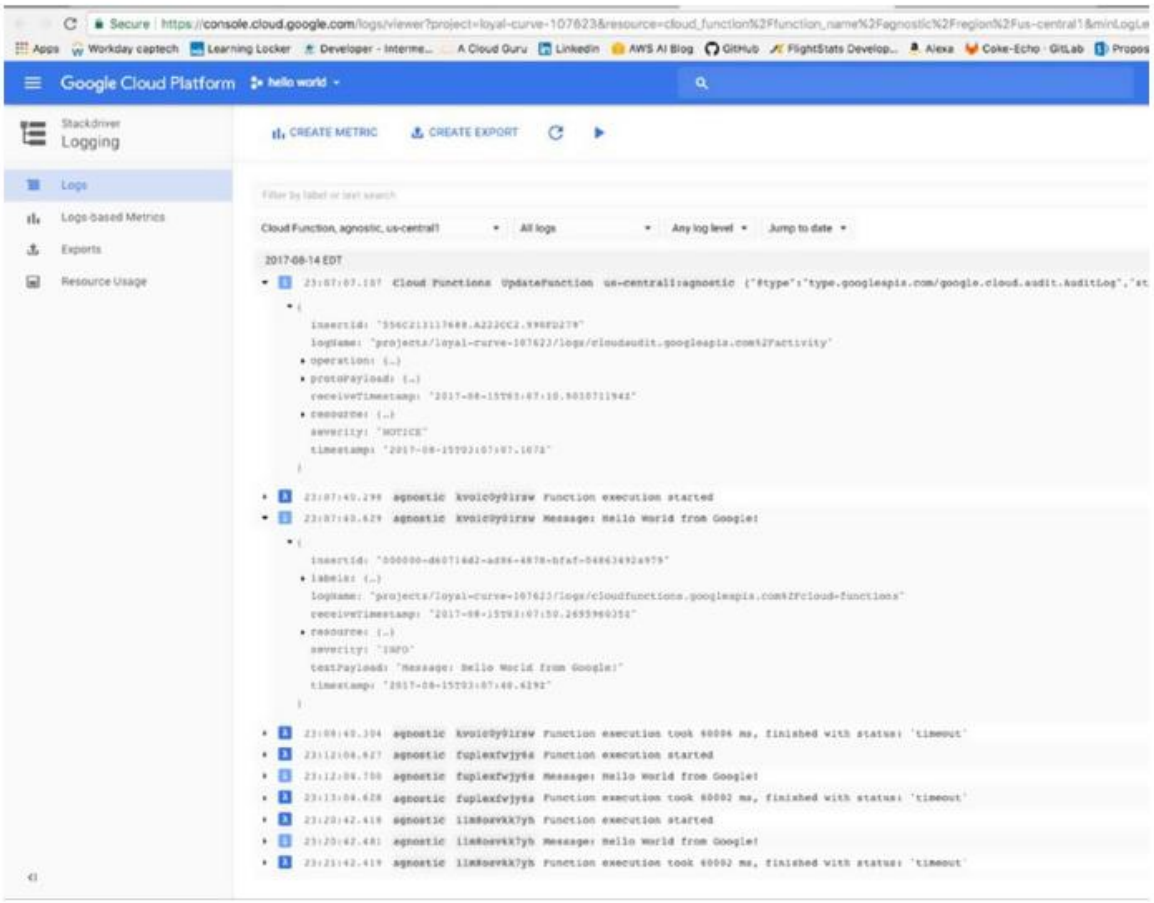
Gambar 6.8. Bukti konsep kami berhasil. Kami dapat melihat pesan dari AWS dengan jelas di log.

Kita seharusnya dapat melihat pesan ini berubah secara dinamis tergantung pada lingkungan tempat kita menyebarkannya. Satu pengembangan potensial di masa mendatang yang perlu diingat untuk pembuktian konsep ini adalah menentukan cara mendeteksi lingkungan kita dan menarik dari folder proyek yang benar setelah kita mengetahui penyedia mana yang kita gunakan.

Kemampuan itu akan membantu karena beberapa alasan. Yang pertama adalah kita akan menghilangkan proses manual yang terlibat dalam pengalihan penyedia dalam kode.

Beberapa kelompok inovatif telah mulai bergerak ke arah ini. Anda dapat melihat ini dengan Serverless Framework, yang mencoba membuatnya semudah mungkin untuk menyebarkan kode Anda ke penyedia cloud apa pun yang Anda inginkan.

Namun, bahkan Serverless Framework mengharuskan Anda melakukan pengaturan untuk mengawali penyebaran. Pengaturan ini biasanya terdiri dari menentukan penyedia cloud dan menginisialisasi lingkungan CLI Anda dengan penyedia ini. Itu tidak banyak dibandingkan dengan proses manual dalam menyebarkan proyek ke fungsi, tetapi saya pikir itu adalah sesuatu yang akan dibuat lebih lancar di masa mendatang. Penyedia cloud bahkan mungkin mulai membuat diri mereka lebih mudah diakses dan terintegrasi di antara lingkungan yang berbeda. Sekadar untuk memastikan bahwa solusi ini berhasil di berbagai lingkungan, saya juga menguji fungsi tersebut di platform Google dan Azure. Seperti yang ditunjukkan pada Gambar 6.9, keduanya berhasil dan menampilkan pernyataan log yang diharapkan.



Gambar 6.9. Pesan respons yang benar dari Google Cloud Functions membuktikan bahwa fungsi Hello World tidak hanya membuat permintaan yang berhasil, tetapi juga fleksibel dan dapat diskalakan di berbagai platform

Sekarang setelah kami menunjukkan bahwa bukti konsep kami berfungsi di berbagai penyedia, kami perlu menambahkan logika penyedia cloud yang sebenarnya ke dalamnya untuk membuktikan bahwa konsep kami dapat diskalakan. Di bagian berikutnya, kami akan membangunnya untuk menambahkan komponen penyimpanan.

Kode dan Contoh Menggunakan Basis Data

Untuk membuktikan bahwa konsep kami dapat diskalakan dengan tepat dengan logika cloud yang berbeda, kami akan membuat aplikasi yang dipicu oleh peristiwa HTTP dan menyimpan informasi peristiwa pemicu tersebut di penyimpanan Blob. Kami akan terus menggunakan skenario Google Cloud kami tentang kamar hotel dengan tamu untuk data masuk kami.

Hal pertama yang ingin kami lakukan adalah menambahkan file JavaScript penyimpanan di setiap folder penyimpanan penyedia kami. File ini akan berisi fungsi yang mengambil dua parameter, pesan dan ID pesan, dan akan menyimpan pesan di penyimpanan Blob penyedia di bawah nilai ID pesan.

Sekarang, kami dapat menulis kode khusus untuk penyedia cloud untuk logika ini.

```
// dependencies
var AWS = require('aws-sdk');
var S3 = new AWS.S3();
var BucketName = 'poc-cloudagnostic-maddie';

module.exports = {
  saveObject: function(message, messageId) {
    console.log('Message: ' + JSON.stringify(message));

    // upload the message to S3
    S3.putObject({
      Bucket: BucketName,
      Key: messageId,
      Body: JSON.stringify(message)
    }, function (err) {
      if (err) {
        console.error('Error: ' + err);
      } else {
        console.log('Success');
    }
    });
  }
}
```

Kami akan mengembalikan hasilnya ke penyimpanan S3 Blob di AWS. Untuk membuat dan mengunggah string, kami hanya perlu menentukan nama bucket, kunci, dan isi tempat penyimpanan. Saya membuat bucket di dalam konsol dan membuatnya terbuka dan tersedia untuk permintaan masuk. Anda juga perlu memastikan peran fungsi Lambda Anda memiliki akses untuk memperbarui bucket S3.

Kebijakan yang dihasilkan terlihat seperti ini:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": "*"
    }
  ]
}
```

Kami juga ingin membuat layanan penyimpanan untuk Azure. Seperti AWS, Azure juga memerlukan nama penyimpanan Blob untuk bucket, pesan yang ingin Anda simpan, dan ID pesan yang ingin Anda gunakan untuk menyimpan pesan Anda. Seperti yang Anda lihat, kodenya sangat mirip antara kedua penyedia ini. Kami mungkin dapat mengekstrak lebih banyak kode dan membuatnya lebih mudah.

```
// dependencies
var azure = require('azure-storage');
var blobClient = azure.createBlobService();
var containerName = 'poc-cloudagnostic';

  module.exports = {
    saveObject: function(message, messageId) {
      console.log('Message: ' + message);
      message = JSON.stringify(message);
      blobClient.createBlockBlobFromText(containerName, messageId,
      message, function(error,
      result, response) {
        if(error) {
          console.log("Couldn't upload");
          console.error(error);
        } else {
          console.log("Upload successful!");
      }
    })
  }
}
```

Untuk penyimpanan Google Cloud, sekali lagi kita memiliki struktur kode yang sangat mirip dengan penyedia cloud sebelumnya. Perbedaan utamanya di sini adalah kita perlu menentukan projectId dan `keyFilename` tempat fungsi tersebut berada. `Project ID` adalah ID unik yang diberikan Google untuk proyek Anda saat penyiapan. Anda dapat mengambilnya dari dasbor Anda.
`keyFilename` hanya mencari jalur ke dokumen `keyfile.json` Anda. Untuk menemukannya, buka Cloud Shell Anda melalui konsol dan masukkan perintah ini:

```
pwd keyfile.json
```

Seharusnya disimpan di suatu tempat di direktori `home/user/`. Setelah koneksi itu terbentuk, kita tinggal membuat bucket dan mengunggah acara kita ke bucket tersebut.

```
// dependencies
var storage = require('@google-cloud/storage');
var gcs = storage({
  projectId: 'loyal-curve-107623',
  keyFilename: '/home/maddie_stigler'
});
var containerName = 'poc-cloudagnostic';

module.exports = {
  saveObject: function(message, messageId) {
    console.log('Message: ' + message);
    gcs.createBucket(containerName, function(err, bucket) {
      //bucket created
      if(!err) {
        var bucket = gcs.bucket(containerName);
        bucket.upload(JSON.stringify(message), function(err, file) {
        if(!err) {
          console.log("success");
          }
        })
      }
    });
  }
}
```

Seperti yang Anda lihat, banyak eksekusi penyimpanan dalam fungsi penyedia cloud kami sangat mirip. Kami mungkin dapat mengonversi banyak kode ini untuk menggunakan variabel lingkungan sehingga kami tidak mereplikasinya di berbagai proyek. Untuk saat ini, kami akan membiarkannya apa adanya untuk menguji dan memastikan semuanya masih berfungsi dengan baik.

Bagian kode berikutnya yang perlu kami perbarui adalah modul index.handler. Seperti dalam modul Provider, kami akan memerlukan modul Storage dan memanggilnya di badan dengan mengirimkannya peristiwa yang masuk dan string untuk `keyfilename`. Saya hanya akan menggunakan 1 untuk saat ini.

```
// dependencies
var provider = 'aws';
var Provider = require('./storage/' + provider + '/provider');
var Storage = require('./storage/' + provider + '/storage');

exports.handler = function (event, context, callback) {
  console.info(event);
  Provider.printProvider("Hello World");
  Storage.saveObject(event, '1');
}
```

Hal terakhir yang perlu kita lakukan sebelum melanjutkan dan menguji kode kita adalah memperbarui file serverless.yml kita. Kita perlu memastikan bahwa file

`storage.json` kita disertakan dalam zip proyek. Kita juga perlu mengonfigurasi pemicu HTTP.

```
helloWorld:
  handler: index.handler
  events:
    - http:
      method: post
  package:
    include:
  - ../../index.js
  - ../../storage/aws/**
  - node_modules/**
```

Jika Anda ingin menyertakan jalur untuk peristiwa HTTP seperti yang kami lakukan pada beberapa contoh sebelumnya, Anda juga dapat mengonfigurasinya dalam berkas tanpa server. Saya sarankan untuk membuka https://serverless.com/framework/docs/ dan membaca dokumentasi untuk penyedia cloud tertentu yang Anda lihat.

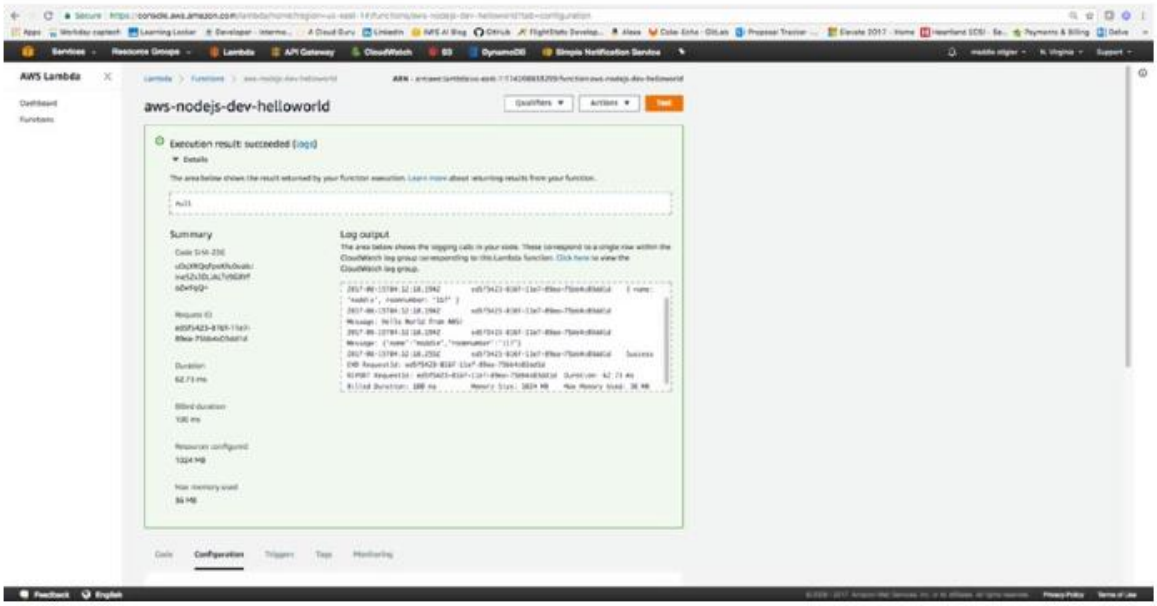
Sekarang kita siap untuk melakukan deploy. Kita dapat mengikuti prosedur yang sama yang kita ikuti sebelumnya dengan menggunakan serverless deployment di setiap folder proyek. Setelah semuanya di-deploy, kita dapat masuk ke konsol dan menyiapkan acara pengujian untuk mencoba dan memicu fungsi kita.

Saya akan menunjukkan contoh dari AWS saja, tetapi fungsi-fungsi ini seharusnya berfungsi untuk semua lingkungan penyedia cloud.

Permintaan POST yang akan saya kirimkan adalah sebagai berikut:

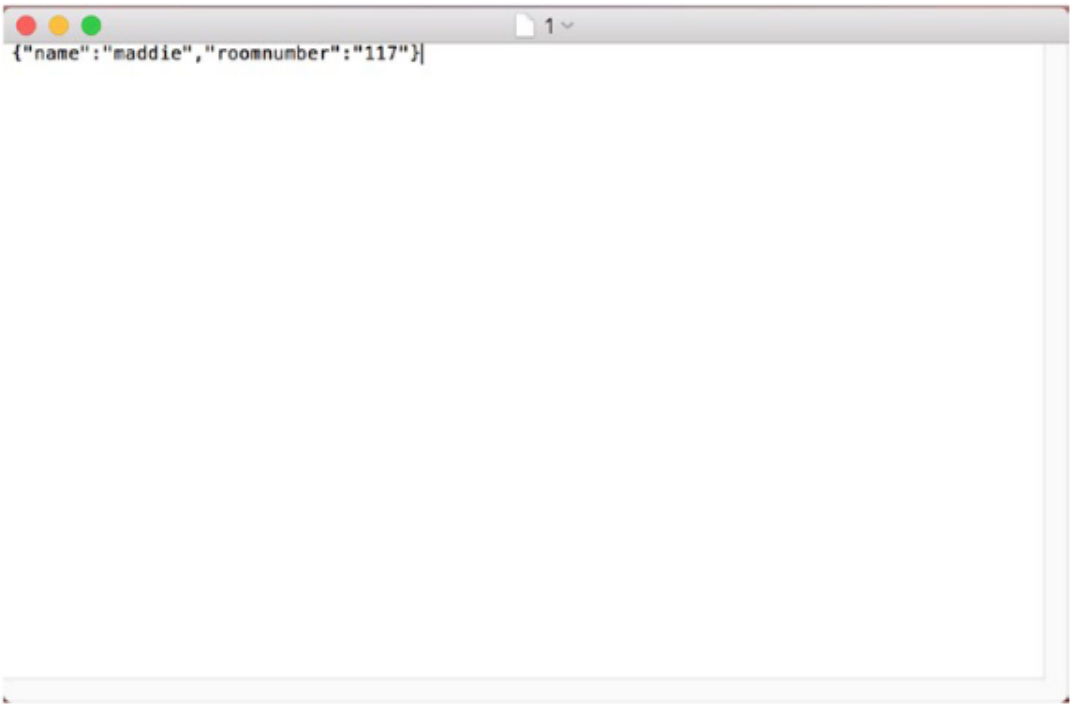
```
{
  "name": "Maddie",
  "roomnumber": "117"
}
```

Saat kami menguji di portal AWS, kami mendapatkan pesan sukses dengan pernyataan pencatatan yang kami sertakan (Gambar 6-10).



Gambar 6.10. Fungsi AWS mengembalikan respons sukses dalam acara pengujian konsol

Sekarang kami perlu memastikan acara kami berhasil masuk ke S3. Navigasi ke bucket yang Anda buat. Anda akan melihat objek 1 baru yang tersimpan di dalamnya (Gambar 6.11). Jika berhasil, selamat! Jika tidak, kembali dan periksa kode Anda dan pastikan semuanya telah ditetapkan dengan benar.



Gambar 6.11. Fungsi AWS mencatat peristiwa masuk ke bucket S3 yang ditentukan

Sebelum kami memberi label keberhasilan pada perluasan bukti konsep kami, kami perlu memeriksa apakah dua peristiwa penyimpanan lainnya terjadi. Di Azure, data peristiwa

Anda akan disimpan di penyimpanan Blob. Anda akan melihat Blob di kontainer yang Anda buat. Di Google, data peristiwa Anda akan disimpan di Google Storage. Anda akan melihat bucket yang Anda buat dengan data peristiwa yang tersimpan di dalamnya. Jika Anda melihat semua ini, selamat! Kami telah resmi membuat solusi agnostik. Memang, ini adalah aplikasi yang sangat sederhana, tetapi idenya dapat diskalakan di berbagai layanan.

Meningkatkan Fungsi Serverless

Peningkatan dengan menggunakan variabel lingkungan dan penerapan model Gunakan variabel lingkungan untuk menyimpan variabel yang kami ulangi dalam logika penyedia kami:

- Gunakan variabel Lingkungan baik dalam Serverless Framework atau disimpan langsung di lingkungan penyedia:

```
# Tentukan variabel lingkungan fungsi di sini
# environment:
# variable2: value2
```

Variabel lingkungan akan menjaga bucket dan jalur kami tetap konsisten di seluruh penyedia. Variabel tersebut juga akan memastikan bahwa kami tidak terlalu sering mengulang kode di seluruh logika penyedia.

Gunakan model untuk memberlakukan permintaan masuk:

1. Siapkan otorisasi pada titik akhir dan pastikan kunci otorisasi ini disimpan dengan benar baik sebagai variabel lingkungan atau di konsol penyedia.
2. Gunakan model JavaScript untuk menyusun data masuk:

```
export interface requestModel {
  name: string,
  roomnumber: string
}
```

3. Kita juga dapat menerapkan pemeriksaan metode permintaan seperti yang kita lakukan pada contoh Google Cloud.

Kode untuk kedua peningkatan pada proyek ini dapat ditemukan di sini: https://github.com/mgstigler/Serverless/tree/master/CloudAgnostic/

6.5 KESIMPULAN

Dalam bab terakhir ini, kita akan membahas cara-cara untuk menjadikan kode kita tidak bergantung pada cloud. Dalam hal ini, kita dapat mengakses layanan AWS, Azure, dan Google Cloud dalam satu fungsi. Jika pemilik bisnis memutuskan untuk memilih satu penyedia cloud daripada yang lain, hal itu akan tercakup dalam solusi ini. Anda mempelajari cara mengintegrasikan berbagai layanan cloud ke dalam berbagai fungsi cloud dan membuatnya mudah diakses dalam lingkungannya sendiri. Pada titik ini, Anda seharusnya dapat membedakan antara fungsi penyedia mana yang dipicu dan bagaimana mereka menanganinya dalam kondisi ini. Bab ini penting karena membahas masalah klien nyata dan mengusulkan solusi untuk masalah tersebut.

DAFTAR PUSTAKA

Adzic, G. (2017). *Serverless computing: Economic and architectural impact*. Journal of Cloud Computing, 6(1), 1-15. https://doi.org/10.1186/s13677-017-0093-9

Ali, S., & Daoud, M. (2019). *Serverless computing for cloud-native services*. Springer.

Amazon Web Services. (2019). *AWS Lambda: What is serverless computing?* https://aws.amazon.com/lambda/

Andrikopoulos, V., Zarras, A., & Pimentel, M. (2019). *Serverless computing: State of the art and future research challenges*. ACM Computing Surveys (CSUR), 52(2), 1-34. https://doi.org/10.1145/3301212

Bakshi, S., & Singh, R. (2020). *The principles of serverless computing for distributed applications*. 2020 IEEE International Conference on Cloud Computing (CLOUD), 1-9. https://doi.org/10.1109/CLOUD.2020.00013

Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2017). *Microservices architecture enables DevOps: Challenges, advantages, and research opportunities*. Future Generation Computer Systems, 74, 1-17. https://doi.org/10.1016/j.future.2017.03.056

Behzad, A., & Altinel, A. (2020). *Serverless computing: From a research perspective*. 2020 International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), 143-147. https://doi.org/10.1109/ICCCBDA49360.2020.9090529

Bojan, J., & Torkashvand, M. (2018). *Serverless architecture: A practical guide to building systems with AWS Lambda*. O'Reilly Media.

Chansrichawla, P. (2018). *Serverless computing: A new paradigm for cloud applications*. Wiley. https://doi.org/10.1002/9781119489079

Chen, Z., & He, H. (2018). *Serverless computing: Programming model, framework, and challenges*. Journal of Computer Science and Technology, 33(6), 1219-1239. https://doi.org/10.1007/s11390-018-1864-7

Choi, M., & Kim, B. (2020). *Serverless architecture: A solution to cloud-native applications*. Springer.

Cloud Academy. (2019). *Serverless computing: Introduction and architecture*. https://cloudacademy.com

Darwish, A., & Mahdy, R. (2020). *Serverless computing in modern cloud platforms*. 2020 International Conference on Cloud Computing and Virtualization (CCV), 44-49. https://doi.org/10.1109/CCV52052.2020.00014

Dellinger, J. (2020). *Building serverless applications with AWS Lambda*. Packt Publishing.

Fischer, M., & Schill, A. (2020). *Serverless computing: A guide to designing and building cloud-native applications*. O'Reilly Media.

Gannon, D. (2018). *The rise of serverless computing and its impact on software architecture*. ACM Transactions on Software Engineering and Methodology, 27(3), 1-18. https://doi.org/10.1145/3214672

Gonzalo, G., & Garcia, J. (2019). *Serverless computing in practice: Using AWS Lambda for cloud computing*. Springer.

Hohpe, G. (2017). *Serverless computing: Key considerations for building and deploying applications*. ACM Queue, 15(5), 28-34.

Huang, Y., & Xu, Z. (2020). *Serverless computing: Concepts, applications, and challenges*. 2020 International Conference on Cloud Computing and Virtualization (CCV), 34-39. https://doi.org/10.1109/CCV52052.2020.00013

Javed, M., & Alhashmi, Z. (2020). *Exploring serverless architectures: Fundamentals and practical implications*. Journal of Cloud Computing, 9(1), 1-17. https://doi.org/10.1186/s13677-020-00209-1

Kaur, R., & Sharma, A. (2021). *Hands-on serverless computing with AWS Lambda*. Packt Publishing.

Khosravi, A., & Amirian, M. (2019). *Serverless computing for microservices: Architecture and case studies*. Springer.

Kolesnikov, L. (2019). *A comprehensive guide to serverless computing*. Packt Publishing.

Larkin, J. (2020). *Getting started with serverless computing and AWS Lambda*. Apress.

Lee, D., & Lee, J. (2020). *Exploring serverless architectures and their applications*. Springer.

Lemos, A., & Silva, R. (2020). *Cloud computing and serverless computing: Building applications in the cloud*. Springer.

Liu, L., & Zhao, X. (2019). *Serverless computing for microservices: Principles and best practices*. IEEE Access, 7, 150699-150711. https://doi.org/10.1109/ACCESS.2019.2947124

Liu, X., & Li, B. (2018). *Serverless computing for distributed systems: Challenges and opportunities*. Future Internet, 10(3), 1-17. https://doi.org/10.3390/fi10030027

Liu, X., & Li, F. (2018). *Serverless computing for cloud applications: Overview and future challenges*. Future Generation Computer Systems, 89, 467-481. https://doi.org/10.1016/j.future.2018.07.033

Liu, Y., & Zhang, W. (2020). *Serverless computing: Applications, techniques, and research challenges*. Journal of Cloud Computing, 9(3), 1-18. https://doi.org/10.1186/s13677-020-00220-7

Mohr, M., & Schuck, R. (2019). *Serverless computing: From infrastructure to development paradigm*. Computer Science Review, 29, 17-29. https://doi.org/10.1016/j.cosrev.2019.06.001

Moussawi, H., & Helal, M. (2020). *Serverless computing: A critical review*. International Journal of Computer Applications, 175(5), 1-9. https://doi.org/10.5120/ijca2020919197

Nouri, M., & Kargar, M. (2019). *Serverless computing: Current status, challenges, and future directions*. 2019 IEEE International Conference on Cloud Computing (CLOUD), 284-291. https://doi.org/10.1109/CLOUD.2019.00051

O'Connor, M. (2021). *Serverless computing: Building scalable, efficient cloud-native applications*. Packt Publishing.

Pahl, C., & Xie, M. (2020). *Designing serverless systems for microservices and cloud-native applications*. Springer.

Pahl, C., & Xie, X. (2020). *Serverless computing: Design and implementation*. Springer. https://doi.org/10.1007/978-3-030-23624-1

Patil, S., & Shah, S. (2020). *Serverless computing: A guide to cloud-based application development*. Springer.

Pereira, M., & Zambon, R. (2018). *The future of serverless computing: Use cases and challenges*. 2018 IEEE International Conference on Cloud Computing (CLOUD), 340-347. https://doi.org/10.1109/CLOUD.2018.00057

Portillo, A., & Martin, J. (2019). *Serverless computing for cloud-native applications*. Wiley.

Radenkovic, M., & Zhang, S. (2020). *Serverless architecture and its impact on cloud services*. Springer.

Richards, M., & Ford, T. (2020). *Serverless architecture: A hands-on guide to AWS Lambda*. O'Reilly Media.

Rosenthal, S., & Ochs, R. (2018). *Serverless computing: Cost-effective solutions for cloud environments*. Computer Science Review, 28, 49-58. https://doi.org/10.1016/j.cosrev.2018.07.001

Santamaria, R. (2019). *Serverless computing: Patterns and best practices*. Addison-Wesley.

Schinagl, S., & Kirsch, M. (2020). *Serverless computing: Managing cloud-based systems without servers*. Wiley.

Shi, H., & Liu, J. (2020). *A survey on serverless computing: Benefits, challenges, and future research*. Future Internet, 12(10), 1-19. https://doi.org/10.3390/fi12100179

Soni, V., & Shah, K. (2020). *Serverless computing: A revolutionary approach to cloud computing*. Springer.

Stojanovic, J., & Malobabic, S. (2020). *Building scalable serverless applications*. Wiley.

Tang, Z., & Yang, X. (2021). *Serverless computing for cloud applications: Design, implementation, and challenges*. IEEE Access, 9, 34356-34375. https://doi.org/10.1109/ACCESS.2021.3050321

Tchouankem, F., & Poirier, B. (2019). *Serverless computing: A new approach to cloud application development*. 2019 IEEE International Conference on Cloud Computing (CLOUD), 19-28. https://doi.org/10.1109/CLOUD.2019.00031

Thomas, S., & Lee, H. (2021). *Mastering serverless computing with AWS Lambda*. Apress.

Torgo, D., & Ayhan, M. (2020). *Serverless computing in cloud applications: Challenges and use cases*. Journal of Computer and System Sciences, 98, 55-68. https://doi.org/10.1016/j.jcss.2019.09.003

Wang, S., & Guo, B. (2019). *Serverless computing in practice: Leveraging the power of cloud services*. Springer.

Wu, X., & Zhou, T. (2020). *Serverless computing for microservice applications: A practical approach*. Springer.

Zekri, M., & Abderrahim, H. (2020). *Exploring serverless architectures for cloud-based applications*. Springer.

Zhan, X., & Liu, Y. (2020). *A comprehensive guide to serverless computing architectures*. Springer.

Zhang, J., & Wu, Z. (2019). *Exploring serverless computing for cloud-native applications*. Journal of Cloud Computing, 8(1), 1-18. https://doi.org/10.1186/s13677-019-0165-2

Zhang, T., & Zhang, Y. (2019). *Serverless computing in cloud-native systems: A new trend in cloud computing*. International Journal of Computer Applications, 178(9), 16-23. https://doi.org/10.5120/ijca2019918531

Zhang, W., & Chen, L. (2019). *Serverless computing: Insights, architectures, and applications*. ACM Computing Surveys (CSUR), 52(5), 1-32. https://doi.org/10.1145/3293664

Zhao, Q., & He, Y. (2019). *Serverless computing: A survey of research and trends*. IEEE Access, 7, 149563-149586. https://doi.org/10.1109/ACCESS.2019.2944851

Zong, X., & Li, Y. (2020). *Serverless computing and cloud application development: Best practices and patterns*. Wiley.

KOMPUTASI TANPA SERVER

(Serverless Computing)

Dr. Agus Wibowo, M.Kom, M.Si, MM.

BIO DATA PENULIS

Penulis memiliki berbagai disiplin ilmu yang diperoleh dari Universitas Diponegoro (UNDIP) Semarang. dan dari Universitas Kristen Satya Wacana (UKSW) Salatiga. Disiplin ilmu itu antara lain teknik elektro, komputer, manajemen dan ilmu sosiologi. Penulis memiliki pengalaman kerja pada industri elektronik dan sertifikasi keahlian dalam bidang Jaringan Internet, Telekomunikasi, Artificial Intelligence, Internet Of Things (IoT), Augmented Reality (AR), Technopreneurship, Internet Marketing dan bidang pengolahan dan analisa data (komputer statistik).

Penulis adalah pendiri dari Universitas Sains dan Teknologi Komputer (Universitas STEKOM) dan juga seorang dosen yang memiliki Jabatan Fungsional Akademik Lektor Kepala (Associate Professor) yang telah menghasilkan puluhan Buku Ajar ber ISBN, HAKI dari beberapa karya cipta dan Hak Paten pada produk IPTEK. Sejak tahun 2023 penulis tercatat sebagai Dosen luar biasa di Fakultas Ekonomi & Bisnis (FEB) Universitas Diponegoro Semarang. Penulis juga terlibat dalam berbagai organisasi profesi dan industri yang terkait dengan dunia usaha dan industri, khususnya dalam pengembangan sumber daya manusia yang unggul untuk memenuhi kebutuhan dunia kerja secara nyata.

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 978-623-8642-62-5 (PDF)
9 786238 642625