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

Desain dan Implementasi

API untuk Cloud Computing dengan Azure dan AWS

YAYASAN PRIMA AGUS TEKNIK

Desain dan Implementasi

API untuk Cloud Computing dengan Azure dan AWS

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

BIODATA PENULIS

Dr. Joseph Teguh Santoso, M.Kom adalah pemimpin yang visioner dan praktisi industri berpengalaman, yang menjabat sebagai Rektor Universitas Sains dan Teknologi Komputer (Universitas STEKOM), salah satu universitas terkemuka di Jawa Tengah, Indonesia. Dengan pengalaman lebih dari 13 tahun di dunia bisnis dan praktisi industri di China, beliau membawa perspektif global dan inovasi yang signifikan ke dalam dunia akademis.Sebagai seorang entrepreneur, penulis adalah pencipta TopLoker.com, sebuah platform inovatif yang merevolusi cara mencari dan menawarkan pekerjaan. TopLoker.com adalah portal lowongan bursa kerja terbesar di Indonesia, khusus untuk pendidikan SMA/SMK sederajat. TopLoker.com telah mendapatkan penghargaan sebagai juara 1 Startup4industry 2022 oleh Kementerian Perindustrian Republik Indonesia. Kontribusi Dr. Joseph dalam menyediakan akses pekerjaan yang luas bagi lulusan SMA/SMK telah membantu banyak individu menemukan peluang kerja yang sesuai dengan keahlian mereka. Selain itu, Dr. Joseph Teguh Santoso, M.Kom adalah pendiri dari dua organisasi yaitu (1) organisasi guru/pendidik PTIC (Perkumpulan Teacherpreneur Indonesia Cerdas) yang bertujuan untuk meningkatkan kualitas pendidikan dan kesejahteraan guru/pendidik dengan wawasan entrepreneurship, serta (2) organisasi industri PERKIVI (Perkumpulan Komunitas Industri dan Vokasi Indonesia) yang berfokus pada pengembangan link and match antara industri dan dunia pendidikan. Sebagai Rektor, Dr. Joseph Teguh Santoso, M.Kom memiliki kepemimpinan yang berorientasi pada hasil, dan berkomitmen untuk mendorong kemajuan Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Saat ini Universitas STEKOM telah mengalami transformasi positif dalam peningkatan kualitas pendidikan, perluasan fasilitas, serta penguatan kemitraan Perguruan Tinggi Nasional dan Internasional. Beliau memprioritaskan pengembangan sumber daya manusia dan penelitian, serta memastikan bahwa universitas berada di garis depan dalam inovasi dan teknologi untuk mencapai tujuan akhir, yaitu lulusan yang mampu bekerja dan sukses setelah lulus.Dr. Joseph Teguh Santoso, M.Kom sering diundang sebagai pembicara di berbagai konferensi nasional maupun internasional dan telah menerima berbagai penghargaan atas dedikasinya dalam bidang pendidikan, industri, dan kewirausahaan.

YAYASAN PRIMA AGUS TEKNIK

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

Desain dan Implementasi
API untuk Cloud Computing dengan Azure dan AWS

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

ISBN :

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

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

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

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

Anggota IKAPI No: 279 / ALB / JTE / 2023

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

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

KATA PENGANTAR

Puji dan syukur kami panjatkan ke hadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, sehingga buku yang berjudul "***Desain dan Implementasi API untuk Cloud Computing dengan Azure dan AWS***" dapat diselesaikan dengan baik. Buku ini ditulis dengan tujuan untuk memberikan pemahaman mendalam mengenai konsep-konsep dasar dalam pengembangan API, serta memperkenalkan cara-cara terbaik untuk merancang dan mengelola API di platform cloud terkemuka, yaitu Microsoft Azure dan Amazon Web Services (AWS).

Dalam era digital yang semakin maju, pengembangan aplikasi dan integrasi sistem menjadi sangat penting. Salah satu komponen krusial yang mendukung hal tersebut adalah API (*Application Programming Interface).* API memungkinkan aplikasi untuk saling berkomunikasi dan bertukar data dengan cara yang efisien dan terstruktur. Namun, untuk mengoptimalkan kinerja dan skalabilitas API, pengembang perlu memahami bagaimana mendesain dan mengembangkan API dengan memperhatikan aspek arsitektur yang tepat, serta memanfaatkan platform cloud seperti Azure dan AWS.

Azure dan AWS adalah dua penyedia layanan cloud terkemuka yang menawarkan berbagai solusi untuk pengembangan dan pengelolaan API. Keduanya menyediakan alat, layanan, dan infrastruktur yang memungkinkan pengembang untuk membuat API yang dapat diandalkan, aman, dan skalabel. Dalam konteks ini, memahami perbedaan serta kelebihan masing-masing platform sangat penting bagi para pengembang untuk memilih solusi yang paling sesuai dengan kebutuhan bisnis dan teknologi yang ada.

Buku ini terdiri dari tujuh bab yang menyajikan berbagai aspek terkait API. Bab 1 memberikan pengantar praktis mengenai API, termasuk sistem aliran data dan operasinya, serta penerapan API di sektor publik. Bab 2 membahas strategi dan arsitektur API, dengan fokus pada manajemen dan kasus penggunaan yang relevan. Bab 3 mengupas pertimbangan dalam pengembangan API serta standar dan versi yang perlu diperhatikan.

Selanjutnya, Bab 4 menjelaskan tentang gerbang API, termasuk konfigurasi dan produk di Azure API Management. Bab 5 berfokus pada keamanan API, membahas autentikasi, otorisasi, serta penerapan token JWT. Bab 6 mengupas konsep serverless dalam pengembangan API, dengan contoh penerapan di Azure dan AWS. Terakhir, Bab 7 menyajikan desain dan pengembangan praktis API dalam konteks layanan mikro dan integrasi perusahaan.

Saya ingin mengucapkan terima kasih kepada semua pihak yang telah berkontribusi dalam penyelesaian buku ini. Tanpa dukungan mereka, buku ini tidak akan terwujud. Saya juga berharap buku ini dapat bermanfaat bagi pembaca dan menjadi referensi yang berguna dalam memahami dan menerapkan API di berbagai bidang. Kritik dan saran dari pembaca sangat saya harapkan untuk perbaikan di masa mendatang. Selamat Membaca.

Semarang, Januari 2025
Penulis

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

DAFTAR ISI

BAB 1
PENGANTAR PRAKTIS TENTANG API

Data adalah Tuhan: permintaan data yang terus meningkat merupakan salah satu faktor mendasar yang memengaruhi disrupsi banyak teknologi komunikasi dan integrasi. Preferensi untuk data daripada operasi merupakan tantangan bagi pengembangan perangkat lunak modern, mengatur dan mengelola integrasi dan aliran data telah menjadi faktor kunci keberhasilan dalam solusi perangkat lunak modern. Permintaan untuk aplikasi perangkat lunak yang berpusat pada data dan lanskap pengembangan yang terus berkembang yang lebih menyukai penerapan alat dan layanan yang ada untuk mencapai kecepatan dan fleksibilitas daripada mengembangkan semuanya dari awal, membuat teknologi integrasi tradisional menjadi usang.

API (*Application Programming Interface*) adalah keadaan evolusi terkini dalam teknologi integrasi dan kita dapat melihat evolusi bergerak menuju integrasi sebagai ruang bahasa. Selain itu, arsitektur perangkat lunak modern seperti layanan mikro dan tanpa server lebih menyukai API daripada teknologi integrasi tradisional dan API membantu arsitektur tersebut untuk berkembang; manfaatnya saling menguntungkan dan saling melengkapi dalam kedua hal. Secara teori, setiap antarmuka yang dapat diprogram dapat disebut sebagai API.

Meskipun istilah Antarmuka Pemrograman Aplikasi (API) bukanlah hal baru dan memiliki banyak bentuk implementasi, penggunaan istilah saat ini umumnya mengacu pada layanan RESTful berbasis HTTP. Bisnis memaparkan data dan operasi melalui API karena berbagai alasan, seperti mencapai kelincahan bisnis, memonetisasi data dan operasi bisnis, integrasi, memungkinkan inovasi, memungkinkan ekosistem bisnis, mematuhi persyaratan peraturan, dan lain-lain. Bab ini memberikan pengantar tentang API dengan penjelasan praktis, dan membangun diskusi menuju topik umum seperti ekonomi API dan bagaimana API digunakan di sektor publik.

1.1 PENGANTAR PRAKTIS API

Industri perangkat lunak sudah dibanjiri dengan kata kunci, dan sebagian besarnya dibuat dengan tujuan untuk memuaskan pemangku kepentingan teknis dan bisnis. Karena kewajiban audiens ganda ini, sebagian besar istilah ini tidak ditujukan kepada pemangku kepentingan bisnis atau teknis dengan benar. Pendekatan praktis buku ini mengeksplorasi makna dan membangun pemahaman umum tentang API untuk konteks pembaca. Ini bukan pencarian definisi baru, tetapi upaya untuk membantu pembaca memahami konteksnya. API memiliki kepribadian ganda: satu didasarkan pada konstruksi bahasa atau dalam bentuk pustaka/kerangka kerja, dan yang kedua adalah, sebagai sistem yang memaparkan data dan operasi.

Konstruksi Bahasa yang Dapat Diprogram

Pertama, mari kita lihat "kepribadian" pertama. Sebagai pengembang, kita menulis kode menggunakan konstruksi bahasa dan memaparkan perilaku. Dalam bahasa

Pemrograman Berorientasi Objek (OOP) yang umum, hal ini dicapai dengan kelas yang mengimplementasikan antarmuka. Umumnya, antarmuka ini diimplementasikan, sehingga parameter disertakan dalam tanda tangan metode hanya saat pemanggil diminta untuk menyediakannya. Jika parameter yang diperlukan dapat diperoleh dari konteks eksekusi, maka kami mencoba menyediakan data lintas bidang tersebut untuk berbagai lapisan melalui kelas bersama, daripada mendeklarasikannya secara eksplisit dalam tanda tangan metode. Contoh di bawah ini menunjukkannya. Antarmuka yang diekspos:

```
internal interface IRegisrationService
{
   Task<int> RegisterBookAsync(Book book);
}

// One of the implementations
public class RegistrationService : IRegisrationService
{
   public async Task<int> RegisterBookAsync(Book book)
   {
      book.UserId = SessionProvider.GetUserId();
      // rest of the logic
   }
}
```

Di sini, `UserId` tidak diparameterisasi dalam konteks eksekusi saat ini, tetapi implementasinya berfungsi karena mengetahui dari mana mendapatkan `UserId`. Ini bagus, tetapi implementasi antarmuka ini terbatas karena implementasi layanan sangat erat kaitannya dengan konteks eksekusi saat ini. Dan pihak lain yang mengimplementasikan antarmuka ini berharap memperoleh UserId secara internal.

Jika, di kemudian hari, klien atau rakitan yang berbeda ingin menggunakan `RegistrationService` dan ingin menyediakan `UserId` dari sumber yang berbeda, mereka akan gagal atau perlu menanganinya dalam implementasi antarmuka, yang mengakibatkan implementasi yang berbeda untuk setiap sumber `UserId` atau logika rumit lainnya. Agar antarmuka dapat digunakan oleh pemanggil eksternal, parameterisasi penting dalam tanda tangan metode itu sendiri, seperti yang ditunjukkan di sini. Hal ini membuat implementasi dan

```
public interface IRegisrationService
{
   Task<int> RegisterBookAsync(Book book, string UserId);
}
```

Membuat antarmuka dengan mempertimbangkan pemanggil eksternal di luar konteks eksekusi, membantu mencapai penggabungan longgar di berbagai lapisan kode. Perubahan ini adalah realisasi tingkat pertama dari antarmuka yang diekspos dalam konteks pengembangan. Antarmuka menjadi dipublikasikan dan diekspos ke pemanggil eksternal selama transisi ini, sehingga dapat dikonsumsi oleh pihak eksternal merupakan karakteristik mendasar dari

antarmuka yang dapat dikonsumsi. Antarmuka dan implementasi ini dapat dipublikasikan sebagai paket. Paket dirujuk dalam konteks eksekusi lain, memastikan kegunaan bagi pihak eksternal, dan dibundel dengan alat pengembangan, komponen, kelas dasar, dokumentasi, dan contoh implementasi lainnya.

Bundel ini umumnya dikenal sebagai Kit Pengembangan Perangkat Lunak (SDK). SDK (*Software Development Kit*) adalah API dengan fitur dan alat tambahan yang menargetkan pengembang yang lebih berpengalaman. Paket dan SDK telah membantu penggunaan kembali kode dan berbagi kode. Perubahan ini memengaruhi cara sistem diprogram; pengembang tidak diharuskan menulis semua kode untuk sistem yang mereka kembangkan. Kerangka kerja dan pustaka muncul, dan sekarang kita semua sudah familier dengan pengelola paket seperti NuGet, NPM, dan masih banyak lagi. Sekarang, kita tidak dapat membayangkan skenario pengembangan tanpa paket.

1.2 SISTEM ALIRAN DATA & OPERASI

Jenis API kedua memungkinkan aliran data dan operasi antara sistem yang berbeda. Kebutuhan akan arsitektur berbasis layanan yang terdistribusi menjadi hal yang sepele karena sistem yang berbeda dan permintaan integrasi yang terus meningkat. Dalam model ini, kontrak layanan (perjanjian data) penting untuk komunikasi antara layanan yang berbeda. Awalnya, implementasi antarmuka khusus bahasa mengambil alih panggung. *Remote Procedure Call* (RPC) adalah level implementasi pertama untuk mengubah tipe API pertama menjadi yang kedua.

Implementasi RPC sangat bergantung pada bahasa dan waktu proses. Karena keterbatasan ini, sistem yang dikembangkan menggunakan bahasa yang berbeda tidak pernah memiliki keberhasilan yang signifikan dalam arsitektur berbasis layanan menggunakan RPC. Tipe API kedua membutuhkan pendekatan yang berbeda. Kebutuhan akan pertukaran pesan yang tidak bergantung pada bahasa dan implementasi antara layanan yang berbeda diakui. Hal ini membuka jalan bagi istilah industri yang sangat digunakan dan sangat membingungkan Arsitektur Berorientasi Layanan *Service-Oriented Architecture* (SOA).

Sebagian besar, implementasi SOA memiliki beberapa bentuk teknologi antrean atau bus layanan dan kontrak layanan diimplementasikan dalam (*Extensible Markup Language*) XML/JSON. Dalam dunia SOA, XML memimpin dengan berbagai jenis implementasi, seperti SOAP. Kemudian seiring berjalannya waktu, munculnya web dan HTTP (*HyperText Transfer Protocol*) membuka jalan bagi layanan web. Layanan web muncul karena meningkatnya kebutuhan data di dunia Internet. Dibandingkan dengan implementasi SOA tradisional, layanan web ramah Internet, tetapi karena standar dan praktik implementasi yang sangat banyak yang dipinjam dari SOA, layanan tersebut tidak cukup gesit untuk memenuhi kebutuhan Internet yang terus berkembang.

Akhirnya, teknologi pertukaran data yang lebih ramah Internet atau dengan kata lain, lebih asli Internet diusulkan. Layanan RESTful didasarkan pada HTTP dan kata kerja HTTP, sementara JSON muncul sebagai format pertukaran data yang lebih ringan dan semakin populer. Jenis API kedua telah berkembang sebagai layanan RESTful berbasis HTTP. Jadi

sekarang, ketika kita menggunakan istilah API, istilah tersebut paling sering merujuk pada layanan RESTful berbasis HTTP dengan kontrak layanan berbasis JSON. Namun seperti yang telah kita bahas, API hadir dalam dua jenis berbeda: dalam bentuk konstruksi bahasa yang dapat diprogram dan layanan RESTful yang dapat diprogram.

Kesamaan antara keduanya adalah kemampuan untuk digunakan oleh pihak eksternal. Dalam konteks buku ini, istilah API merujuk pada jenis kedua, yaitu antarmuka yang dapat diprogram berbasis HTTP yang ramah internet. Sekarang kita memahami apa itu API dan dua bentuk berbeda yang dapat diambilnya di dunia teknik, serta makna API dalam konteks industri saat ini. Di bagian selanjutnya dari bab ini, kita akan fokus pada ekonomi API dan bagaimana API digunakan di sektor publik.

Ekonomi API

Perubahan teknologi yang cepat dan tren yang berkembang, seperti *Internet of Things* (IoT), cloud, kolaborasi layanan, AR (*Augmented Reality*), VR (*Virtual Reality*), dan MR (*Mixed Reality*), dan banyak kata kunci lainnya, sangat memengaruhi hampir setiap bisnis untuk mencari model baru yang memenuhi tekanan industri dan memastikan kelangsungan hidup. Tren ini membantu penyebaran API dan ekonomi API. Setiap penggunaan API untuk keuntungan ekonomi dapat digeneralisasikan sebagai "ekonomi API." Bisnis memiliki kemampuan dan operasi data yang unik, dan API memaparkan data dan operasi untuk menciptakan peluang baru.

Bisnis dengan data dan operasi berharga yang mahal untuk dibuat menjualnya untuk dikonsumsi melalui API. Ini adalah penjualan langsung. Contoh yang bagus adalah layanan kognitif yang menjual AI sebagai layanan melalui API. Microsoft & Google memiliki beragam layanan tersebut. Bisnis telah meningkatkan fokus mereka untuk menciptakan pengalaman pelanggan baru dan menemukan peluang baru untuk melayani pelanggan dengan lebih baik dan lebih efisien. API membantu melakukan hal ini, tidak hanya dengan mengekspos data dan operasi, tetapi juga dengan memungkinkan inovasi dan pemikiran desain dari luar organisasi.

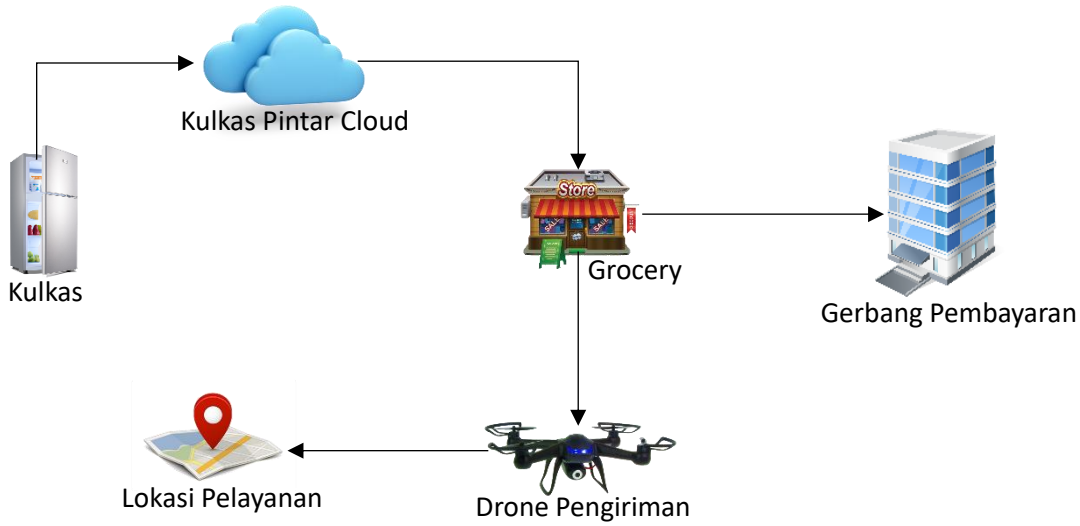
Misalnya: pengecer mode mengekspos katalog produk dan operasi pencariannya melalui API, yang memicu pengembang eksternal untuk membuat aplikasi dengan kemampuan seperti pencarian berbasis lokasi atau berbasis gambar. Meskipun pengecer mode menyediakan API secara gratis kepada pengembang, ia memperoleh inovasi dan pemikiran desain, yang mungkin tidak sepenuhnya dimanfaatkan oleh organisasi internal. Inovasi dan pemikiran desain eksternal ini adalah pola umum yang dapat kita amati dalam banyak aplikasi mashup. Karena pengembang eksternal menggunakan API pengecer mode dan mashup dengan pencarian berbasis lokasi dan gambar lain yang tersedia dari penyedia lain untuk menciptakan pengalaman pelanggan baru, ini adalah situasi yang saling menguntungkan bagi semua pihak yang terlibat.

Cara lain untuk menciptakan pengalaman pelanggan yang kaya adalah dengan berfokus pada pengalaman tanpa hambatan dengan memungkinkan ekosistem yang terdiri dari berbagai sistem. Misalnya, sebagian besar produsen gadget yang dapat dikenakan memiliki integrasi dengan layanan kesehatan. Jumlah langkah, jarak, detak jantung, dan detail lainnya dari pengguna biasanya diintegrasikan dengan penyedia melalui API, yang memungkinkan

ekosistem pengalaman pelanggan yang lancar. Pertimbangkan contoh di bawah ini, di mana lemari es pintar secara otomatis memesan lebih banyak susu saat habis. Gambar 1-1 menunjukkan layanan dasar, pemangku kepentingan yang berbeda, dan integrasi tingkat tinggi di antara mereka.

Pabrikan lemari es telah mengekspos API (gateway IoT), yang menerima data sensor dari lemari es. Asumsikan beberapa sensor susu ajaib mampu mengukur jumlah susu di lemari es, dan data sensor dikumpulkan dan diproses di cloud. Jika level susu berada di bawah ambang batas yang dikonfigurasi, alat tersebut terhubung ke API toko kelontong. API ini menerima pesanan dan mengonfirmasi pembayaran dengan menghubungi gateway pembayaran, yang merupakan API lain. Kemudian, API tersebut mengirimkan pesan ke layanan pengiriman berbasis drone untuk mengambil susu dari toko kelontong dan mengirimkannya ke pelanggan.

Layanan pengiriman drone menggunakan API lokasi dan cuaca untuk menyelesaikan pengiriman. Dalam skenario kehidupan nyata, akan ada lebih banyak API yang terlibat dari berbagai pihak, tetapi ini adalah contoh yang layak untuk memahami model bisnis yang menggunakan API dari berbagai pihak.



Gambar 1.1 Skenario Kulkas Pintar

Bagi produsen kulkas, memiliki cloud pintar dan memproses data sensor IoT memungkinkan pengalaman pelanggan yang bernilai tambah. Bagi toko kelontong, menerima pesanan melalui API merupakan peluang untuk e-commerce. Produsen kulkas dapat mengaktifkan ekosistem dengan beberapa toko kelontong, menempatkan pesanan yang meningkatkan bisnis, dan dapat mengenakan biaya kepada toko untuk manfaat ini. Gateway pembayaran menggunakan model bisnis berbasis komisi di sekitar layanan pengiriman API, mengekspos API-nya untuk mengelola pesanan pengiriman ini adalah e-commerce berbasis layanan.

Ia berbicara dengan API layanan utilitas lain untuk menyelesaikan tugasnya, seperti API lokasi dan cuaca, yang gratis atau berbayar untuk penggunaan komersial. Jadi penyedia data lokasi dan cuaca menjual data mereka secara langsung melalui API. Dalam kasus di atas, kita

melihat berbagai tujuan API untuk berbagai pihak dalam berbagai model bisnis. Beberapa melakukan penjualan langsung, beberapa mampu menyediakan pengalaman pelanggan yang lebih kaya dan pemberdayaan ekosistem, beberapa mengekspos operasi mereka berdasarkan komisi atau e-commerce. Semua model ini adalah versi ekonomi API yang berbeda, dan mewakili berbagai cara API dapat menghasilkan manfaat ekonomi bagi organisasi.

1.3 API DI SEKTOR PUBLIK

Memberdayakan dan memberdayakan pemerintah melalui strategi integrasi dan data merupakan tujuan utama API di sektor publik. Banyak pemerintah telah bekerja secara aktif untuk menyediakan pengalaman yang lebih baik bagi warga negara, bisnis, dan pemerintah lain melalui strategi digitalisasi dan API. Ada tiga model layanan pemerintah:

- G2C: Pemerintah ke Warga
- G2B: Pemerintah ke Bisnis
- G2G: Pemerintah ke Pemerintah

G2C: Pemerintah ke Warga

Model ini melayani warga dengan menyediakan data dan efisiensi operasional mesin pemerintah melalui API. Dengan mengekspos API, pengembang dan kantor negara dapat membuat model dan aplikasi integrasi baru, yang memungkinkan otomatisasi dan digitalisasi berbagai layanan. Pemerintah India telah memperkenalkan nomor identifikasi digital baru bagi penduduk melalui Unique Identification Authority of India. Sasaran utama proyek ini, yang dikenal sebagai Aadhaar, adalah untuk memungkinkan identifikasi digital yang unik, terintegrasi dengan berbagai API pemerintah lainnya untuk memberikan pengalaman warga negara yang lancar.

Salah satu pernyataan misi Aadhaar dengan jelas mengidentifikasi sasaran ini: “Mendorong inovasi dan menyediakan platform bagi lembaga publik dan swasta untuk mengembangkan aplikasi yang terhubung dengan Aadhaar”. Contoh bagus lainnya adalah proyek data pemerintah terbuka Jepang. *Information Technology Promotion Agency* (IPA), sebuah entitas yang didanai 100% oleh pemerintah, menjalankan serangkaian program API dan data terbuka. Ini termasuk API yang memaparkan data dari sumber manajemen bencana, khususnya untuk gempa bumi. Strategi data terbuka ini memungkinkan organisasi dan pengembang aplikasi individu untuk membuat aplikasi yang berguna untuk melayani warga negara.

G2B: Pemerintah ke Bisnis

Layanan API pemerintah juga dapat menguntungkan bisnis; penting bagi strategi e-pemerintah yang baik untuk mendukung bisnis dan merangsang ekonomi. Integrasi dan layanan pemerintah yang terkait dengan bisnis dapat didigitalkan untuk melayani bisnis, meningkatkan efisiensi, dan mendatangkan aliran pendapatan baru bagi pemerintah. Pemerintah Singapura merupakan pemain terkemuka dalam strategi data, inisiatif Smart Nation-nya merupakan mahakarya data dan API. Di situs Smart Nation (`https://www.smartnation.sg/resources/open-data`), Anda akan menemukan serangkaian API pemerintah dan titik akhir data yang melayani warga negara dan bisnis.

Inisiatif pemerintah-ke-bisnis yang terkenal mencakup API penyimpanan data untuk pengembang, yang mengungkap jutaan titik data sektor publik seperti LTA Data Mall (berbagai set data terkait transportasi), dan API MAS (API dari otoritas moneter Singapura yang membantu lembaga keuangan dan penyedia aplikasi lainnya).

Melalui API ini, bisnis dan perusahaan rintisan memiliki akses ke sumber daya pemerintah untuk membangun bisnis, perpajakan, informasi bisnis, dan audit regulasi.

G2G: Pemerintah ke Pemerintah

Interaksi layanan antarpemerintah terjadi antara departemen sektor publik dari pemerintah yang sama atau antara dua pemerintah yang berbeda. Amerika Serikat menggunakan sejumlah besar API untuk memfasilitasi transaksi data antara lembaga federal dan departemen keamanannya, sebagian besar informasi yang terkait dengan keamanan nasional, narkotika, dan keselamatan publik.

API ini bertindak sebagai titik integrasi atau titik akhir pengiriman data. Pemerintah Selandia Baru menggunakan strategi data yang terintegrasi dengan departemen internal pemerintah di negara lain untuk memerangi perdagangan manusia. Selain itu, bank sentral dan otoritas serupa menjalankan banyak integrasi sistem antarnegara menggunakan API.

Ringkasan

API adalah keadaan evolusi terkini dalam integrasi dan pertukaran data antara sistem yang berbeda, tetapi sifat implementasinya dan kelincahan bisnis yang disediakannya menjadikannya lebih dari sekadar mekanisme integrasi. Dalam konteks bisnis, API adalah elemen penting dari penciptaan nilai ekonomi. Hal ini menjadikan API sebagai topik diskusi tidak hanya di kalangan pengembang, tetapi juga di ruang rapat dan pemerintahan. Arsitektur berbasis layanan modern lebih menyukai API dan merangkul komunikasi dan integrasi layanan berbasis API. Bab berikutnya merinci strategi API, dan bagaimana strategi tersebut dikembangkan dan dijalankan untuk mendukung visi bisnis. Anda juga akan membaca tentang bagaimana penyampaian nilai API terjadi di berbagai tingkatan, dan bagian-bagian mendasar dari arsitektur API.

BAB 2
STRATEGI DAN ARSITEKTUR API

Munculnya API dan realisasi nilai yang dapat dibawanya bagi bisnis telah menjadikan API sebagai topik yang relevan di luar dunia teknologi dan membawanya ke ruang rapat. Keputusan bisnis dan teknologi masing-masing mendorong strategi API dan arsitektur API. Dalam hal implementasi, API dapat bersifat privat atau publik. API privat melayani sekumpulan pemangku kepentingan tertentu, dan biasanya tidak diekspos di luar pihak-pihak tertentu ini. API publik terbuka untuk dikonsumsi oleh siapa saja (ini tidak berarti gratis). Ada beberapa faktor yang menentukan apakah suatu API bersifat publik atau privat, termasuk keamanan, strategi monetisasi, tren data, dan standar regulasi.

Penting dan tak terelakkan bagi pemangku kepentingan bisnis untuk melibatkan diri dalam pembuatan dan penyusunan strategi API, tidak hanya untuk membuat keputusan yang tepat terkait data dan operasi yang diekspos API, tetapi juga untuk menetapkan harapan, tujuan, dan kendala bagi lingkungan operasional; mendukung visi bisnis melalui strategi API; dan mengevaluasi berbagai model bisnis. Strategi bisnis yang dipilih harus diimplementasikan untuk memastikan tujuan bisnis tercapai. Mengintegrasikan aspek teknis ke dalam strategi API yang teridentifikasi merupakan peran utama arsitektur API. Dalam kebanyakan kasus, area strategi API dan arsitektur API berjalan bersama dengan saling melengkapi atau bertentangan dalam beberapa kasus.

2.1 STRATEGI API

Setiap keputusan bisnis yang melibatkan perencanaan, pengorganisasian, atau tata kelola API dianggap sebagai strategi API. Pemangku kepentingan bisnis mengusulkan visi untuk API, dan pemangku kepentingan teknis mengerjakan desain dan pengembangannya untuk mencapai sasaran bisnis yang ditetapkan. Implementasi API sering kali memicu sejumlah integrasi, sementara pembersihan data mengalir dari sistem lama dan mengungkap spesifikasi domain yang saling bertentangan antara berbagai pihak. Misalnya, pengecer mode yang ingin menarik inovasi dan ekosistem aplikasi dari pengembang eksternal akan memerlukan API yang mengungkap katalog produk dan penjualan, minimal.

Berdasarkan sistem TI dan implementasi yang ada, ini akan memicu integrasi dua sistem atau lebih untuk memberikan fungsionalitas API. Implementasi API awal dari sistem besar atau operasi bisnis sering kali memicu atau mengungkap kompleksitas bisnis dan kurangnya pemahaman domain umum dalam organisasi. Akibatnya, implementasi API terkadang mendorong berbagai pengembangan tambahan, seperti pembersihan data, pemodelan bahasa umum, pemfaktoran ulang basis kode, dan pembaruan alat dan kerangka kerja.

Pengembangan ini tidak dianggap sebagai komponen implementasi API, tetapi merupakan bagian dari misi proyek. Nilai yang diharapkan dari API sangat penting bagi banyak keputusan bisnis yang terkait dengan inovasi, operasi bisnis, integrasi, dan monetisasi.

Pertimbangan ini dapat dikategorikan di bawah aspek strategis bisnis berikut:

1. *Orientasi bisnis:* API disusun berdasarkan orientasi bisnis organisasi. Ini memaparkan tujuan operasi bisnis, yang ingin dicapai dengan implementasi API. Misalnya:
    a. API yang dikembangkan untuk integrasi dua sistem secara teknis dapat mengungkapkan titik integrasi, tetapi dalam istilah bisnis hal ini dilakukan karena konsolidasi dua operasi bisnis yang merupakan bagian dari strategi bisnis tingkat tinggi.
    b. Beberapa organisasi memaparkan API untuk menciptakan dan mengembangkan ekosistem bisnis. Maskapai penerbangan dapat menawarkan pemesanan hotel sebagai tambahan dari operasi pemesanan penerbangan inti mereka. Ini dimungkinkan melalui integrasi dan kolaborasi bisnis antara hotel dan perusahaan penerbangan.
    c. Penyedia layanan perawatan kesehatan dapat memaparkan tren tertentu dalam data kesehatan kepada badan pemerintah. Di beberapa negara, ini merupakan persyaratan peraturan.
2. *Menarik inovasi dan disrupsi:* Organisasi mengekspos data dan operasi (aset TI organisasi) melalui API untuk menarik inovasi dan disrupsi dari luar organisasi, menyuntikkan pemikiran dan keterampilan baru ke dalam bisnis. Organisasi mendapatkan manfaat dari wawasan tentang data bisnis mereka dengan mengeksposnya kepada ilmuwan data, sementara ilmuwan data sering mencari tumpukan data besar untuk melakukan penelitian. Di sini, manfaatnya saling menguntungkan.
3. *Monetisasi:* Organisasi dengan data dan operasi bisnis yang berharga menjualnya secara langsung melalui API.
    a. Organisasi dengan data atau operasi yang berharga mengeksposnya sebagai API, yang menghasilkan arus kas langsung. Konsumen sering membayar biaya untuk menggunakan API ini. Contohnya termasuk API peta dan API kognitif.
    b. Mengekspos operasi sebagai API menawarkan fleksibilitas yang lebih besar untuk integrasi dan peluang untuk menjadi bagian dari ekosistem, sehingga memungkinkan lebih banyak peluang bisnis.

Aspek strategi API di atas menentukan apakah API bersifat privat atau publik, data dan operasi yang akan diekspos, keamanan dan autentikasi, strategi monetisasi, kebijakan penggunaan, dan pembatasan. Mengekspos aset organisasi melalui API publik memiliki manfaat adopsi, inovasi, dan monetisasi pengembang. Pada saat yang sama, hal itu membawa risiko mengekspos aset TI bisnis organisasi ke berbagai audiens eksternal dan meningkatkan permukaan serangan, hal itu juga dapat memiliki kerugian mengekspos data tertentu ke pesaing, jika tidak direncanakan dengan baik. Untuk menghindari dampak negatif tersebut dan memanfaatkan manfaat utama, strategi dan arsitektur API harus bekerja sama untuk menentukan, mengatur, dan menerapkan langkah-langkah yang benar dan tingkat eksposur yang benar dari data yang benar.

2.2 KASUS PENGGUNAAN STRATEGI API

Bayangkan sebuah organisasi dengan pengalaman lebih dari 30 tahun yang menyediakan alat berbasis web untuk penilaian real estat. Aplikasi web tersebut dilisensikan kepada profesional penilaian yang berkualifikasi sebagai langganan bulanan. Profesional penilaian ini mengunjungi lokasi tertentu, mencatat temuan mereka dalam alat tersebut, dan membuat laporan, yang dikirimkan melalui pos ke pihak-pihak terkait.

Pada tahun 2015, organisasi tersebut menyadari bahwa pasar penilaian real estat mengalami gangguan teknologi. Teknologi yang muncul menantang praktik standar; pemeliharaan prediktif dengan pembelajaran mesin dan teknologi sensor berbasis IoT mulai mendominasi pasar, sehingga menciptakan lingkungan bisnis yang menantang bagi organisasi. Pemangku kepentingan bisnis dan teknis organisasi mengembangkan strategi API yang terdiri dari dua item utama:

- Mengekspos data ke lembaga pembelajaran mesin terpilih untuk menghadirkan inovasi bagi organisasi, melalui API pribadi yang diekspos ke mitra terpilih.
- Mengekspos API untuk diintegrasikan dengan bank dan perusahaan asuransi, yang merupakan pengguna utama laporan penilaian. Sebelumnya, laporan dikirim secara manual, dan dengan strategi baru ini, ekosistem API mitra pribadi digunakan untuk mencapai aliran data yang lebih lancar.

Dalam kasus ini, strategi API ditata dengan jelas dengan dua area fokus utama. Salah satunya adalah menghadirkan inovasi bagi organisasi dan tetap relevan dalam lingkungan bisnis yang berubah dengan cepat; yang kedua adalah memperkuat integrasi dan memungkinkan ekosistem untuk menciptakan hubungan bisnis yang lebih erat dan meningkatkan pengalaman sistem.

Rantai Nilai API

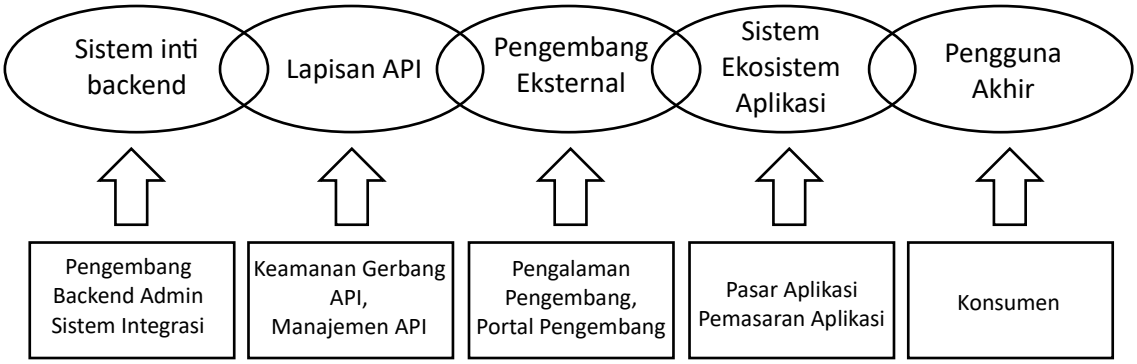
Implementasi API menyentuh berbagai tingkat dan lapisan organisasi. Implementasi API modern sering kali mencakup pemangku kepentingan eksternal dan penyedia nilai lainnya seperti mitra, pemasok, pelanggan, dan pengembang. API mengintegrasikan dan memfasilitasi digitalisasi arus bisnis dengan menghubungkan berbagai pemangku kepentingan dengan aset TI organisasi. Istilah "rantai nilai API" mengacu pada seluruh ekosistem dan urusan antara aset, penyedia API, dan konsumen API.

Misalnya, toko ritel busana mengekspos katalog produk dan operasi penjualannya sebagai API publik. Keputusan diambil untuk memanfaatkan pembelian konsumen berbasis aplikasi. Untuk mencapai hal ini, API harus dapat mengakses sistem TI internal minimal, sistem inventaris dan penjualan. Dengan asumsi kedua sistem ini sudah ada, keputusan untuk mengekspos data dan operasi sebagai API akan memicu arus data antara sistem ini dan lapisan API. Kemudian, API akan digunakan oleh pengembang aplikasi.

Pengembang ini akan menerbitkan aplikasi yang menggunakan API ini, dan API harus memfasilitasi komunitas pengembang dan memberikan pengalaman pengembang yang tepat untuk mempertahankan keterlibatan yang stabil dengan pengembang. Pengalaman pengembang ini hadir dalam dua bentuk. Yang pertama didasarkan pada pengalaman teknis, termasuk dokumentasi, proses onboarding, dan SDK. Yang kedua adalah keuntungan finansial

bagi pengembang eksternal, seperti strategi komisi dan kebijakan periklanan. Terakhir, aplikasi yang dipublikasikan akan menciptakan ekosistem aplikasi.

Aplikasi akan digunakan oleh pengguna akhir sebagian besar pengguna tidak menyadari keseluruhan rantai nilai ini dan kompleksitasnya. Gambar 2.1 menunjukkan bagaimana berbagai lapisan, pemangku kepentingan yang berbeda di setiap lapisan, alat dan proses di berbagai tingkatan saling terhubung dan membentuk rantai nilai API.



Gambar 2.1 Rantai Nilai Api

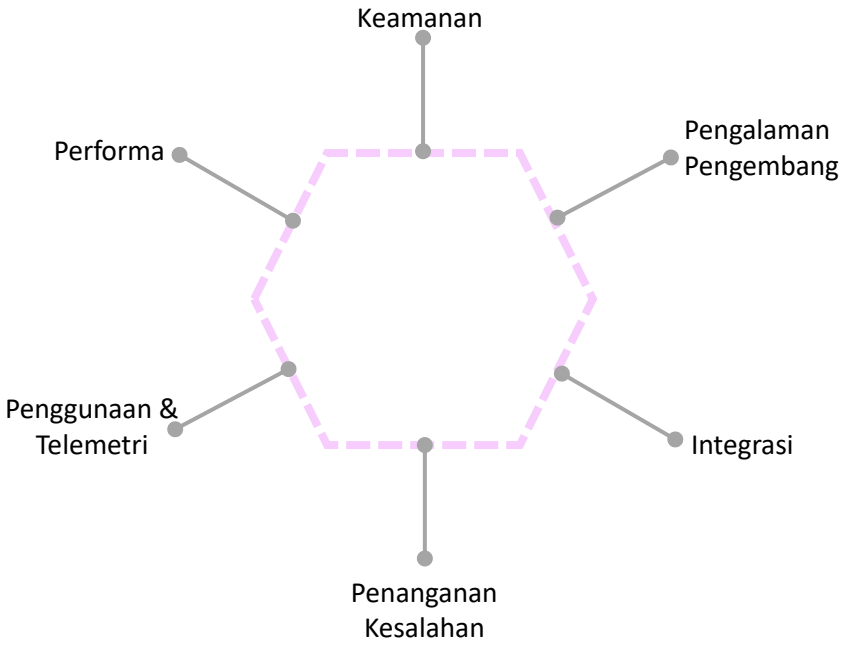
Interkoneksi antara pihak-pihak ini merupakan contoh yang baik dari rantai nilai API dasar.

2.3 ARSITEKTUR API

Pemangku kepentingan bisnis dan pemangku kepentingan lain yang berada di persimpangan bisnis dan teknologi, seperti arsitek perusahaan, pengurus data, dan penginjil organisasi lainnya menentukan tujuan dan strategi API. Keputusan ini akan dievaluasi dalam hal upaya, anggaran, konteks organisasi, dan model aset TI terkini yang diperlukan untuk melaksanakan implementasi API. Implementasi API selanjutnya menjadi tanggung jawab pemangku kepentingan teknis, yang bekerja sama erat dengan pemangku kepentingan bisnis untuk memahami tujuan, pelaksanaan, dan batasan strategi.

Misalnya, CEO jaringan toko ritel memiliki tujuan untuk meningkatkan pendapatan. Ia berkolaborasi dengan pemangku kepentingan lain, seperti tenaga penjualan dan CTO, menemukan bahwa pembelian berbasis seluler dari situs e-commerce mereka meningkat, dan memutuskan untuk meluncurkan aplikasi seluler guna membangun pengalaman konsumen yang lebih nyaman, bersama dengan manfaat lain dari ekosistem aplikasi. Organisasi memutuskan API dan strategi, dan pemangku kepentingan teknis bertanggung jawab untuk mengimplementasikannya.

Untuk merencanakan implementasi, pemangku kepentingan teknis harus memutuskan arsitektur API dan mengidentifikasi kendala dan kekuatan utama yang berperan. Arsitektur API yang umum mencakup enam aspek utama, yang ditunjukkan pada Gambar 2-2. Masing-masing atribut ini memiliki kendala dan pengaruhnya sendiri pada keseluruhan desain API dalam konteks bisnis tertentu.



Gambar 2.2 Aspek Arsitektur API Keamanan: Keamanan Sangat Penting

- *Keamanan:* Keamanan sangat penting. Aspek ini harus mengatasi masalah teknis seperti autentikasi, otorisasi, serangan injeksi, dan serangan DDoS. Ada juga masalah keamanan dalam konteks bisnis, seperti data dan operasi apa yang perlu diekspos, cara mengeksposnya, dan kepada siapa. Mengekspos data dan operasi yang mengungkapkan informasi internal tentang bisnis dapat menciptakan keuntungan bagi pesaing.
- *Pengalaman pengembang:* Keberhasilan implementasi API bergantung pada adopsi. Pengembang adalah audiens utama API, dan harus memiliki pengalaman yang baik dengan semua aspek desainnya. Ini termasuk portal pengembang, forum, alat pengembang, dan titik akhir API uji coba.
- *Performa:* Performa API diharapkan di setiap level, tidak hanya dalam hal responsivitas, tetapi juga ketersediaan. Caching adalah teknik umum yang digunakan untuk meningkatkan responsivitas API. Integrasi berantai banyak dan terjemahan data yang lambat dari sistem lama adalah penyebab umum masalah performa.
- *Integrasi:* Sebagian besar implementasi API tingkat perusahaan memiliki integrasi dengan banyak sistem, dan mayoritas sistem ini adalah sistem lama. API tersebut berkomunikasi dengan semacam integrasi lama atau berkomunikasi dengan API pembungkus, yang melakukan pekerjaan kotor internal. Menggabungkan hasil dari sistem yang berbeda dan melakukan operasi data sebelum mengeksposnya adalah praktik umum untuk banyak implementasi API. Beberapa alat gerbang API memiliki kemampuan terjemahan dan transformasi data dasar yang siap pakai.
- *Penggunaan & Telemetri:* Mengukur penggunaan dan pencatatan serta menganalisis telemetri merupakan aspek penting lain dari arsitektur API, dan ada sejumlah alat yang tersedia untuk membantu hal ini. Pemantauan membantu dalam memahami

penggunaan dan adopsi API, dan analisis berbasis titik akhir mengungkap pola dalam cara titik akhir API digunakan dan titik akhir mana yang digunakan bersama-sama. Rincian ini akan membantu mengoptimalkan desain API secara berkelanjutan.

- *Penanganan kesalahan*: Penanganan kesalahan menentukan bagaimana API harus berperilaku selama kesalahan tingkat aplikasi dan kegagalan sistem. Kegagalan sistem ditangani di bawah arsitektur sistem, sedangkan kegagalan tingkat aplikasi ditangani di bawah arsitektur API. Penanganan kesalahan tingkat aplikasi harus menangani kontrak kesalahan, dokumentasi kesalahan, tingkat informasi kontrak kesalahan, keamanan, dan batasan akses tertentu.

Seperti disebutkan di atas, masing-masing atribut ini memiliki kekuatannya sendiri. Arsitektur API harus mengidentifikasi tingkat pengaruh yang benar untuk setiap atribut tergantung pada persyaratan bisnis dan konteksnya.

2.4 MANAJEMEN API

Manajemen API adalah solusi yang mencakup kumpulan alat yang digunakan untuk merancang dan mengelola API, mengacu pada standar dan alat yang digunakan untuk mengimplementasikan arsitektur API. Ada beberapa produk Manajemen API di pasaran, dan alat Manajemen API adalah beberapa alat perusahaan yang paling dihormati. Microsoft; AWS; IB; dan vendor seperti Apigee, MuleSoft (sekarang Salesforce), dan WSO2 semuanya menawarkan alat Manajemen API.

Meskipun vendor yang berbeda memuat alat Manajemen API mereka menggunakan produk yang berbeda, mereka semua menawarkan solusi untuk desain API, gerbang API, analitik API, dan katalog API.

- *Desain API:* Desain API mencakup kemampuan/fitur seperti mengimpor API dari spesifikasi, membuat titik akhir API, menentukan kontrak layanan, dan menghasilkan dokumentasi.
- *API gateway:* API gateway memungkinkan konfigurasi mesin API gateway, manipulasi permintaan dan respons, penulisan ulang URL, caching, penegakan keamanan dan pra-autentikasi, serta penerapan aturan keamanan berbasis permintaan.
- *Analisis API:* Analisis API mencakup analisis penggunaan API, yang sering kali dikonfigurasi sebagai bagian dari API gateway dan dilacak serta dipantau. Analisis menyediakan wawasan penggunaan dan telemetri serta dasbor pelaporan.
- *Katalog API:* Katalog API dapat mengambil berbagai bentuk dalam implementasi khusus vendor, tetapi secara umum, katalog tersebut mencantumkan API yang tersedia dan konfigurasi akses lainnya.

Misalnya, satu solusi Manajemen API dapat memiliki banyak API; beberapa mungkin bersifat publik, sedangkan sisanya dapat berupa akses privat yang terkait dengan autentikasi tertentu. Alat Manajemen API yang disebutkan di atas mencakup fitur-fitur terperinci seperti dokumentasi API, penerbitan API, penerjemahan protokol, penerjemahan data, transformasi data, kapabilitas keamanan, portal pengembang, caching, pembuatan versi, pembuatan SDK klien, pemantauan penggunaan dan telemetri, penulisan ulang URL, dan banyak lagi.

Manajemen API dan ekosistem alat merupakan bisnis besar dalam industri TI. Banyak penyedia layanan integrasi menawarkan alat Manajemen API, dan penggunaan API oleh perusahaan memicu permintaan akan solusi Manajemen API. Dalam bab-bab berikut, kita akan melihat berbagai fitur alat Manajemen API yang ditawarkan oleh Azure dan AWS. Alat Manajemen API ini ditawarkan sebagai layanan cloud.

Ringkasan

Strategi API mendefinisikan apa yang ingin dicapai organisasi menggunakan API. Ini adalah titik pengambilan keputusan penting dari desain dan inisiasi API. Implementasi API menyentuh berbagai lapisan perusahaan, serta pemangku kepentingan eksternal. Aliran nilai antara lapisan-lapisan ini menjaga keberhasilan implementasi API. Arsitektur API menciptakan visi arsitektur yang berasal dari strategi API dan meletakkan fondasi teknis implementasi API menggunakan komponen arsitektur API. Alat Manajemen API dikemas dengan alat untuk memfasilitasi implementasi API. Dalam bab berikutnya, kita akan mempertimbangkan dasar-dasar desain API dan panduan dasar untuk menulis API yang tepat.

BAB 3
PENGEMBANGAN API

Pengembang masa kini memiliki akses ke berbagai alat dan kerangka kerja untuk membuat layanan web RESTful. Menjalankan layanan cepat dengan titik akhir CRUD untuk model basis data itu mudah, tetapi pengembangan semacam ini tidak menghasilkan API yang dikelola sepenuhnya. Perbedaan utamanya adalah tidak semua layanan web adalah API. Implementasi API harus disusun strateginya lebih dari sekadar titik akhir CRUD; API harus memfasilitasi proses bisnis dan aliran data, mengikuti semantik URI permintaan dan kata kerja HTTP, memiliki pengalaman dan dokumentasi pengembang yang tepat, menerapkan langkah-langkah keamanan yang diperlukan, dan memiliki definisi yang tepat tentang kontrak layanan dan versi.

Saat pengembangan API dimulai, sebagian besar tim pengembangan langsung menggunakan kerangka kerja API favorit mereka dan memulai pengembangan tanpa mempertimbangkan standar atau tim terjebak dengan detail standar implementasi API seperti URI, kata kerja HTTP, pengecualian, pengalaman pengembang, kode status HTTP, dan perang penamaan. Bab ini menyediakan serangkaian panduan yang ringkas namun menyeluruh untuk memulai pengembangan API Anda dengan keseimbangan sempurna antara pengembangan dan standar API.

3.1 PERTIMBANGAN PENGEMBANGAN API

API berbeda dari aplikasi web dan layanan web. Secara teknis, API masa kini lebih mirip dengan implementasi layanan RESTful, tetapi tujuan API berbeda dan melampaui ekspektasi dibandingkan dengan layanan RESTful biasa. Semua API dengan semantik RESTful dapat dianggap sebagai layanan RESTful, tetapi tidak semua layanan RESTful bukan API. API diimplementasikan untuk mengekspos data dan operasi kepada pemanggil. Kegunaan dan adaptasi API oleh berbagai pemanggil eksternal merupakan faktor keberhasilan yang krusial untuk setiap implementasi.

Untuk mencapai hal ini, implementasi API memiliki pertimbangan tambahan yang harus dihadapi. Seperti yang dinyatakan, alat dan kerangka kerja modern menawarkan fitur yang kaya untuk memberikan implementasi API yang diperlukan. Memahami pertimbangan pengembangan API yang mendasar diperlukan untuk mengimplementasikan API yang dapat dikembangkan dengan baik.

Parameter Eksplisit

Implementasi API harus tanpa status dan tidak bergantung pada status konsumen, yang berarti bahwa API harus menerima parameter dari pemanggil dan tidak boleh bergantung pada model sisi klien yang persisten dengan status. Contoh umum adalah: ada layanan RESTful yang dikembangkan untuk melayani aplikasi web tertentu. Dalam kasus seperti itu, biasanya layanan ini dikembangkan untuk menerima data dari cookie, yang dapat diterima dari aplikasi web ke panggilan layanan RESTful. Namun, jika layanan RESTful yang sama dimaksudkan untuk

berfungsi sebagai API bagi konsumen yang berbeda, mengandalkan cookie akan merusak banyak hal. Jadi implementasi API harus memiliki parameter eksplisit dan menerima data dari parameter URL atau header HTTP atau melalui isi permintaan HTTP.

Hindari Titik Akhir yang Diperintahkan Konsumen

API dan konsumen mengirim dan menerima pesan menggunakan kontrak layanan yang ditentukan. Definisi kontrak layanan ditentukan oleh layanan atau konsumen. Keduanya merupakan metode yang diterima, tetapi kontrak yang ditentukan konsumen berpusat pada aplikasi, sedangkan kontrak yang ditentukan layanan didasarkan pada entitas dan operasi bisnis.

Namun, sebaiknya hindari titik akhir yang menyajikan data yang diperintahkan konsumen data yang diperintahkan konsumen adalah tentang definisi kontrak layanan yang berisi model tampilan aplikasi tertentu, seperti berbagai format data dan API yang mengekspos titik akhir untuk agregasi data sederhana. Dalam beberapa kasus, respons API berisi gaya visual seperti kode warna. Jenis implementasi ini harus dihindari sebisa mungkin, karena mereka menghubungkan implementasi API dengan klien tertentu dan tampilan aplikasi tertentu.

Dokumentasi

API harus memiliki dokumentasi tentang standar yang digunakan, versi, sintaks URI, dan kode kesalahan. Pengalaman pengembang bukanlah elemen opsional dalam pengembangan API; alat seperti Swagger dan TRex membantu dalam membuat dokumentasi dan pengalaman pengembang. Alat Manajemen API yang lengkap menyediakan dokumentasi yang lengkap dan pengalaman pengembang.

Keamanan

Keamanan API adalah suatu keharusan. Keamanan tidak hanya tentang autentikasi dan otorisasi, tetapi juga tentang data apa yang diekspos dalam kontrak layanan mana dan bagaimana titik akhir dikonsumsi oleh konsumen. Menampilkan properti data dalam respons kesalahan dapat membantu konsumen untuk mengoreksi permintaan dan mencoba lagi dengan parameter permintaan yang benar, tetapi pada saat yang sama, hal ini dapat membuka celah keamanan yang dapat dieksploitasi oleh peretas untuk memperoleh data tertentu.

Penting untuk mencapai keseimbangan antara API yang memiliki respons yang membantu bagi klien tanpa mengekspos informasi sensitif. Selain itu, API publik harus memiliki langkah-langkah keamanan seperti keamanan berbasis IP, melacak penggunaan kunci API, atau membatasi kecepatan panggilan. Alat Manajemen API modern menawarkan aspek keamanan berbasis permintaan yang disebutkan di atas.

Pembuatan versi

API adalah perangkat lunak, dan perangkat lunak berevolusi. Pengembangan API harus mempertimbangkan pembuatan versi; pembuatan versi API mencakup dua aspek, satu adalah pembuatan versi URI dan yang kedua adalah pembuatan versi kontrak layanan. Ada banyak teknik pembuatan versi API yang tersedia, dan teknik yang tepat harus dipilih pada tahap awal pengembangan. Pemberitahuan segera kepada pengembang tentang versi baru dan terutama penyusutan versi lama sangat penting.

3.2 STANDAR PENGEMBANGAN API

Ada beberapa praktik terbaik yang harus diikuti dalam pengembangan API. Seperti yang dinyatakan dalam pengantar bab ini, terkadang pengembang kewalahan dengan informasi yang tersedia tentang standar ini. Tujuan dari bagian ini adalah untuk memberikan serangkaian standar minimum terbaik untuk memulai pengembangan API dengan lebih sedikit gesekan, pada saat yang sama standar ini memungkinkan pengembang untuk memperluas ke implementasi yang lebih komprehensif jika diperlukan. Lima standar di bawah ini diekstraksi dari banyak implementasi API modern dan dikompilasi sebagai panduan pengembang yang praktis.

Kata Kerja HTTP

Kata kerja HTTP adalah elemen tindakan utama dalam komunikasi HTTP. Tabel 3.1 menunjukkan kata kerja HTTP yang paling umum digunakan dalam pengembangan API.

Tabel 3.1 Panduan Singkat Kata Kerja HTTP

Kata Kerja HTTP	Penggunaan Umum
GET	Dapatkan satu entitas atau daftar entitas
POST	Buat sebuah entitas
DELETE	Hapus suatu entitas (ini bisa berupa penghapusan sementara)
PUT	Ganti entitas
PATCH	Perbarui properti suatu entitas

PUT vs PATCH: Pengembang sering kali membingungkan kedua kata kerja ini. PATCH adalah tambahan terbaru pada kata kerja HTTP, dan didefinisikan dengan baik dalam RFC 5789. Perbedaan antara permintaan PUT dan PATCH tercermin dalam cara server memproses entitas terlampir untuk mengubah sumber daya yang diidentifikasi oleh Request-URI. Dalam permintaan PUT, entitas terlampir adalah versi modifikasi dari sumber daya yang disimpan di server asal, dan klien meminta agar versi yang disimpan diganti dengan versi baru dalam isi permintaan.

Namun, dengan PATCH, entitas terlampir berisi serangkaian instruksi yang menentukan bagaimana sumber daya yang saat ini berada di server asal harus diubah untuk menghasilkan versi baru. Metode PATCH memengaruhi sumber daya yang diidentifikasi oleh Request-URI, dan mungkin juga memengaruhi sumber daya lainnya; yaitu, sumber daya baru dapat dibuat, atau sumber daya yang sudah ada diubah, dengan penerapan PATCH. Umumnya diamati bahwa PUT digunakan dalam banyak operasi penyuntingan, beberapa API menggunakan PATCH.

Kode Status HTTP

Kode status HTTP menunjukkan status respons dari server, dan ditetapkan dalam rentang. Beberapa implementasi API juga memiliki kode HTTP sendiri. Tabel 3.2 menunjukkan kode status HTTP yang paling umum untuk memulai pengembangan API dengan cepat.

Tabel 3.2 Panduan Cepat Kode Status HTTP

Kode Status	Penggunaan Umum
200	**OK** - Setiap permintaan yang berhasil diproses. Mungkin mengandung atau tidak mengandung muatan.
201	**CREATED** - Kode respons umum untuk permintaan POST. Isi berisi URI entitas yang baru dibuat.
202	**ACCEPTED** - Permintaan diterima. Instruksikan klien untuk melanjutkan. Mungkin berisi URI untuk memeriksa status permintaan ini, yang dapat digunakan klien untuk melakukan tindak lanjut.
204	**NO CONTENT** - Kode respons umum untuk permintaan PUT/PATCH/DELETE. Biasanya, badan tidak berisi muatan.
400	**BAD REQUEST** - Status umum untuk menunjukkan masalah dalam permintaan, yaitu validasi.
401	**UNAUTHORIZED** - Digunakan untuk menunjukkan autentikasi/otorisasi klien telah gagal.
403	**FORBIDDEN** - Digunakan untuk menunjukkan otorisasi telah gagal atau prasyarat telah gagal.
404	**NOT FOUND** - Sumber daya yang diminta tidak ditemukan.
408	**REQUEST TIMEOUT** - Server tidak dapat memproses permintaan dalam waktu yang ditentukan. Terkadang payload berisi informasi percobaan ulang.
500	**INTERNAL SERVER ERROR** - Pengecualian yang tidak tertangani dan kesalahan server termasuk dalam kategori ini. Memberi tahu klien bahwa masalahnya ada pada server.

Ada kode tambahan yang tersedia untuk respons yang lebih terperinci, dan ada variasi pada respons ini; misalnya, kesalahan validasi dalam entitas dapat mengembalikan "400 - Bad Request" dengan informasi terperinci, atau "412 - Precondition Failed." Kekhawatiran lain yang dimiliki pengembang adalah tentang penggunaan 404 sebagai kode status untuk permintaan pengambilan sumber daya, seperti mendapatkan entitas berdasarkan ID.

Beberapa berpendapat bahwa mengembalikan 404 tidak valid, karena titik akhir untuk mengambil sumber daya valid, tetapi entitas itu sendiri tidak tersedia. Jadi mereka menganjurkan untuk mengembalikan 400 sebagai BadRequest atau 200 dengan isi kosong, bukan 404, karena titik akhir ditemukan. Ada banyak argumen seperti ini, tetapi merupakan praktik desain dan pengembangan yang baik untuk konsisten dalam menggunakan kode respons HTTP dan memiliki pesan terperinci di bagian badan, khususnya selama skenario berbasis 400 dan 500.

Penanganan Kesalahan

Penanganan kesalahan penting, dan harus diterapkan dengan cara yang membantu konsumen dalam memperbaiki masalah dalam permintaan dan membimbing mereka untuk mengakses API kembali. Praktik terbaik yang umum adalah mengembalikan respons kesalahan dengan setidaknya tiga parameter:

- Kode status HTTP yang benar
- Kode kesalahan khusus API
- Pesan kesalahan yang mudah dipahami manusia

Kode kesalahan khusus API membantu penerapan logika klien dengan mudah, daripada

memproses pesan string yang mudah dipahami manusia. Ini juga membantu dalam penerapan logika kontrol aliran yang baik pada klien. Respons kesalahan dapat berisi detail seperti tautan coba lagi, interval waktu coba lagi, dan parameter bantuan tambahan untuk mengubah objek respons.

Tabel 3.3 Properti Kontrak Layanan Detail Masalah

Property	*Tipe*	*Deskripsi*
type	string	URI untuk jenis kesalahan
title	string	Deskripsi singkat tentang kesalahan tersebut
detail	string	Penjelasan rinci tentang kesalahan tersebut
instance	string	Contoh kesalahannya

RFC 7807 - 'Detail Masalah untuk API HTTP' yang terbaru memiliki informasi yang baik tentang membangun kontrak layanan detail masalah. Berdasarkan RFC, respons detail masalah mencakup properti berikut. Tabel 3.3 menunjukkan detail setiap entitas detail masalah. Objek dapat diperluas dengan properti kustom. Daftar 3.1. Contoh Pesan Detail Masalah seperti yang Ditetapkan dalam RFC 7807

```
{
   "type": "https://example.com/probs/out-of-credit",
   "title": "You do not have enough credit.",
   "detail": "Your current balance is 30, but that costs 50.",
   "instance": "/account/12345/msgs/abc",
   "balance": 30
}
```

Catatan Mengirim informasi dalam respons kesalahan merupakan keputusan penting dan harus dipertimbangkan dengan hati-hati. Dalam contoh di atas, mengirimkan saldo saat ini dalam respons kesalahan memiliki kelebihan dan kekurangannya sendiri. Pertimbangan keamanan secara keseluruhan dalam pengembangan harus menentukan keputusan tersebut. Misalnya, jika pengembangan menekankan keamanan tinggi dengan mempertimbangkan pelanggaran yang diasumsikan, maka lebih baik tidak mengekspos saldo dalam respons kesalahan.

Sintaksis URL

Dalam layanan RESTful berbasis HTTP, parameter dilewatkan dalam URI permintaan atau header HTTP, atau dalam badan permintaan. Terserah pengembang API untuk menentukan parameter apa yang dikirim pada jalur mana. Umumnya, permintaan GET meneruskan parameter dalam URI, kecuali ada persyaratan khusus untuk melakukannya seperti parameter pencarian kompleks yang dikirim dalam badan permintaan.

Sintaks URI dapat berbasis string kueri atau berdasarkan fragmen URI, dan ada diskusi tak berujung tentang metode mana yang lebih baik. Kerangka kerja API modern mendukung kedua sintaksis jika konvensi penamaan tertentu diikuti, tetapi sintaksis berbasis fragmen URI umumnya lebih disukai karena pendekatan semantiknya: `api/orders/1` daripada

`api/orders?id=1`. Dimungkinkan untuk menempatkan parameter di tengah URI; ini membuatnya lebih mudah dibaca: `api/order/1/products` daripada `api/order/products?orderId=1`.

Pada saat yang sama, fragmen URI tidak dapat bekerja dengan baik pada semua skenario, seperti operasi pencarian/filter, terutama saat kita memerlukan titik akhir pencarian dengan parameter yang berubah-ubah. Jika kita hanya memiliki satu parameter nilai kunci pencarian pada waktu tertentu, maka pendekatan termudah adalah memperkenalkan operasi filter dan mengambil dua parameter, satu adalah tipe properti filter (parameter kunci) dan properti kedua adalah parameter nilai. Dalam contoh di bawah ini, "kategori" dan "mode" adalah parameter: `api/produk/filter/{kategori}/{mode}`.

Namun, skenario pencarian yang lebih kompleks lebih baik dilayani dengan muatan permintaan. Mengirim permintaan GET dengan muatan yang menentukan properti, nilai, dan kriteria filter dalam badan permintaan akan lebih cocok daripada merangkai fragmen URI yang panjang, ini juga dapat memengaruhi batasan panjang URI. RESTful tidak banyak mendefinisikan sintaks URI, tetapi pengembang sering berdebat tentang topik ini. Dengan menggunakan pendekatan segmentasi URI di atas, URI menjadi lebih ramah pengguna, dan menjaga URI permintaan bersih dari karakter seperti “?”, “=”, dan “#.”

3.3 PEMBUATAN VERSI

Standar pembuatan versi API umumnya ditangani dengan tiga cara berbeda. Metode mana pun yang digunakan, API mengharapkan versi dari klien sebagai parameter, dan jika klien tidak menentukan versinya, API akan kembali ke versi default atau akan menampilkan kesalahan. Versi API dapat dinyatakan dalam URI permintaan sebagai fragmen (lebih umum) atau sebagai parameter string kueri: `api/v1/products` atau `api/products?api-version=1`. Atau, nyatakan versi API dalam header HTTP. Umumnya, kunci header “`api-version`” digunakan untuk meneruskan versi ke server, tetapi pengembang dapat menggunakan kunci header kustom mereka sendiri.

Atau nyatakan versi API dalam kunci header Accept standar. Header ini digunakan untuk negosiasi konten; dalam API berbasis JSON, header Accept berisi `application/JSON` sebagai nilai default, tetapi beberapa API mengharapkan nomor versi berada di header Accept: `application/massrover.v2+json`. Terlepas dari metode yang digunakan, implementasi harus konsisten di semua titik akhir API. Selain itu, satu API dapat memiliki lebih dari satu metode yang diaktifkan dalam implementasinya. Metode implementasi yang paling mudah dan paling langsung adalah dengan mencantumkan versi di URL.

Kick-Start Pengembangan API

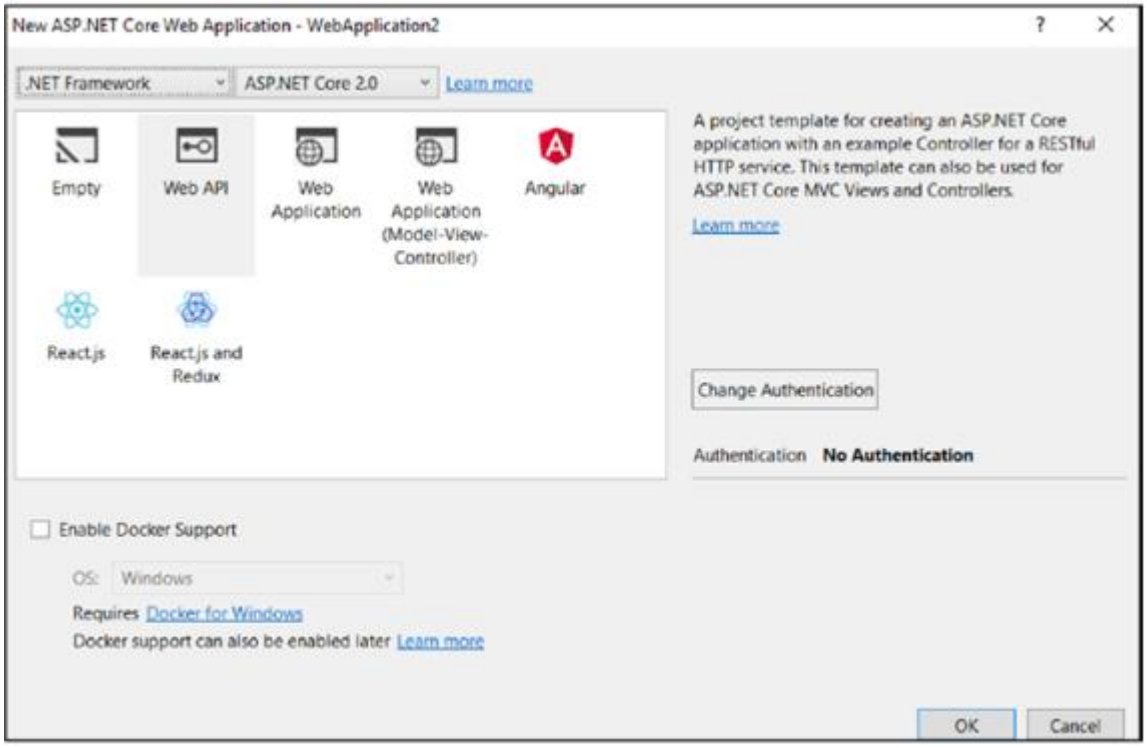
Bagian ini menjelaskan pendekatan ringkas untuk memulai pengembangan API menggunakan ASP.NET Core dan perkakas Visual Studio terkait. Kami juga akan mengeksplorasi cara menggunakan spesifikasi API untuk menjelaskan API RESTful. Spesifikasi OpenAPI (OAS) digunakan untuk tujuan ini, bersama dengan perkakas Swagger lain yang tersedia. OAS menggunakan antarmuka standar yang tidak bergantung pada bahasa untuk mendeskripsikan API RESTful, yang memungkinkan manusia dan komputer untuk menemukan

dan memahami kemampuan layanan tanpa akses ke kode sumbernya. Tersedia berbagai alat yang lengkap untuk mengimplementasikan spesifikasi OAS dan membuat dokumentasi API dari deskripsi API.

Spesifikasi OpenAPI (OAS) secara resmi dikenal sebagai Swagger. Versi Swagger sebelumnya menyertakan spesifikasi dan alat API. Pemilik Swagger, SmartBear, menyumbangkan spesifikasinya, sehingga membuatnya independen dari perkakas dan membuat spesifikasi tersebut netral terhadap vendor. Perkakas tersebut tetap menggunakan nama merek Swagger. Jadi, dalam konteks saat ini, OpenAPI merujuk pada spesifikasi yang netral terhadap vendor, dan Swagger merujuk pada alat yang digunakan untuk mengimplementasikan spesifikasi tersebut. Tersedia berbagai alat lain untuk mengimplementasikan OAS.

Implementasi: ASP.NET Core

Contoh API (MassRover API) memiliki titik akhir CRUD untuk satu entitas. Untuk memulai pengembangan, buat aplikasi ASP.NET Core Web API sederhana di Visual Studio 2017. Gambar 3.1 menunjukkan templat proyek yang dipilih di Visual Studio.



Gambar 3.1 Proyek Api Asp.Net Core

Tambahkan folder dalam proyek dan beri nama "`Models.`" Buat model Produk dalam folder ini.

`Product.cs`

```
public class Product
{
  public int Id { get; set; }
```

```
    public string Name { get; set; }
    public DateTime? ModifiedDate { get; set; }
}
```

Tambahkan folder lain dan beri nama "Errors." Folder ini akan berisi semua kelas dan enum yang digunakan untuk menyediakan implementasi penanganan kesalahan API sebagaimana didefinisikan dalam RFC 7807.

ErrorCode.cs

```
public enum ErrorCode
{
    RequestContentMismatch = 18000,
    EntityNotFound = 18500
}
```

Contoh enum ErrorCode tidak berisi standar apa pun; implementasinya khusus untuk aplikasi tersebut. Saat Anda mengembangkan API, pastikan kode kesalahan konsisten di seluruh aplikasi dan dokumentasi, karena pengguna API harus membuat keputusan dan menulis kode berdasarkan standar yang ditetapkan. Tambahkan kelas abstrak di folder Errors, bernama "ErrorMessage," dengan properti dasar kontrak kesalahan, dan terapkan dua kelas konkret ErrorMessage khusus bernama "RequestContentErrorMessage" dan "EntityNotFoundErrorMessage."

ErrorMessage.cs

```
public abstract class ErrorMessage
{
    public ErrorCode Code { get;set; }
    public string Type { get; set; }
    public string Title { get; set; }
    public string Detail { get; set; }
    public string Instance { get; set; }
    public string Info { get; set; }
}

public class RequestContentErrorMessage : ErrorMessage
{
    public RequestContentErrorMessage()
    {
        Code = ErrorCode.RequestContentMismatch;
        Type = $"https://massrover.com/doc/errors/#
            {ErrorCode.RequestContentMismatch.ToString()}";
    }
}

public class EntityNotFoundErrorMessage : ErrorMessage
{
    public EntityNotFoundErrorMessage()
```

```
    {
        Code = ErrorCode.RequestContentMismatch;
        Type = $https://massrover.com/doc/
        errors/#{ErrorCode.EntityNotFound.ToString()};
    }
}
```

Selanjutnya, tambahkan kelas lain untuk mengembalikan instance ErrorMessage yang benar dalam konteks yang tepat. Kelas ini mensimulasikan layanan yang menghasilkan instance `ErrorMessage` yang benar. Dalam implementasi di dunia nyata, ini akan menjadi bagian dari logika bisnis.

ErrorService.cs

```
public static class ErrorService
{
    public static ErrorMessage
    GetRequestContentMismatchErrorMessage()
    {
        return new RequestContentErrorMessage
        {
            Title = $"Request content mismatch",
            Detail = $"Error in the request context."
        };
    }

    public static ErrorMessage GetEntityNotFoundErrorMessage
    (Type entity, int id)
    {
        return new EntityNotFoundErrorMessage
        {
            Title = $"{entity.Name} not found",
            Detail = $"No {entity.Name.ToLower()} found for
            the supplied id - {id}"
        };
    }
}
```

Sekarang, mari kita buat pengontrol ASP.NET Core dengan tindakan untuk operasi CRUD entitas Product. Untuk melakukannya, tambahkan pengontrol Web API kosong bernama "`ProductsController`" (titik akhir tercantum dalam Tabel 3.4). Jika Anda mau, Anda dapat menghapus ValuesController yang dibuat dengan templat Visual Studio.

Tabel 3.4 Titik Akhir `ProductsController`

Nama Aksi	Metode HTTP	Respon	Respon Kesalahan
GetProducts	GET	200 List of Products	-
GetProductByld	GET	200 Product	404 – Entity Not Found
CreateProduct	POST	201 New Product	-

UpdateProduct	PUT	204 No Content	400 – Bad Request 404 – Entity Not Found
DeleteProduct	DELETE	204 No Content	404 – Entity Not Found

ASP.NET Core menyediakan atribut yang berguna dalam mendekorasi API. Atribut ini berguna selama pengembangan, dan alat seperti Swagger memanfaatkan deskripsi atribut tersebut dalam menghasilkan deskripsi API. MassRover API adalah contoh implementasi referensi dan tidak menyediakan standar apa pun dalam pengodean, struktur aplikasi, arsitektur tingkat kode, atau pemisahan masalah.

Ini adalah aplikasi referensi sederhana yang digunakan untuk menjelaskan dan menguji informasi terperinci dalam pengembangan API, bukan referensi pengembangan/arsitektur. Ikuti petunjuk berikut dan kembangkan kode untuk ProductController.cs. Pertama, tambahkan koleksi produk (hardcoded) di pengontrol seperti yang ditunjukkan di bawah ini.

```
[Produces("application/json")]
[Route("api/products")]
public class ProductsController : Controller
{
   private static List<Product> _products = new
   List<Product>
   {
      new Product {Id = 1, Name = "Lithim L2",
      ModifiedDate = DateTime.UtcNow.AddDays(-2)},
      new Product {Id = 2, Name = "SNU 61" }
   };
}
```

Selanjutnya, mari tambahkan dua tindakan HTTP GET: satu untuk mengambil semua produk, dan satu lagi untuk mengambil produk berdasarkan ID-nya, parameter diteruskan melalui jalur URI. Perhatikan juga bahwa metode tindakan didekorasi dengan atribut yang tepat untuk metode HTTP, parameter rute, dan jenis respons. Setiap tindakan juga memiliki komentar XML-nya sendiri.

```
/// <summary>
/// Gets list of all Products
/// </summary>
/// <returns>List of Products</returns>
/// <response code="200">List of Products</response>
[HttpGet]
[ProducesResponseType(typeof(List<Product>), 200)]
public IActionResult GetProducts()
{
   return Ok(_products);
}
/// <summary>
/// Gets product by id
```

```
/// </summary>
/// <param name="id">Product id</param>
/// <returns>Product</returns>
/// <response code="200">Product</response>
/// <response code="404">No Product found for the
specified id</response>
[HttpGet("{id}")]
[ProducesResponseType(typeof(Product), 200)]
[ProducesResponseType(typeof(EntityNotFoundError
Message), 404)]
public IActionResult GetProductById(int id)
{
   var product = _products.SingleOrDefault(p => p.Id == id);

   if (product != null)
      return Ok(product);
   else
      return NotFound
         (ErrorService.GetEntityNotFoundErrorMessage
         (typeof(Product), id));
}
```

Tambahkan tindakan untuk membuat produk baru. HTTP POST membuat produk baru, dengan mengambil parameter di badan permintaan.

```
/// <summary>
/// Creates new product
/// </summary>
/// <param name="product">New Product</param>
/// <returns>Product</returns>
/// <response code="201">Created Product for therequest</response>
[HttpPost]
[ProducesResponseType(typeof(Product), 201)]
public IActionResult CreateProduct([FromBody]Product product)
{
   product.Id = _products.Count + 1;
   _products.Add(product);

   return CreatedAtRoute(new { id = product.Id }, product);
}
```

Tambahkan metode PUT untuk mengganti produk dengan ID yang ditentukan. Tindakan ini memerlukan dua parameter: ID produk yang akan diganti (ini diteruskan sebagai variabel jalur) dan produk yang akan diganti dengan nilai baru, yang diteruskan dalam badan permintaan. Tindakan ini mengembalikan respons NoContent dengan kode status HTTP 204 untuk penggantian produk yang berhasil. Jika tidak, respons akan muncul dengan dua kontrak kesalahan yang berbeda. Salah satunya adalah "400 Bad Request," dengan konten permintaan yang tidak cocok saat nilai ID dalam jalur tidak cocok dengan nilai ID produk dalam badan permintaan. Yang lainnya adalah "404 Not Found," saat entitas yang diminta dengan ID yang

ditentukan tidak ditemukan.

```
/// <summary>
/// Replaces a product
/// </summary>
/// <param name="id">New version of the Product</param>
/// <param name="product">New version of the Product</param>
/// <returns></returns>
/// <response code="204">No Content</response>
/// <response code="400">Request mismatch</response>
/// <response code="404">No Product found for the
specified id</response>
[HttpPut("{id}")]
[ProducesResponseType(204)]
[ProducesResponseType(typeof(RequestContentErrorMessage),400)]
[ProducesResponseType(typeof(EntityNotFoundErrorMessage), 404)]
public IActionResult UpdateProduct(int id, [FromBody] Product product)
{
  if (id != product.Id)
    return BadRequest(ErrorService.GetRequest
    ContentMismatchErrorMessage());

  var existingProduct = _products.SingleOrDefault
  (p => p.Id == product.Id);

  if (existingProduct != null)
  {
    existingProduct = product;
    existingProduct.ModifiedDate = DateTime.UtcNow;
  }
  else
    return NotFound
      (ErrorService.GetEntityNotFoundErrorMessage
      (typeof(Product), product.Id));

  return NoContent();
}
```

Tambahkan titik akhir hapus menggunakan tindakan HTTP DELETE.

```
/// <summary>
/// Deletes a product
/// </summary>
/// <param name="id">Product id</param>
/// <returns></returns>
/// <response code="204">No Content</response>
/// <response code="404">No Product found for the specified id</response>
[HttpDelete("{id}")]
[ProducesResponseType(204)]
[ProducesResponseType(typeof(EntityNotFoundError Message), 404)]
public IActionResult DeleteProduct(int id)
```

```
{
   var product = _products.SingleOrDefault(p => p.Id == id);
      if (product != null)
         _products.Remove(product);
      else
         return NotFound
            (ErrorService.GetEntityNotFoundErrorMessage
            (typeof(Product), id));
   return NoContent();
}
```

Ini menyimpulkan kode di ProductController. Selanjutnya, kita akan menyiapkan alat Swagger di ASP.NET Core dan memanfaatkan alat dengan elemen atribut yang dihias.

3.4 MENYIAPKAN SWAGGER

Instal paket `Swashbuckle.AspNetCore` di proyek dengan menjalankan perintah berikut di Package Manager Console (PMC):

```
Install-Package Swashbuckle.AspNetCore.
```

Selanjutnya, siapkan `Startup.cs` untuk mengaktifkan perkakas Swagger dan mengaktifkan serta menjalankan UI Swagger di proyek. Di `Startup.cs`, perbarui metode `ConfigureServices`, seperti yang ditunjukkan di bawah ini. Perbarui nama API (MassRover API) dan versi API (v1) di `SwaggerDoc`, dan perbarui jalur untuk dokumentasi XML agar Swagger dapat digunakan melalui `IncludeXmlComments`. Dalam contoh, untuk menyediakan jalur XML, instal paket `Microsoft.Extensions.PlatformAbstractions` menggunakan PMC. Jalankan yang berikut ini:

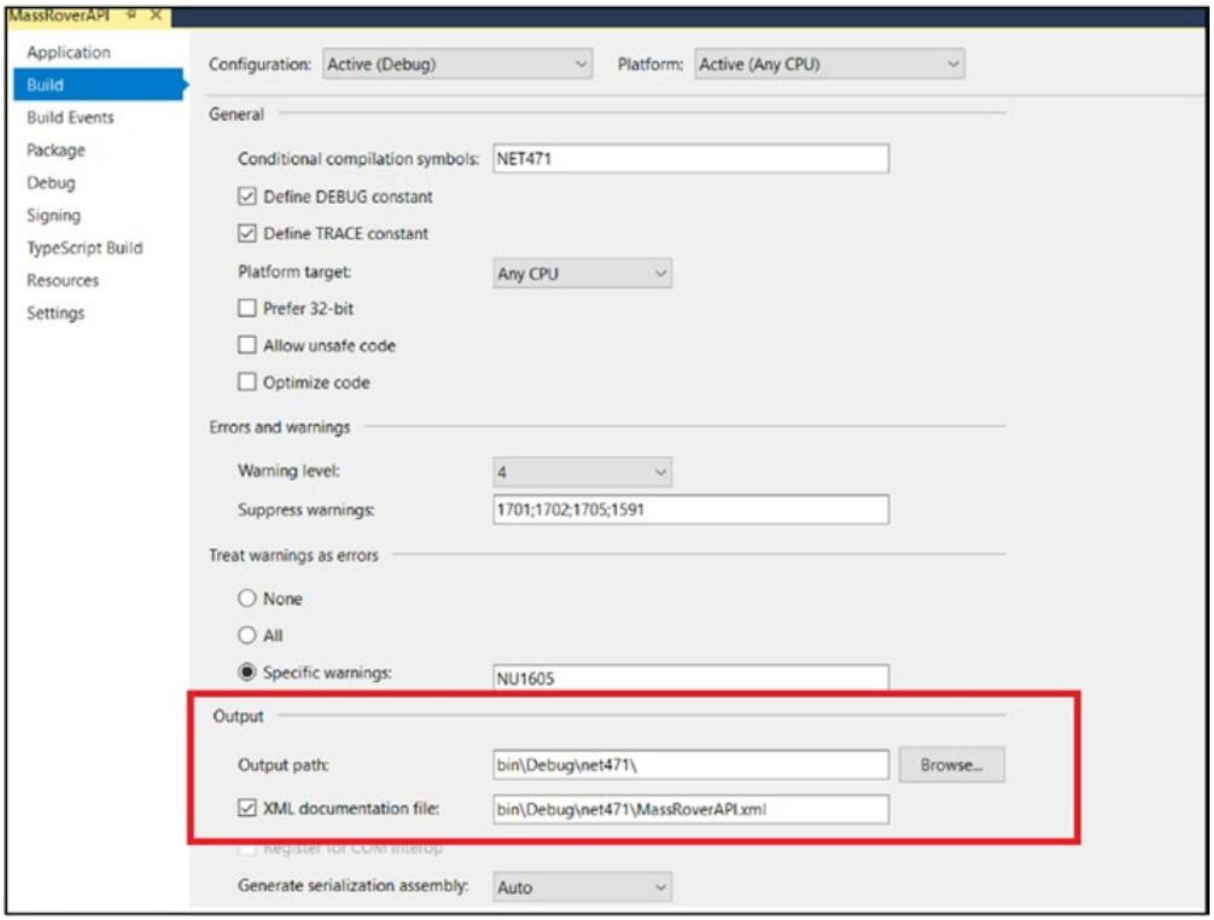
```
Install-Package Microsoft.Extensions.PlatformAbstractions
public void ConfigureServices(IServiceCollection services)
{
   services.AddMvc();
   services.AddSwaggerGen(c =>
   {
      c.SwaggerDoc("v1", new Info { Title = "MassRover API", Version = "v1" });
      c.IncludeXmlComments
         (Path.Combine(PlatformServices.Default.Application.ApplicationBasePath
         ,"MassRoverAPI.QuickStartSample.xml"));
   });
}
```

Perbarui metode Konfigurasi seperti yang ditunjukkan di bawah ini. Ini akan mengaktifkan UI Swagger dan menetapkan titik akhir definisi Swagger.

```
public void Configure(IApplicationBuilder app,
IHostingEnvironment env)
```

```
{
   if (env.IsDevelopment())
   {
      app.UseDeveloperExceptionPage();
   }

   app.UseSwagger();
   app.UseSwaggerUI(s =>
   {
      s.SwaggerEndpoint("/swagger/v1/swagger.json",
      "MassRover Open API");
   });

   app.UseMvc();
}
```

Terakhir, kami akan memerintahkan Visual Studio untuk membuat dokumentasi XML berdasarkan komentar. Navigasi ke Properti Proyek, lalu tab Bangun, dan aktifkan berkas dokumentasi XML. Secara default, ini adalah jalur untuk bin. Lihat Gambar 3.2.



Gambar 3.2 Pengaturan Dokumentasi XML di Visual Studio

Konfigurasi bangun di Visual Studio memiliki pengaturan khusus untuk setiap konfigurasi, jadi Anda harus melakukan langkah ini untuk setiap konfigurasi bangun (debug, rilis, dan lain-lain)

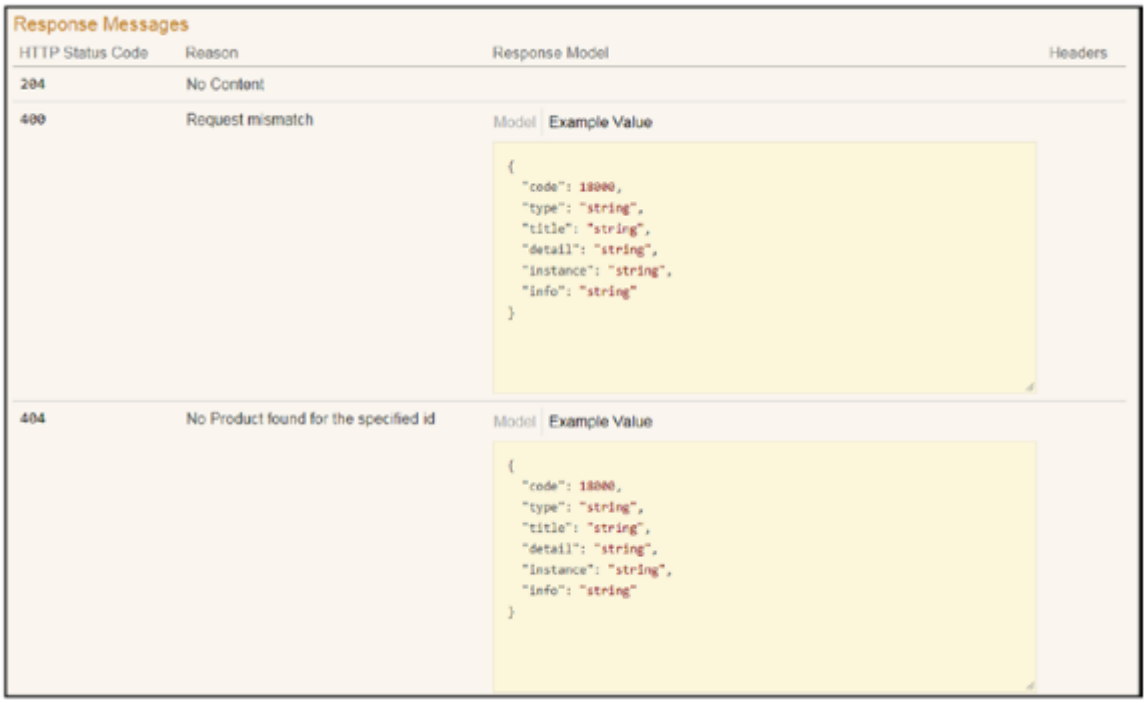
guna menghasilkan berkas XML dalam pengaturan konfigurasi bangun masing-masing. Perhatikan Masalah Garis Hijau: Mengaktifkan dokumentasi XML menyebabkan Visual Studio mencari komentar XML untuk setiap implementasi. Ini menciptakan garis hijau peringatan Visual Studio di mana-mana. Untuk menghilangkannya, Anda dapat menambahkan nomor aturan (1519) dalam teks Sembunyikan Peringatan, seperti pada Gambar 3.2.

Jalankan API dan Swagger

Sekarang Anda dapat menjalankan aplikasi. Navigasi ke URL `http://{host}:{port yang ditentukan}/swagger`. UI Swagger mudah dipahami, dan memanfaatkan dokumentasi XML yang dihasilkan oleh Visual Studio (Gambar 3.3). UI ini menyediakan detail titik akhir, parameter, jenis respons, dan kode respons. UI ini juga menyediakan deskripsi komprehensif dari komponen-komponen ini seperti yang dijelaskan dalam komentar XML.

Gambar 3.3 UI Swagger

Jika Anda memperluas metode PUT, Anda akan melihat layar yang mirip dengan yang ditunjukkan pada Gambar 3.4, termasuk penjelasan terperinci tentang kode respons HTTP dan pesan respons. Perhatikan bahwa Swagger memanfaatkan atribut `ProducesResponseType` untuk menghasilkan model kontrak kesalahan.



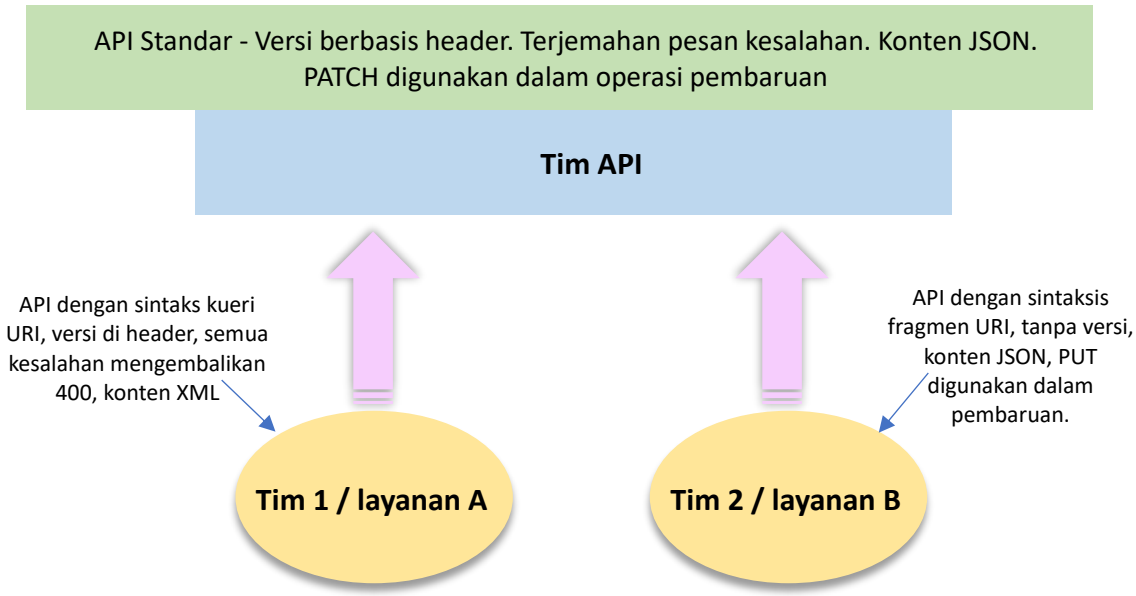
Gambar 3.4 UI Swagger yang Dihasilkan untuk PUT

Sumber lengkap dari Contoh Mulai Cepat API MassRover dapat ditemukan di repo ini: `https://github.com/thuru/MassRoverAPI`

Orientasi Tim dalam Pengembangan API

Implementasi API memerlukan pengetahuan bisnis dan keterampilan teknis. Ketika beberapa tim bekerja sama untuk menghasilkan layanan, tim API lintas sektor secara horizontal diperlukan untuk menggabungkan titik akhir yang berbeda di bawah satu pengalaman API. Tim API umumnya digunakan dalam implementasi arsitektur berbasis layanan mikro.

Dalam lingkungan layanan mikro, tim yang berbeda bekerja pada layanan yang berbeda, sehingga menciptakan API dengan standar yang berbeda. Aplikasi klien untuk pengguna bisnis yang mengakses layanan mikro ini memerlukan satu pengalaman API standar. Tim API memainkan peran penting dalam menstandardisasi beberapa aliran layanan di bawah satu saluran standar API. Gambar 3-5 menggambarkan skenario ini.



Gambar 3.5 Tim API Bekerja di Beberapa Tim

Ringkasan

Mengembangkan produk perangkat lunak yang lengkap memerlukan pertimbangan tertentu di awal untuk menghindari kebingungan dan menentukan standar pengembangan. Hal ini cukup umum dalam setiap proyek perangkat lunak, tetapi implementasi API memerlukan upaya yang lebih signifikan dalam mendefinisikan standar, dokumentasi, dan perjanjian pengembang ini, karena semuanya dikonsumsi oleh pihak eksternal.

Bab ini menyediakan panduan cepat dengan standar yang diperlukan dengan keseimbangan antara pengembangan cepat dan standar API. Sebagian besar standar yang dijelaskan dalam bab ini cukup untuk memulai pengembangan API. Nantinya dalam proses tersebut, pengembang harus mempertimbangkan standar dan praktik pengembangan API yang lebih kompleks seiring dengan perkembangan bisnis.

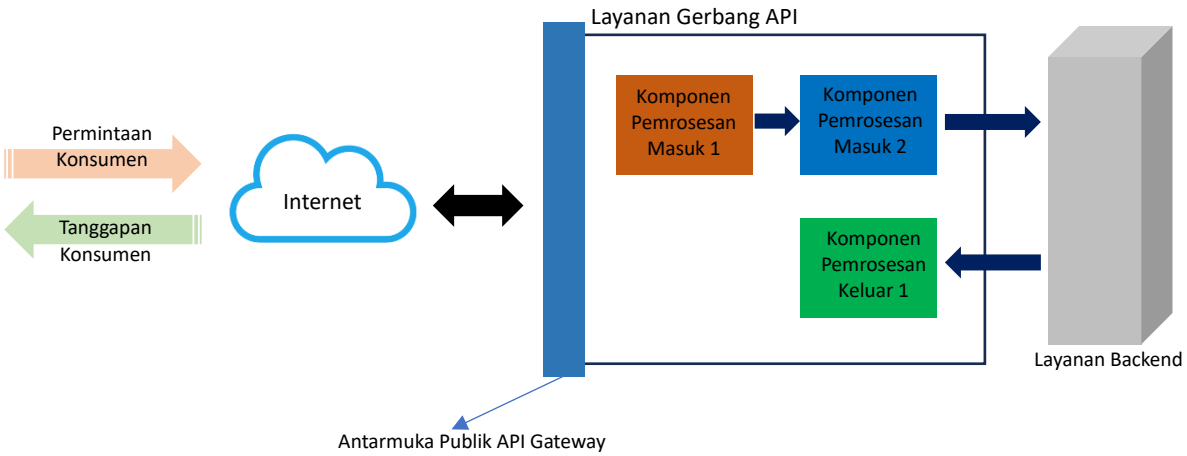
BAB 4
GERBANG API

Gerbang API merupakan lapisan terdepan untuk API, yang bertindak sebagai CDN. Meskipun tidak selalu beroperasi pada server tepi seperti CDN, fungsi utamanya adalah untuk mengabstraksi API atau layanan yang mendasarinya dan menyediakan titik akses yang seragam bagi konsumen. Selain itu, gerbang API menyediakan fitur nilai tambah terkait lainnya seperti penyimpanan sementara, keamanan, manajemen, negosiasi konten, dan manajemen kebijakan. Kumpulan fitur ini, yang ditumpuk dengan komponen arsitektur lain seperti pengalaman pengembang, integrasi perusahaan, telemetri, kebijakan akses dan permintaan, dan kontrol akses, membentuk layanan yang dikenal sebagai manajemen API.

Sebagian besar gerbang API komersial ditawarkan sebagai bagian dari alat dan layanan manajemen API. Bab ini membahas dua layanan manajemen API berbeda yang tersedia dari dua platform cloud publik utama: Manajemen API dari Azure dan Gerbang API dari AWS. Perhatikan bahwa Manajemen API dan Gerbang API merupakan nama masing-masing dari layanan manajemen API dari Azure dan AWS; keduanya merupakan nama produk dan tidak boleh disamakan dengan istilah umum "Gerbang API" dan "Manajemen API."

4.1 GATEWAY API DI CLOUD PUBLIK

Platform cloud publik modern, khususnya Azure dan AWS, memiliki banyak layanan dan penawaran platform. Manajemen API adalah salah satu layanan yang dijual oleh sebagian besar penyedia cloud publik.



Gambar 4.1 Tinjauan Umum API Gateway

Layanan gerbang API di cloud publik menerima permintaan dari Internet, paling umum melalui HTTP/HTTPS. Saat permintaan masuk ke gerbang API, layanan tersebut melakukan tindakan pada permintaan masuk, meneruskan permintaan ke layanan backend (atau terkadang tidak, tergantung pada aturan yang dikonfigurasi, seperti permintaan yang di-cache atau balasan

default), lalu melakukan tindakan dalam respons keluar. Gambar 4.1 menunjukkan skenario ini. Pada Gambar 4.1, konsumen membuat permintaan melalui Internet publik.

Ada berbagai kemungkinan jenis konsumen, termasuk aplikasi seluler, situs web, perangkat IoT, layanan API lainnya, dan konsumen manusia langsung. Layanan API gateway di cloud publik menerima permintaan dari Internet dan memproses permintaan tersebut berdasarkan konfigurasinya. Proses tersebut bertindak sebagai alur kerja, mirip dengan server web modern mana pun. Kami mengonfigurasi aturan statis dan dinamis dalam alur kerja dan, berdasarkan konfigurasi ini, berbagai komponen memproses permintaan sebelum mengirimkannya ke layanan backend. Tidak semua komponen pemrosesan mengubah permintaan beberapa berfungsi sebagai pemantauan dan pemeriksaan keamanan.

Beberapa komponen pemrosesan masuk mungkin tidak mengirimkan permintaan tertentu, tergantung pada validasi dan aturan yang ditetapkan. Setelah pemrosesan permintaan masuk selesai, jika validasi, aturan, dan konfigurasi memungkinkan, permintaan akan masuk ke layanan backend. Layanan backend melihat permintaan yang diproses, bukan permintaan asli yang dibuat oleh konsumen. Selain itu, gateway API memutuskan titik akhir backend mana yang akan dikirimi permintaan hal ini tidak dikontrol oleh konsumen. Layanan backend memberikan respons (baik berhasil atau gagal) ke gateway API, dan komponen pemrosesan keluar gateway API menerapkan aturan dan validasi yang dikonfigurasi dan mengirimkan respons ke konsumen.

Gateway API memiliki kontrol penuh atas permintaan yang diterimanya, dan memiliki kontrol yang sama atas respons yang diberikannya kepada konsumen. Gambar 4.1 menggambarkan ikhtisar mendasar gateway API di platform cloud publik. Internal implementasi, konfigurasi API, menghubungkan layanan backend, dan konfigurasi aturan bervariasi berdasarkan implementasi khusus vendor. Di bagian berikutnya, kita akan melihat detail implementasi menggunakan Azure API Management dan AWS API Gateway. Sebagai contoh, kita akan menggunakan API produk MassRover sebagai backend.

Pemetaan Titik Akhir

Seperti yang digambarkan pada Gambar 4.1, gateway API merupakan batas untuk permintaan klien, dan berada di antara layanan backend dan permintaan. Dalam model ini, gateway API dapat memiliki pemetaan yang berbeda antara titik akhir layanan backend dan titik akhir antarmuka gateway API. Bagian ini menjelaskan berbagai kemungkinan pemetaan dan pola yang terkait dengan titik akhir ini.

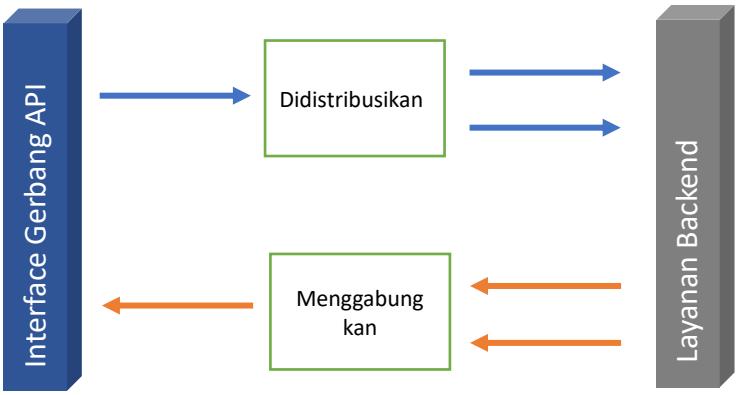
Pemetaan Satu-ke-Satu

Pemetaan satu-ke-satu bersifat langsung: satu layanan backend dipetakan ke satu titik akhir antarmuka gateway API. Gambar 4.2 menggambarkan hal ini.

Gambar 4.2 Pemetaan Satu-ke-Satu

Pemetaan Satu-ke-Banyak

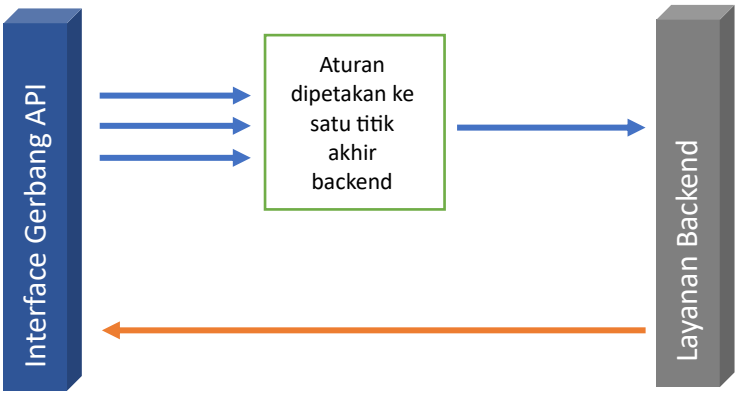
Dalam kasus ini, satu titik akhir antarmuka gerbang API dipetakan ke banyak titik akhir layanan backend yang berbeda. Pemetaan ini digunakan dalam pola komposisi API untuk mengurangi perjalanan pulang pergi dalam mengambil data, seperti informasi ulasan produk dan detail produk, dalam satu panggilan. Gambar 4.3 menggambarkan hal ini.



Gambar 4.3 Pemetaan Satu-ke-Banyak

Pemetaan Banyak-ke-Satu

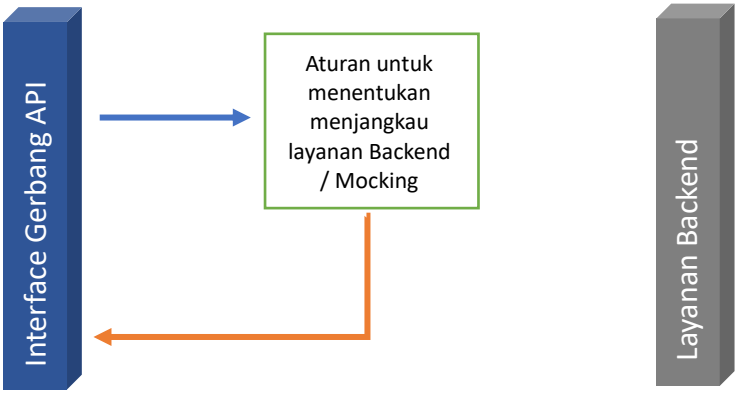
Pemetaan ini digunakan untuk mengabstraksikan operasi langsung dari layanan backend dan memberikan makna bisnis. Banyak titik akhir antarmuka gateway API dipetakan ke satu titik akhir layanan backend. Misalnya, suatu entitas memiliki status yang berbeda, dan setiap perubahan status secara teknis merupakan operasi pembaruan. Dalam konteks bisnis, setiap perubahan status dapat menjadi operasi yang berbeda dalam hal otorisasi dan makna semantik, sehingga setiap antarmuka gateway API memiliki titik akhir yang berbeda yang diekspos dengan pola URL yang bermakna, tetapi dipetakan ke titik akhir satu pembaruan dari layanan backend. Gambar 4.4 menggambarkan hal ini.



Gambar 4.4 Pemetaan Banyak-ke-Satu

Pemetaan Satu-ke-Tidak Ada

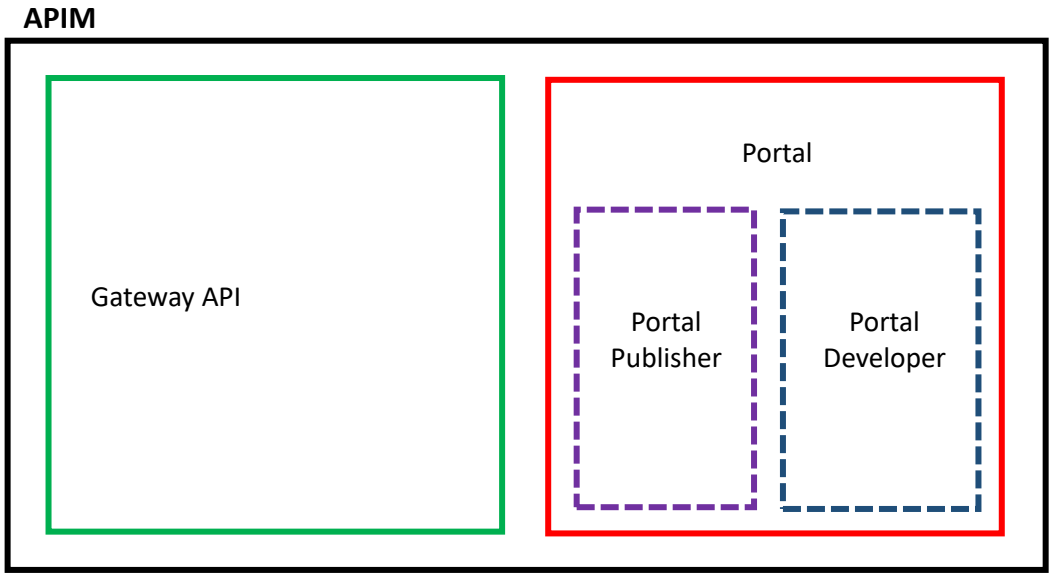
Gateway API bagus dalam menangani permintaan, dan mampu menjalankan beberapa logika tetap dan mengembalikan respons tanpa terhubung ke layanan backend. Sering kali, pemetaan ini digunakan dalam tiruan selama pengembangan. Juga, dalam beberapa kasus, meskipun titik akhir dipetakan ke layanan backend, gateway API dapat menghasilkan respons tanpa menghubungi layanan tersebut. Gambar 4.5 menggambarkan hal ini.



Gambar 4.5 Pemetaan Satu-ke-Tidak

4.2 AZURE API MANAGEMENT

Di bagian ini, kita akan melihat cara mengimplementasikan gateway API menggunakan layanan Azure API Management. API Management dibundel dengan API Gateway dan fitur manajemen API lainnya, seperti portal pengembang, keamanan, katalog API, dan penyimpanan sementara. Azure API Management memiliki dua komponen: API Gateway dan portal. Portal menyediakan dua pengalaman berbeda, penerbit dan pengembang. Secara tradisional, kedua pengalaman ini disampaikan melalui dua aplikasi web berbeda, yang ditunjukkan pada Gambar 4.6.



Gambar 4.6 Komposisi Logis Layanan Manajemen API Azure

Saat membuat instans layanan Manajemen API Azure, Azure menyediakan gateway API dan portal. Gateway API adalah mesin inti, yang menerima permintaan, memprosesnya, menghubungkan ke layanan backend, dan menanggapi permintaan. Portal penerbit menyediakan antarmuka administrasi untuk mengonfigurasi gateway API dan portal pengembang. Portal pengembang mencakup antarmuka dan alur kerja untuk orientasi pengembang, langganan API, dan fitur terkait pengalaman pengembang lainnya. Sebagai pengembang API, Anda akan menghabiskan banyak waktu di portal penerbit untuk mengonfigurasi gateway API dan portal pengembang.

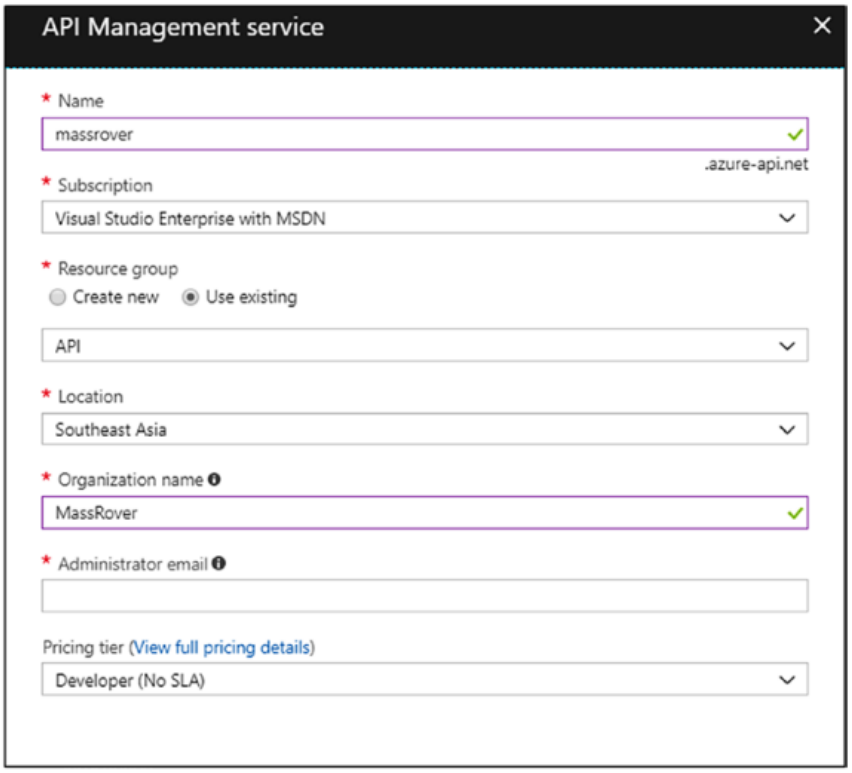
Membuat Layanan Manajemen API Azure

Buka langganan Azure Anda dan cari "Manajemen API", lalu pilih Manajemen API untuk membuat instans layanan. Anda akan melihat bilah pembuatan layanan Manajemen API Azure seperti yang ditunjukkan pada Gambar 4.7.

Ikuti langkah-langkah berikut di Azure API Management creation blade.

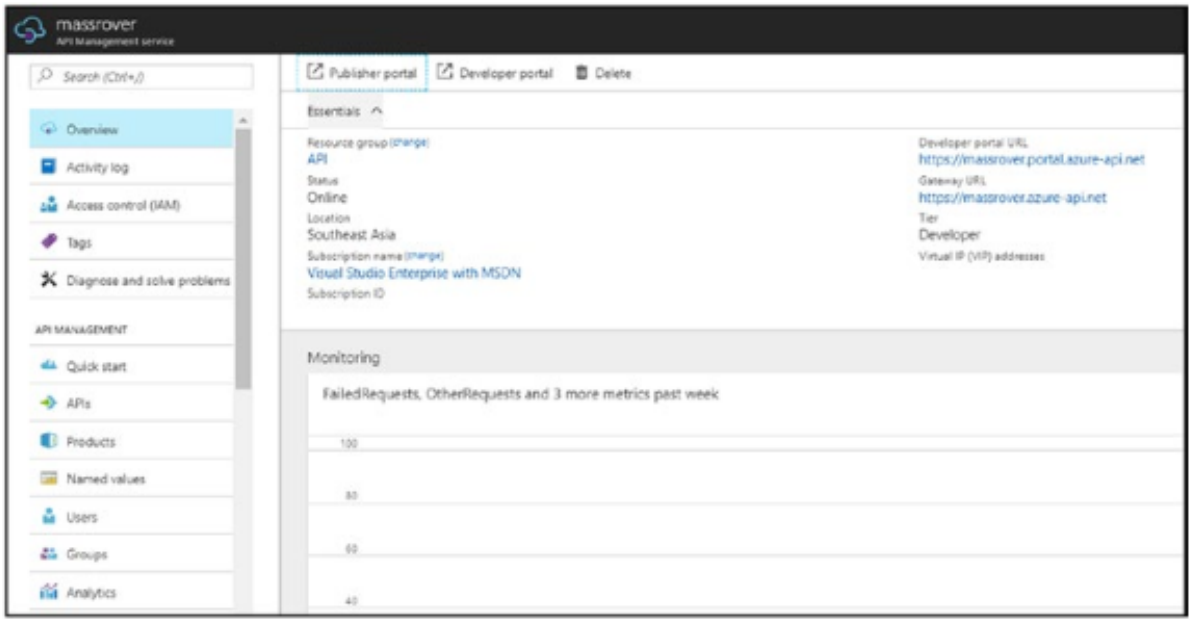
1. Berikan nama. Nama ini menetapkan URL gateway dan portal API. Nantinya, Anda dapat mengonfigurasi DNS untuk URL ini. URL gateway API muncul sebagai yourname.azure-api.net, dan URL portal muncul sebagai yourname.portal.azure-api.net. Menambahkan "`/ admin`" ke ini akan membuka portal penerbit.
2. Pilih langganan dan grup sumber daya (atau buat grup sumber daya baru), dan pilih lokasi.
3. Tentukan nama organisasi. Nama ini akan muncul di portal pengembang sebagai organisasi yang menerbitkan API.
4. Tentukan alamat email administrator. Pengguna yang membuat instans layanan akan menjadi administrator default, jadi sebaiknya berikan alamat email pengguna ini hingga Anda ingin orang lain bertindak sebagai administrator.
5. Pilih tingkatan harga. Tingkatan Pengembang adalah penawaran yang paling lengkap, dengan batasan permintaan/respons yang memadai dalam skenario

pengembangan/pengujian. Setelah melengkapi formulir, Anda dapat membuat instans layanan Azure API Management.



Gambar 4.7 Azure API Management Creation Blade

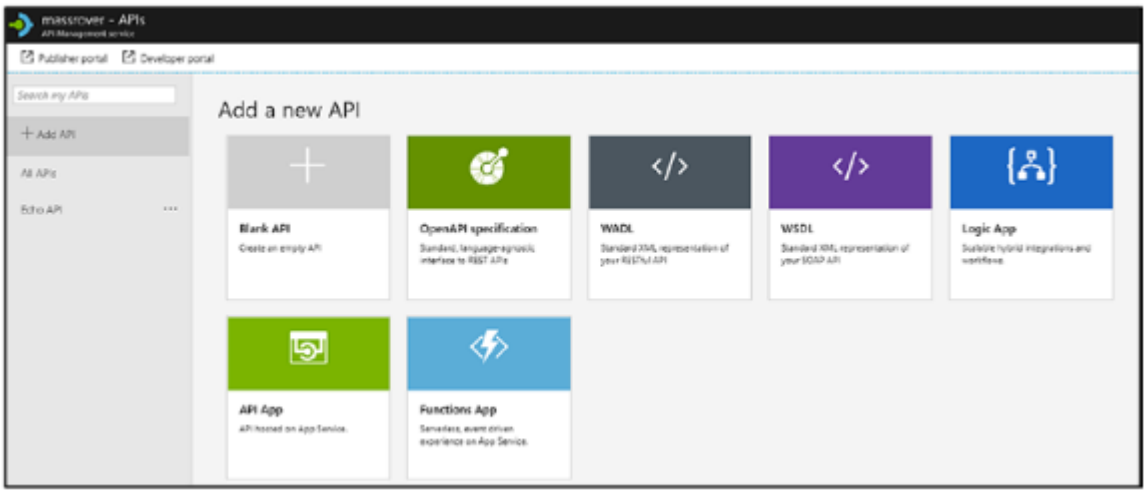
Gambar 4.8 memperlihatkan instans layanan Azure API Management segera setelah dibuat.



Gambar 4.8 Bilah Tinjauan Umum Azure API Management

Anda mungkin memperhatikan bahwa tautan untuk portal penerbit dan pengembang muncul di bagian atas. Selain itu, bagian tajuk bilah tinjauan umum menampilkan URL portal pengembang dan gateway API. URL portal penerbit sama dengan URL portal pengembang; Anda harus menambahkan "/admin" di bagian akhir. Catatan Portal Azure menawarkan dua pengalaman yang berbeda.

Saat Anda mengeklik tautan portal penerbit dan portal pengembang di bilah tinjauan umum, portal ini akan dibuka (selama masih berlaku) di tab yang berbeda. Namun, Microsoft Azure sedang dalam proses menciptakan pengalaman di portal.azure.com. Di masa mendatang, kita dapat berasumsi bahwa portal.azure.com akan menjadi ruang kerja utama untuk portal penerbit dan pengembang, menggantikan portal penerbit. Buku ini merujuk ke pengalaman di portal.azure.com sesering mungkin.

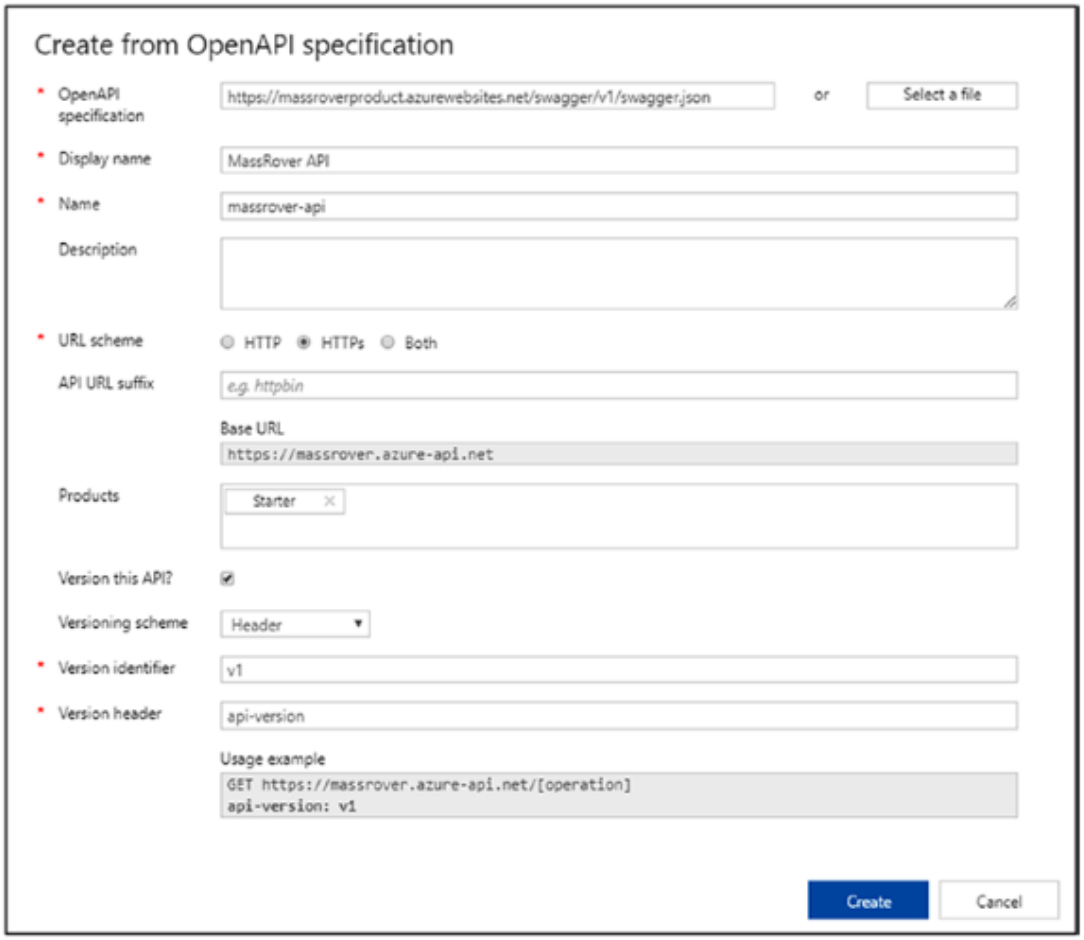


Gambar 4.9 API Blade

Di bilah tinjauan umum, yang ditunjukkan pada Gambar 4.8, di sisi kiri, Anda akan melihat beberapa item menu berbeda di bawah bagian Manajemen API. Item ini menentukan struktur layanan Manajemen API Azure. Klik opsi API ini akan membuka bilah API seperti yang ditunjukkan pada Gambar 4.9. API blade ini adalah titik awal utama untuk membuat API di layanan Azure API Management, dengan beberapa opsi yang tersedia. Dalam hal Azure API Management, API adalah kumpulan titik akhir yang mungkin terhubung atau tidak ke layanan backend. API juga dapat menyertakan titik akhir dari lebih dari satu layanan backend, dan satu layanan backend juga dapat disertakan dalam beberapa API. Pemetaan titik akhir layanan backend ke gateway API dibahas di bagian "Pemetaan Titik Akhir" dalam bab ini.

Menghubungkan ke Layanan Backend

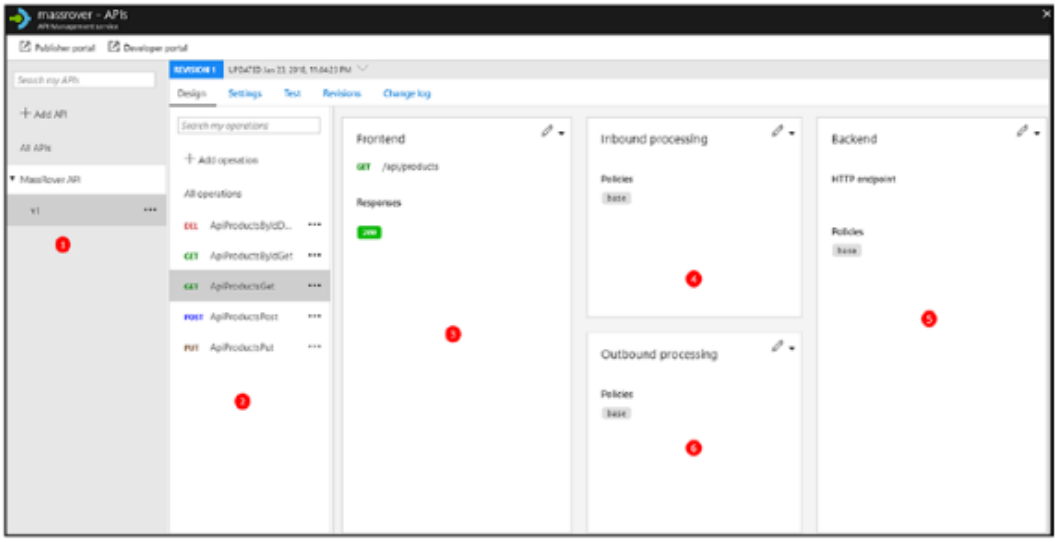
Pertama, kita harus membuat layanan backend agar layanan Azure API Management dapat terhubung; kita akan menggunakan MassRover API. Pastikan untuk meng-hosting MassRover API sehingga dapat diakses dari layanan Azure API Management.



Gambar 4.10 Buka Popup Spesifikasi API

Dengan asumsi MassRover API di-hosting di layanan aplikasi Azure API, mari impor menggunakan spesifikasi Open API. Karena kita telah mengonfigurasi spesifikasi Open API dalam implementasi MassRover API, kita dapat melakukannya dengan GUI portal. Seperti yang ditunjukkan pada Gambar 4.9, klik pada petak spesifikasi "Open API" untuk memulai proses. Azure API Management menawarkan berbagai opsi untuk mengimpor dan membuat API. Bab ini berfokus pada pembuatan API dari spesifikasi Open API; ini adalah praktik yang umum digunakan dalam pengembangan modern. Ini akan membuka jendela pop-up, seperti yang ditunjukkan pada Gambar 4.10. Anda dapat menentukan URL definisi Swagger dari API MassRover yang dihosting atau mengunggah file JSON.

Isi nama tampilan dan nama internal API. Anda juga dapat menyertakan deskripsi jika Anda menginginkannya. Pilih protokol URL—HTTP, HTTPS, atau keduanya. HTTPS ideal untuk keamanan dan kompresi konten. Sufiks URL API bersifat opsional; URL dasar akan berubah berdasarkan sufiks Anda. Kemudian, pilih produk. Di Azure API Management, produk adalah paket API yang dapat dikonsumsi. Kita akan membahasnya lebih rinci nanti di bab ini, tetapi pada titik ini Anda akan memilih "Produk Awal" (salah satu produk yang telah ditentukan sebelumnya).



Gambar 4.11 Ruang Kerja Konfigurasi API (Portal Baru)

Kita dapat membuat versi API dengan memilih kotak centang. Merupakan praktik yang baik untuk membuat versi API Anda dari awal, meskipun backend tidak mendukung ini. Pilih skema pembuatan versi ("Header" dipilih dalam contoh ini). Selanjutnya, berikan pengenal versi. Ini dapat berupa nomor versi utuh, tanggal, waktu, atau nilai string lainnya (nomor versi utuh digunakan dalam contoh ini, dengan awalan "v"). Terakhir, berikan kunci header untuk versi tersebut pengaturan umumnya adalah "api-version," yang digunakan dalam contoh ini. Setelah melengkapi formulir, klik "Buat" untuk mengimpor API MassRover ke Azure API Management.

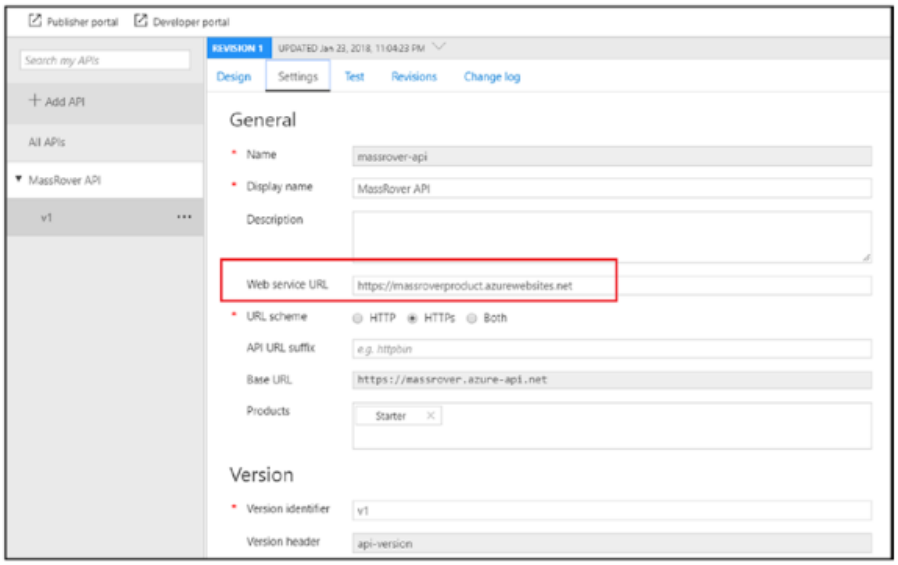
Sekarang gateway tersebut menjadi fronting layanan backend MassRover. Setelah langkah ini, Anda akan melihat API MassRover di bagian API (selain API Echo, yang disediakan secara default). Gambar 4.11 menunjukkan hal ini. Di sisi kiri, Anda dapat melihat API yang dibuat beserta versinya. Di sisi kanan, Anda akan melihat daftar titik akhir dan ruang kerja terkait. Titik akhir ini adalah titik akhir yang tersedia di layanan backend (API MassRover). Kita dapat menambahkan, mengedit, atau menghapus titik akhir sesuai kebutuhan.

1. API di Azure API Management Service. Masing-masing memiliki subbagian khusus untuk setiap versi.
2. Titik akhir dari versi API yang dipilih.
3. Panel konfigurasi front-end.
4. Bagian konfigurasi pemrosesan masuk.
5. Bagian konfigurasi layanan backend.
6. Bagian konfigurasi pemrosesan keluar.

4.3 MENGONFIGURASI TITIK AKHIR API

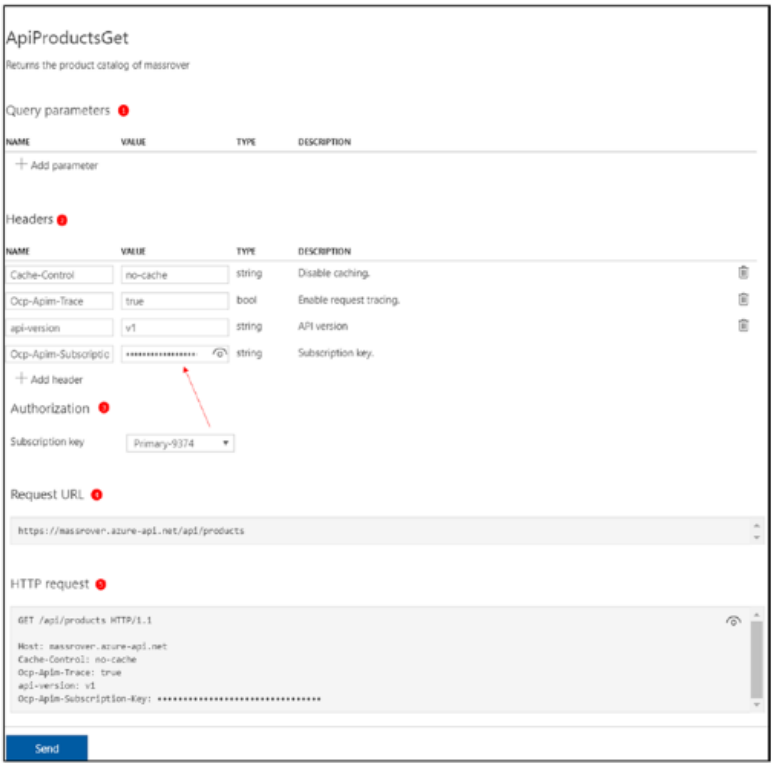
Setelah mengimpor API menggunakan Spesifikasi API Terbuka, kita perlu menghubungkan layanan backend agar API dapat berfungsi. Kita dapat menyediakan URL layanan backend dalam pengaturan API. Saat Anda memilih versi API, sebelum memilih titik akhir apa pun, navigasikan ke pengaturan dari menu atas, seperti yang ditunjukkan pada

Gambar 4.12.



Gambar 4.12 Mengonfigurasi URL Layanan Backend

Di sana, Anda dapat mengisi URL layanan backend. Ini harus menjadi URL dasar dari MassRover API yang dihosting. Perhatikan bahwa URL ini harus dapat diakses oleh layanan Azure API Management. Simpan pengaturan untuk mempertahankan perubahan. Karena implementasi API MassRover mematuhi praktik API standar dan Spesifikasi API Terbuka, pemetaan titik akhir menjadi mudah. Klik tab Uji untuk menguji gerbang API.



Gambar 4.13 Konsol Uji Admin Azure API Management

Gambar 4.13 memperlihatkan tab Uji dengan titik akhir `ApiProductsGet` yang dipilih. Ini adalah titik akhir yang dipetakan untuk "mendapatkan semua produk" di layanan backend. Seperti yang dapat Anda lihat pada Gambar 4-13, bilah uji ini memiliki beberapa bagian:

1. *Parameter kueri:* Tetapkan parameter kueri untuk URL permintaan di bagian ini.
2. *Header:* Tetapkan header permintaan. Anda dapat melihat beberapa header yang telah ditetapkan sebelumnya di bagian ini. Salah satunya adalah "versi-api", yang dikonfigurasi pada langkah sebelumnya. Header lainnya, terutama "ocp-apim-trace", menginstruksikan URL jejak untuk permintaan tersebut. Selama pengujian, sebaiknya tetapkan ini sebagai "benar", karena Azure API Management akan menghasilkan URL sementara dari log jejak yang dapat digunakan di luar portal. Header lainnya adalah "`opsi ocp-apim-subscription-key`". Nilai header ini memiliki autentikasi ke layanan Azure API Management. Setiap konsumen harus memiliki kunci langganan untuk melakukan panggilan ke Azure API Management. Kami akan membahas kunci langganan ini nanti di bab ini.
3. *Otorisasi:* Ini adalah bagian tempat kita memilih kunci langganan. Kunci ini dibuat secara otomatis oleh layanan Azure API Management, dan kita dapat memilih kunci primer atau sekunder. Dua kunci tersedia untuk mendukung fallback selama siklus kunci.
4. *URL Permintaan:* Anda dapat melihat URL permintaan di sini. Jika Anda menambahkan parameter kueri, Anda akan melihat bahwa bagian pratinjau ini berubah secara dinamis.
5. *Permintaan HTTP:* Bagian ini mempratinjau permintaan sebelum dikirim.

Klik tombol Kirim di bagian bawah bilah konsol pengujian. Ini akan mengirim permintaan ke API MassRover yang dihosting. Karena header `ocp-apim-trace` disetel ke true, kita dapat melihat URL hasil pelacakan di header respons "`ocp-apim-trace-location`." Informasi yang sama terstruktur di tab Pelacakan. Gambar 4.14 menunjukkan pesan respons di portal.

Gambar 4.14 Pesan Respons Pengujian Admin Azure API Management

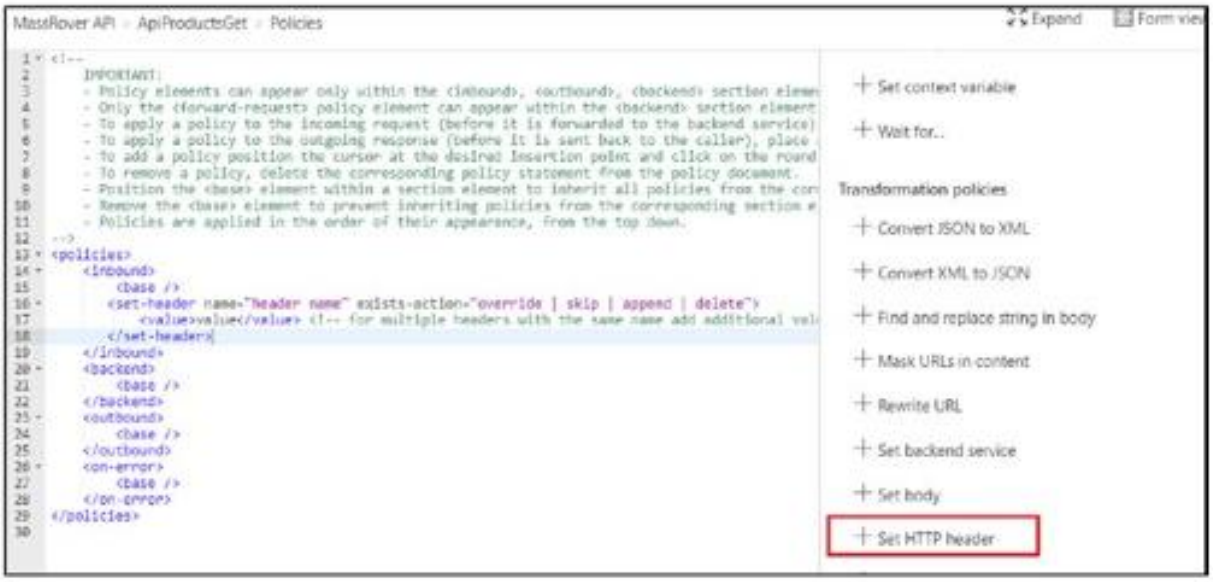
4.3 KEBIJAKAN KONFIGURASI

Selanjutnya, mari konfigurasikan beberapa kebijakan untuk API menggunakan tab Desain. Kita dapat mengonfigurasi titik akhir individual dengan pengaturan yang lebih disesuaikan menggunakan aturan masuk dan keluar. Klik pada titik akhir, lalu klik bagian pemrosesan masuk (#4 pada Gambar 4.11). Di sudut kanan, Anda akan melihat ikon Edit. Portal Azure menyediakan dua mode edit yang berbeda editor berbasis formulir, yang menawarkan fungsionalitas yang lebih sedikit tetapi masih cukup praktis untuk sebagian besar pengeditan cepat, dan editor berbasis kode, tempat kita dapat menetapkan kebijakan berdasarkan cuplikan kebijakan XML.

Gambar 4.15 menunjukkan editor kode, dan di sisi kiri Anda dapat melihat daftar aturan yang dapat digunakan. Di bagian ini, mari hapus header api-version dari permintaan, karena ini belum diimplementasikan dalam layanan backend (mengirim ini di header tidak akan membahayakan). Kita juga akan menetapkan kebijakan caching untuk titik akhir `ApiProductsGet`.

Gambar 4.15 Editor Kode Kebijakan Masuk

Menavigasi ke editor kebijakan berbasis kode dari bagian aturan masuk, XML kebijakan adalah satu dokumen dengan semua bagian kebijakan yang disertakan masuk, keluar, backend, dan kesalahan. Di bagian masuk, klik kebijakan Set HTTP Header di bawah bagian Kebijakan transformasi. Ini akan menyuntikkan templat cuplikan XML Set HTTP Header. Lihat Gambar 4.16.



Gambar 4.16 Cuplikan XML Kebijakan (Tetapkan Header HTTP)

Di bawah ini adalah templat cuplikan XML.

```
<set-header name="header name" exists-action="override | skip | append | delete">
   <value>value</value>
</set-header>
```

Edit templat dengan memasukkan `nama header = api-version dan exists-action = delete`. Karena tindakannya adalah delete, kita tidak memerlukan elemen value. Setelah penggantian, kebijakan akan muncul sebagai berikut:

```
   <set-header name="api-version" exists-action="delete">
   </set-header>
```

Selanjutnya, kita akan menetapkan kebijakan penyimpanan sementara untuk titik akhir ini selama 60 detik. Layanan Azure API Management akan menyimpan sementara respons selama 60 detik dan menanggapi permintaan dengan latensi rendah. Aturan penyimpanan sementara harus ditetapkan dengan dua kebijakan. Saat permintaan masuk, kita harus memeriksa cache untuk nilai yang disimpan sementara, dan untuk ini kita memerlukan kebijakan pencarian cache di bagian masuk. Saat respons diterima dari layanan backend, kita harus menyimpan nilai permintaan tertentu untuk ini, kita memerlukan kebijakan penyimpanan cache di bagian keluar.

Dalam kebijakan pencarian cache, kita akan mengatur untuk mencari cache, di bawah konteks permintaan yang ditentukan. Dalam konfigurasi kebijakan di bawah ini, konteks permintaan disimpan sementara untuk semua pengembang di semua grup pengembangan (pengalaman pengembang dijelaskan di bagian berikutnya). Atribut header Accept dan header Accept-Charset akan bervariasi, yang berarti entri cache individual ditetapkan untuk nilai yang bervariasi untuk header tersebut. Kita juga dapat menetapkan sejumlah nilai header dan string

kueri.

```
<cache-lookup vary-by-developer="false" vary-by-developergroups="false">
  <vary-by-header>Accept</vary-by-header>
  <vary-by-header>Accept-Charset</vary-by-header>
</cache-lookup>
```

Kebijakan penyimpanan cache ditetapkan dengan periode waktu habis. Durasi cache default adalah 60 detik, dengan maksimum 2.592.000 detik (satu bulan) untuk kebijakan ini.

```
  <cache-store duration="60"/>
```

Selanjutnya, mari kita manipulasi header respons. Diasumsikan aman untuk menyembunyikan implementasi dan detail server dari layanan backend, jadi mari kita tambahkan kebijakan ke aturan keluar untuk menghapus nilai header "X-powered-By" dari layanan backend.

```
  <set-header name="X-Powered-By" exists-action="delete">
  </set-header>
```

Setelah kita mengonfigurasi semua kebijakan, rangkaian kebijakan lengkap yang berlaku dapat diverifikasi di portal menggunakan opsi "Hitung kebijakan efektif". Dengan menggunakan opsi ini, Anda akan melihat kebijakan tambahan yang berasal dari kebijakan cakupan produk. Kita akan melihat produk-produk ini di bagian selanjutnya.

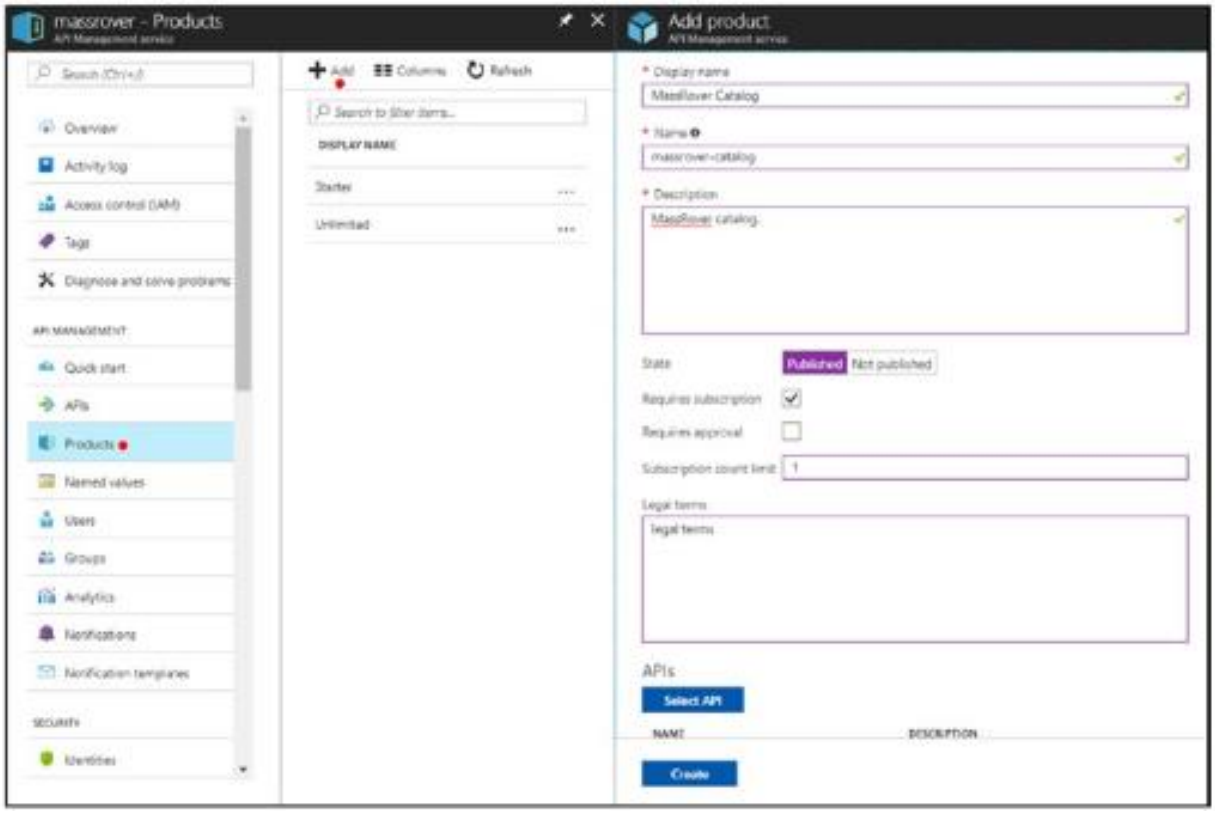
Di bawah ini adalah rangkaian total kebijakan yang dikonfigurasi khusus untuk titik akhir yang dipilih.

```
  <policies>
    <inbound>
    <base />
    <set-header name="api-version" exists-action="delete" />
    <cache-lookup vary-by-developer="false" vary-bydeveloper-groups="false">
    <vary-by-header>Accept</vary-by-header>
    <vary-by-header>Accept-Charset</vary-by-header>
    </cache-lookup>
    </inbound>
    <backend>
      <base />
    </backend>
    <outbound>
      <base />
      <cache-store duration="60" />
      <set-header name="X-Powered-By" exists-action="delete" />
    </outbound>
    <on-error>
      <base />
    </on-error>
  </policies>
```

Anda dapat menguji beberapa titik akhir dan merasakan efek dari kebijakan yang dikonfigurasi. Layanan Azure API Management memiliki serangkaian kebijakan yang kaya tempat logika pengambilan keputusan yang bersyarat dan kompleks dapat dikonfigurasi. Azure API Management juga mendukung penambahan kebijakan ini melalui C#. Anda dapat membaca selengkapnya tentang kebijakan ini di sini: https://docs.microsoft.com/en-us/azure/api-management/api-management-policies

4.4 PRODUK DI AZURE API MANAGEMENT

Produk adalah paket habis pakai di Azure API Management. Produk berisi API, dan satu produk dapat memiliki beberapa API. Kita dapat membuat produk menggunakan portal, dan setiap produk memiliki pengaturan yang berbeda. Secara default, layanan Azure API Management disediakan dengan dua produk: Dimulai dan Tidak Terbatas. Mari buat produk dan lihat opsi apa saja yang tersedia. Navigasi ke bagian Produk dan klik Tambahkan untuk membuat produk baru (lihat Gambar 4.17).



Gambar 4.17 Blade Produk Azure API Management

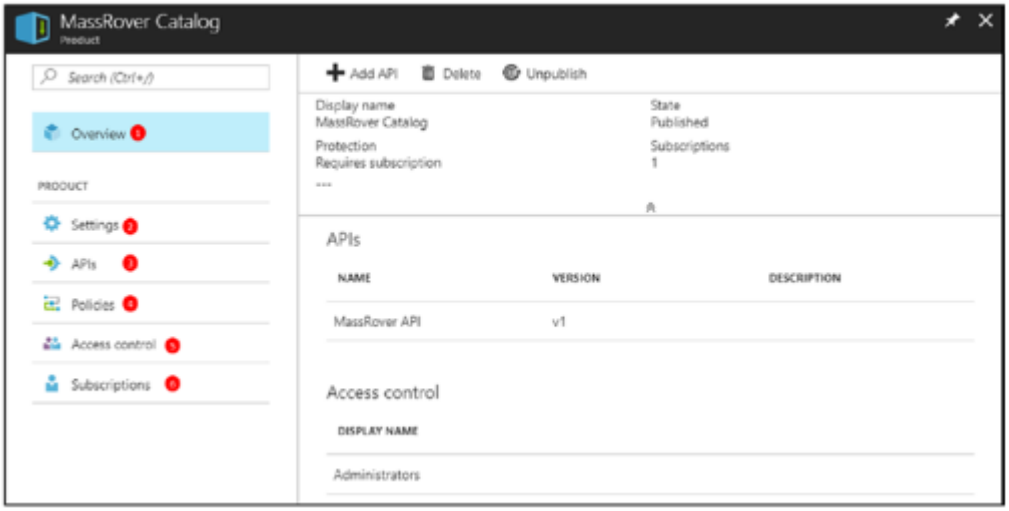
Produk memiliki properti berikut:

1. *Nama tampilan:* Nama yang dapat dilihat publik, digunakan di portal, di antara tempat lainnya.
2. *Nama:* Nama sumber daya Azure, ini tidak dapat diedit nanti.
3. *Deskripsi:* Deskripsi singkat produk. Ini adalah properti wajib.
4. *Status:* Produk dapat dipublikasikan atau tidak dipublikasikan. Hanya produk yang

dipublikasikan yang tersedia di portal pengembang. Status tidak dipublikasikan dianalogikan dengan mode draf atau mode nonaktif. Secara default, status tidak dipublikasikan, yang berarti produk hanya akan tersedia untuk administrator.

5. *Memerlukan langganan*: Menunjukkan apakah langganan pengembang diperlukan untuk menggunakan produk.
6. *Memerlukan persetujuan:* Menunjukkan apakah permintaan langganan dari pengembang (yang dibuat melalui portal pengembang) memerlukan persetujuan dari administrator di portal penerbit.
7. *Tingkat jumlah langganan:* Jumlah langganan untuk produk ini yang dapat diberikan kepada satu pengembang. Satu langganan per pengembang sudah cukup dalam sebagian besar kasus.
8. *Ketentuan hukum:* Ketentuan hukum opsional yang harus disetujui dan dipatuhi oleh pengembang untuk menggunakan API.

Tetapkan Status ke Diterbitkan dan centang "Memerlukan Langganan." Terakhir, pilih API yang akan disertakan dalam produk (kita dapat menambahkan lebih dari satu). Kita dapat menambahkan MassRover API v1 ke produk ini. Setelah kita membuat produk, produk tersebut akan terlihat di bilah produk. Klik pada produk Katalog MassRover yang baru dibuat untuk mengonfigurasinya. Gambar 4.18 menunjukkan bilah produk.



Gambar 4.18 Bilah Konfigurasi Produk

Di sini, kita dapat mengonfigurasi pengaturan untuk produk. Anda akan melihat sejumlah tab:

1. *Ikhtisar:* Bagian ini menunjukkan ikhtisar produk. Di bagian atas, kita dapat melihat opsi untuk mengubah status produk, dan kita dapat menambahkan atau menghapus API. Selain itu, bilah ini menunjukkan ringkasan kontrol akses.
2. *Pengaturan:* Bagian ini sama dengan bilah pembuatan produk. Di sini, kita dapat mengedit properti produk, kecuali nama sumber daya Azure.
3. *API:* Gunakan bagian ini untuk menambahkan atau menghapus API ke atau dari produk.
4. *Kebijakan:* Ini adalah konfigurasi kebijakan untuk produk. Sebelumnya, kita

mengonfigurasi kebijakan di tingkat titik akhir, dan di sini kita dapat mengonfigurasinya di tingkat produk, tempat kebijakan diterapkan ke semua titik akhir dari semua API yang disertakan dalam produk.

5. *Kontrol akses:* Di bagian ini, Anda dapat menambahkan atau menghapus grup yang memiliki akses ke produk.
6. *Langganan:* Di sini, Anda dapat menangguhkan, membatalkan, atau menghapus langganan ke produk.

Layanan Azure API Management memiliki tiga grup kontrol akses bawaan dengan izin yang ditentukan.

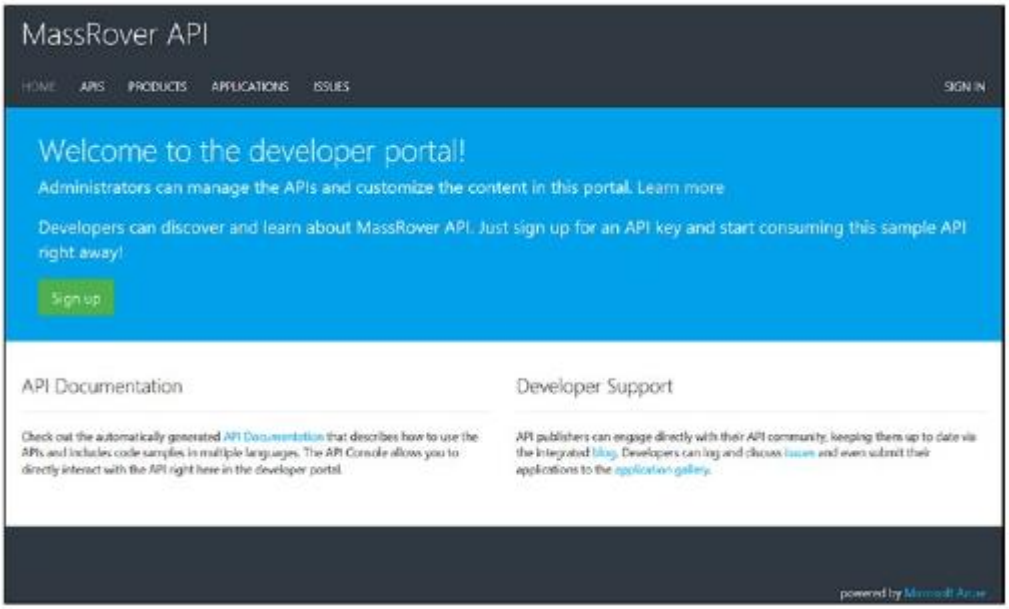
- *Administrator:* Administrator mengelola instans layanan Azure API Management, membuat API, operasi, dan produk yang digunakan oleh pengembang.
- *Pengembang:* Pengguna portal pengembang yang diautentikasi termasuk dalam grup ini. Pengembang adalah pelanggan yang membangun aplikasi menggunakan API Anda. Pengembang diberikan akses ke portal pengembang dan membangun aplikasi yang memanggil operasi API.
- *Tamu:* Pengguna portal pengembang yang tidak diautentikasi, seperti calon pelanggan yang mengunjungi portal pengembang instans Azure API Management, termasuk dalam grup ini. Mereka dapat diberikan akses baca-saja tertentu, seperti kemampuan untuk melihat API tetapi tidak memanggilnya.

Selain itu, kita dapat menambahkan grup kustom, tetapi level kontrol akses tetap berada dalam grup bawaan. Terutama saat menambahkan semua pengembang ke satu grup pengembang akan memaparkan semua produk ke semua pengembang. Untuk menghindari hal ini, kita dapat menggunakan grup kustom untuk mengelompokkan pengembang dan memberi mereka akses.

4.5 PENGALAMAN PENGEMBANG AZURE API MANAGEMENT

Sekarang kita memiliki sebuah produk, dan kita dapat mengirimkannya ke pengembang yang akan menggunakan API tersebut. Seperti yang dinyatakan, produk adalah paket habis pakai dari layanan Azure API Management. Pertama, navigasikan ke portal pengembang. Anda bisa mendapatkan URL portal pengembang dari bilah ikhtisar layanan Azure API Management, yang ditunjukkan pada Gambar 4.8. Salin URL dan navigasikan ke sana menggunakan sesi peramban pribadi. Ini akan mencegah Anda masuk ke portal pengembang sebagai administrator.

Portal pengembang mentah akan terlihat mirip dengan gambar tangkapan layar yang ditunjukkan pada Gambar 4.19. Di bagian atas, Anda akan melihat tab Beranda, API, Produk, Aplikasi, dan Masalah. Bagian API mencantumkan API dalam instans layanan Manajemen API Azure, dan Anda dapat melihat produk yang tersedia di tab Produk. Jika Anda mengeklik tab Produk, Anda akan melihat produk Dimulai dan Tidak Terbatas (yang merupakan produk default yang dibuat oleh layanan Manajemen API Azure selama penyediaan), tetapi bukan Katalog MassRover. Ini karena kami belum masuk ke portal pengembang, dan sesuai dengan kontrol akses produk, Katalog MassRover hanya dapat diakses oleh administrator.



Gambar 4.19 Portal Pengembang

Tab Aplikasi menunjukkan daftar aplikasi terdaftar yang menggunakan API yang berbeda. Tab Masalah adalah bagian pelaporan masalah pengembang. Secara keseluruhan, portal pengembang menyediakan pengalaman yang komprehensif, mulai dari autentikasi hingga langganan hingga katalogisasi aplikasi hingga pelaporan masalah. Sebagai pengguna anonim, Anda dapat melihat API dan produk (yang memiliki akses tamu yang diaktifkan).

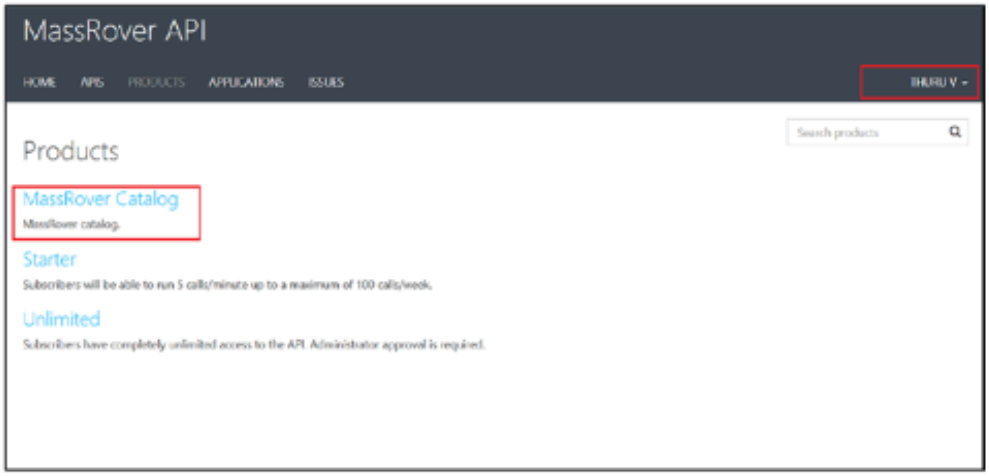
Jika Anda memiliki kunci langganan yang valid, Anda dapat menggunakannya dan menguji API tersebut. Akses tamu diaktifkan untuk calon konsumen API, dan mereka dapat melihat titik akhir dan dokumentasi yang tersedia. Mereka tidak dapat melakukan panggilan kecuali mereka memiliki kunci langganan yang valid. Dalam alur kerja yang umum, pengembang dapat mengakses portal pengembang dengan salah satu dari tiga cara.

1. Pengembang dapat mendaftar di portal pengembang itu sendiri menggunakan pendaftaran sederhana atau penyedia login yang dikonfigurasi, seperti Facebook atau Google. Ini dapat dikonfigurasi oleh administrator di portal Azure.
2. Administrator dapat menambahkan pengembang melalui portal menggunakan autentikasi dasar. Dalam hal ini, administrator menetapkan nama pengguna dan kata sandi untuk pengembang. Pengembang dapat mengubah kata sandi nanti.
3. Administrator dapat mengirim undangan ke pengembang. Administrator melengkapi profil dasar dan undangan akan dikirim ke pengembang beserta tautan. Pengembang kemudian dapat menyelesaikan pendaftaran menggunakan tautan dan menetapkan kata sandi. Daftarkan diri Anda sebagai pengembang, atau kirim undangan dan masuk ke portal pengembang sebagai pengembang.

Untuk mengakses produk Katalog MassRover, kita perlu menambahkan grup pengembang. Anda dapat melakukan tindakan ini di bagian kontrol akses bilah konfigurasi produk (lihat Gambar 4.18). Setelah menambahkan grup pengembang ke produk, masuk ke portal pengembang menggunakan kredensial pengembang. Anda sekarang akan melihat Katalog

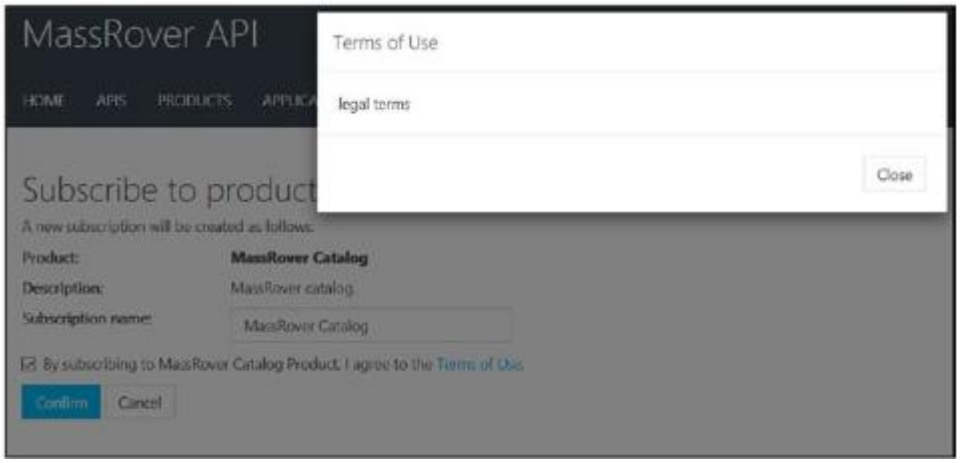
MassRover di bawah tab Produk (lihat Gambar 4.20). Sebagai pengembang, Anda dapat berlangganan produk ini untuk menggunakannya.

Alur kerja pengembang Azure API Management cukup komprehensif, dan dapat dikonfigurasi sesuai keinginan Anda. Email dikirim pada berbagai tahap alur kerja pengembang, dan templat email sepenuhnya dapat disesuaikan di portal. Pencitraan merek portal pengembang juga dapat disesuaikan. Buku ini sengaja tidak membahas area tersebut.



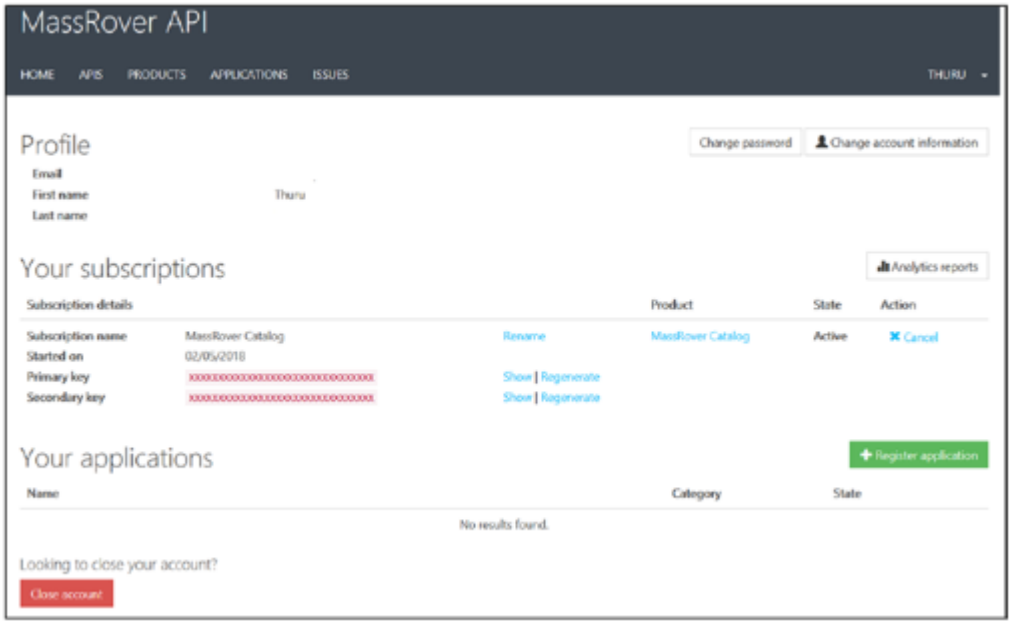
Gambar 4.20 Produk Portal Pengembang—Pengembang Masuk

Klik produk, terima syarat dan ketentuan, lalu klik berlangganan. Hal ini ditunjukkan pada Gambar 4.21.



Gambar 4.21 Berlangganan Produk Pengalaman Pengembang

Karena kami mengonfigurasi produk Katalog MassRover untuk menerima langganan tanpa persetujuan, Anda akan melihat produk tersebut di halaman pengembang segera setelah Anda berlangganan, seperti yang ditunjukkan pada Gambar 4.22.



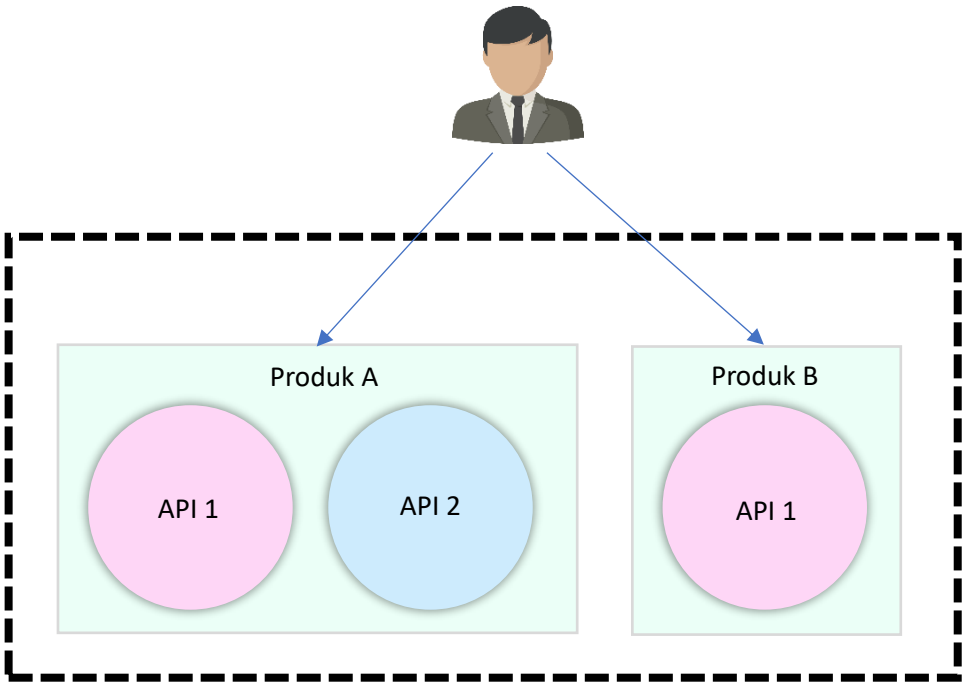
Gambar 4.22 Halaman Langganan Pengembang Pengalaman Pengembang

Pengembang dapat mengakses dan membuat kunci langganan di halaman langganan pengembang. Kunci tersebut harus dikirimkan di bawah tajuk ocp-apim-subscription-key. Kunci langganan berfungsi sebagai fitur keamanan tingkat pertama, tetapi yang lebih penting, kunci tersebut digunakan untuk melacak pengembang, penggunaan, dan konfigurasi kebijakan. Pengembang dapat menggunakan kunci langganan mereka dan menguji API di portal pengembang itu sendiri. Portal pengembang Azure API Management menyertakan konsol pengujian dengan contoh kode untuk menggunakan titik akhir. Pengembang juga dapat mendaftarkan aplikasi mereka menggunakan portal pengembang.

4.5 STRUKTUR KOMPONEN MANAJEMEN API AZURE

Dalam Manajemen API Azure, API adalah kumpulan titik akhir, yang mungkin atau mungkin tidak terhubung ke layanan backend. Selain itu, titik akhir ini juga dapat terhubung ke berbagai layanan backend. Setiap versi API diperlakukan sebagai API terpisah. Kebijakan dapat dikonfigurasi di tingkat API dan titik akhir. API ditautkan dengan produk, dan produk adalah paket yang dapat dikonsumsi. Kita juga dapat mengonfigurasi kebijakan di tingkat produk. Produk dapat mencakup beberapa API.

Pengembang dapat menggunakan produk yang dipublikasikan (titik akhir API yang disertakan dalam produk). Pengembang dapat memiliki lebih dari satu langganan untuk setiap produk. Setiap langganan diidentifikasi oleh kunci langganan, dan setiap kunci langganan dicakup ke suatu produk. Pengembang tidak dapat menggunakan satu kunci langganan untuk dua produk yang berbeda (lihat Gambar 4.23).



Gambar 4.23 Struktur Manajemen API Azure

Langganan diberikan melalui portal pengembang. Seperti yang diilustrasikan, pengembang memiliki dua langganan untuk Produk A dan satu langganan untuk Produk B. Di portal pengembang, Anda dapat mengonfigurasi alur kerja tambahan selama proses langganan, seperti proses masuk eksternal dan pemrosesan pembayaran.

AWS API Gateway

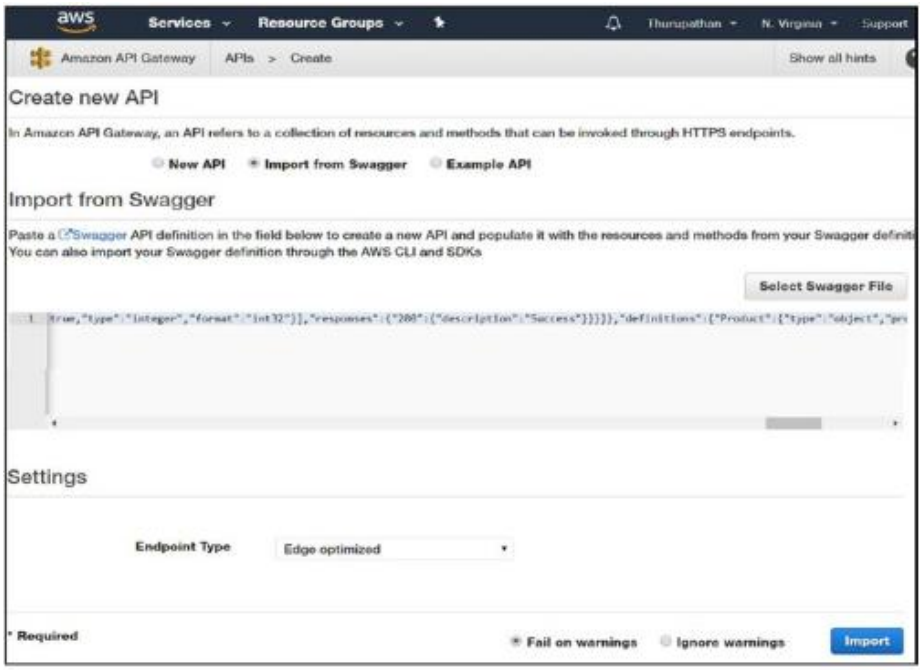
API Gateway adalah nama komersial dari layanan manajemen API yang ditawarkan di AWS. Anda dapat membuat instans AWS API Gateway di AWS. AWS API Gateway menawarkan fitur-fitur seperti caching, kontrol permintaan, autentikasi, tiruan, dan penerbitan API melalui AWS Marketplace. API Gateway terhubung erat dengan layanan AWS lainnya seperti VPC dan Lambda. Layanan ini juga memungkinkan pembuatan SDK klien untuk platform populer seperti iOS dan Android. Layanan ini memiliki alur antarmuka publik API Gateway, pemrosesan permintaan, koneksi ke layanan backend (atau tiruan), dan pemrosesan respons.

Membuat Layanan AWS API Gateway

Dengan asumsi Anda memiliki akun AWS dengan izin IAM untuk melakukan tindakan, masuk ke konsol AWS, cari API Gateway, dan buat instans AWS Gateway. Klik “Impor dari Swagger” dan salin definisi Swagger dari API MassRover di panel (Gambar 4.24), atau unggah berkas definisi. AWS juga memiliki contoh opsi implementasi API. Di bagian Setelan, pilih jenis titik akhir. Ada dua opsi untuk ini.

- *Edge optimized:* Ini adalah opsi default, yang mengaktifkan distribusi AWS Cloud Front dan meningkatkan waktu koneksi. Ini adalah pilihan yang baik dalam kebanyakan kasus. Permintaan API dari klien akan diarahkan ke server edge CloudFront terdekat di seluruh wilayah AWS (mirip dengan CDN).
- *Regional:* Opsi ini akan mengarahkan permintaan klien ke gateway API khusus wilayah, melewati distribusi CloudFront. Permintaan dari wilayah yang sama memiliki manfaat

untuk menghindari perjalanan pulang pergi yang tidak perlu ke CloudFront, tetapi permintaan dari wilayah lain mungkin memiliki latensi. Ini dapat dicapai dengan menyebarkan gateway API khusus wilayah di wilayah target.



Gambar 4.24 Penyediaan Gateway API AWS

Klik Impor untuk mengimpor definisi API MassRover. Setelah impor, Anda akan melihat API di panel. AWS API Gateway menyusun API sebagai sumber daya dan metode. Segmen API URI adalah sumber daya, dan tindakan HTTP disebut sebagai metode. Dalam segmen URI seperti api/products, API dan products adalah dua sumber daya yang berbeda. Suatu sumber daya dapat memiliki sumber daya dan metode lain, dan parameter jalur juga dapat ditambahkan sebagai sumber daya. Anda dapat menggunakan kurung kurawal untuk menunjukkan parameter jalur, seperti yang ditunjukkan pada Gambar 4.25. Dalam API MassRover, products dan parameter jalur (ID) adalah sumber daya yang berbeda dan memiliki metode di dalamnya. Metode dikaitkan dengan kata kerja HTTP.

Gambar 4.25 Struktur API di AWS API Gateway

Anda dapat memilih sumber daya dan menambahkan sumber daya atau metode ke dalamnya menggunakan menu tarik-turun Tindakan di bagian atas. Saat menambahkan parameter jalur sebagai sumber daya, gunakan kurung kurawal. Klik "method" untuk mengonfigurasinya.

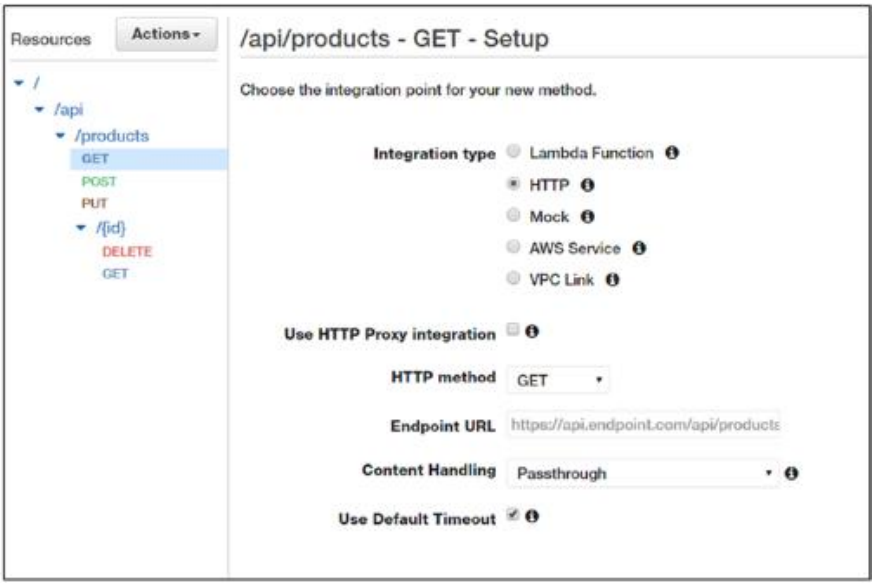
Konfigurasikan Metode

Karena kita hanya mengimpor definisi API, konfigurasi harus dilakukan. Ada beberapa opsi berbeda untuk mengonfigurasi layanan backend untuk suatu metode. Klik metode GET di bawah api/products, dan Anda akan melihat panel yang ditunjukkan pada Gambar 4.26. Ini adalah panel fase 1, dan kita harus memilih titik integrasi untuk metode yang dipilih. Dari konfigurasi dasar, pilih opsi HTTP dan tempel URL titik akhir produk API MassRover yang dihosting di kotak teks URL Titik Akhir. Ini akan melengkapi integrasi backend untuk metode yang dipilih.

Anda juga dapat memilih "tiruan" sebagai opsi dan kemudian terhubung ke layanan yang tepat. Penanganan Konten menawarkan tiga opsi berbeda untuk menangani isi permintaan dari metode tertentu.

- Pass-through: Ini adalah opsi default, digunakan saat tidak diperlukan konversi konten.
- Konversi ke biner: Ini digunakan saat layanan backend mengharuskan isi dalam format biner, saat permintaan asli dikirimkan dalam format base64.
- Konversi ke teks: Mengonversi permintaan isi input biner ke string base64. Ini digunakan saat layanan backend mengharuskan isi biner dalam format berbasis string.

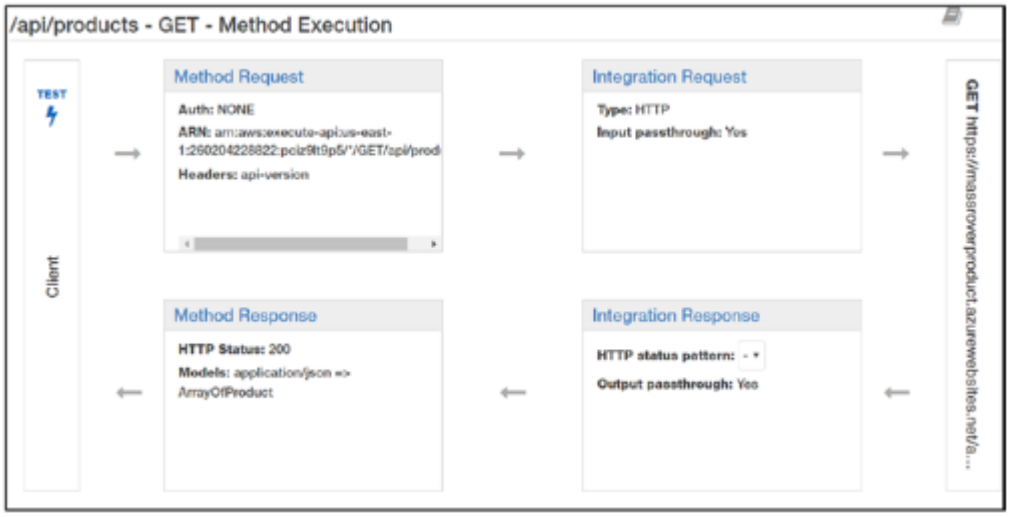
Integrasi proksi HTTP digunakan untuk menyederhanakan layanan backend sebagai titik masuk API tunggal. Kata kerja HTTP ANY digunakan dalam kasus ini, karena ini akan memungkinkan metode untuk menerima permintaan HTTP apa pun dari klien.



Gambar 4.26 Konfigurasi Metode API Panel 1

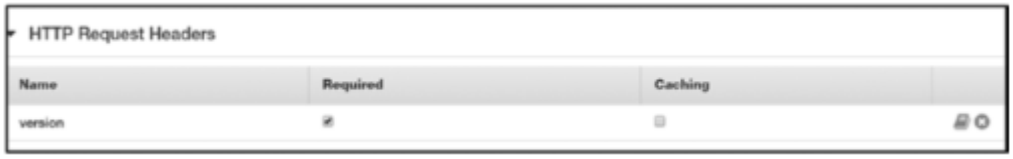
Setelah mengonfigurasi titik integrasi, kita dapat menguji metode tersebut. Klik metode tertentu, dan Anda akan melihat panel seperti yang digambarkan pada Gambar 4.27. Ini

menunjukkan aliran permintaan dan respons AWS API Gateway. Permintaan diterima oleh antarmuka publik AWS Gateway, lalu diteruskan ke permintaan metode, lalu ke proses permintaan integrasi, dan akhirnya ke layanan backend. Demikian pula, respons diterima dalam respons integrasi, diproses, dan diteruskan ke klien.



Gambar 4.27 Konfigurasi Metode API Panel 2

Dalam alur kerja ini, parameter jalur, parameter string kueri, nilai header, dan pemetaan isi muatan konten semuanya harus dipetakan antara permintaan metode dan permintaan integrasi. Pemetaan ini digunakan untuk menambahkan, mengedit, atau menghapus nilai tertentu. Jika klien mengirim permintaan dengan header "`version`" tetapi layanan backend mengharapkan header "`api-version`," maka ini harus dipetakan antara permintaan metode dan permintaan integrasi. Untuk melakukan ini, pertama-tama kita harus menentukan variabel permintaan metode. Klik pada Permintaan Metode dan perluas bagian Header Permintaan HTTP, yang ditunjukkan pada Gambar 4.28. Tambahkan header dari permintaan klien.



Gambar 4.28 Pemetaan Header Metode

Selanjutnya, kita akan membuat pemetaan nilai header ini dalam permintaan integrasi. Kembali ke panel alur kerja (Gambar 4.27). Klik Permintaan Integrasi dan perluas bagian Header Permintaan HTTP, yang ditunjukkan pada Gambar 4.29. Di sini, kita dapat mengatur pemetaan menggunakan sintaks pemetaan AWS API Gateway. Sintaks ini sangat mudah:

```
method.request.{path|header|querystring}.{parameter_name}.
```

Dalam kasus ini, kita harus memetakan header (versi) yang masuk ke versi api. Buat nilai header dan terapkan pemetaan berikut dalam kotak teks (ditunjukkan pada Gambar 4.29): method.request.header.version

Gambar 4.29 Pemetaan Header Permintaan Integrasi

Setelah integrasi, uji titik akhir metode. Anda akan memberikan nilai header bernama "versi" dalam permintaan. Kita dapat memeriksa pemetaan nilai header dalam log.

```
Method request headers: {version=v1}

Endpoint request headers: {api-version=v1, x-amzn-apigateway-api-
id=pciz9lt9p5, Accept=application/json, User-
Agent=AmazonAPIGateway_pciz9lt9p5, X-Amzn-Trace-Id=Root=1-5a8001d3-
ad377f517777323dd3897df2}
```

Untuk menghapus header yang diteruskan ke layanan backend, Anda cukup menghapus pemetaan dalam permintaan integrasi, seperti yang ditunjukkan dalam log di bawah ini. Ada header (abc) dalam permintaan, tetapi tidak ada dalam permintaan yang diteruskan ke layanan backend. Perhatikan bahwa saat Anda menambahkan parameter dalam permintaan metode, parameter tersebut secara otomatis dipetakan dalam permintaan integrasi, jadi Anda harus menghapus pemetaan secara eksplisit.

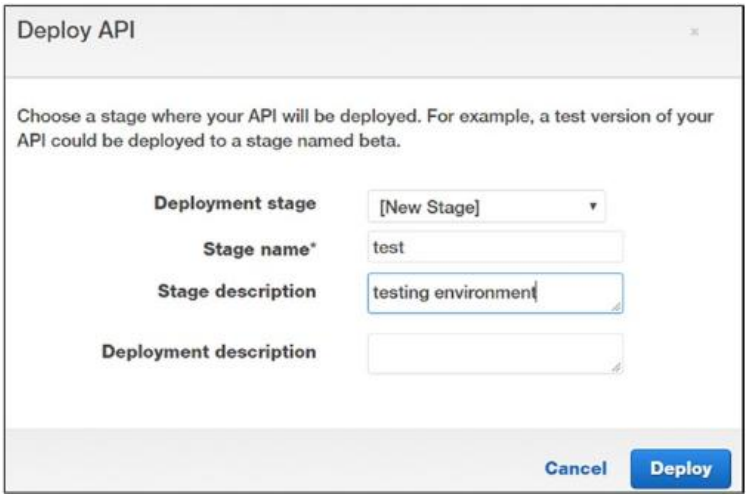
```
Method request headers: {abc=zzz, version=v1}

Endpoint request headers: {api-version=v1, x-amzn-apigateway-api-
id=pciz9lt9p5, Accept=application/json, User-
Agent=AmazonAPIGateway_pciz9lt9p5, X-Amzn-Trace-Id=Root=1-5a800263-
ed2e227b694e156930d6176e}
```

Anda juga dapat menentukan model untuk payload permintaan/respons di bagian Model API dan menggunakannya dalam pemetaan. Selain itu, respons Gateway digunakan untuk mengonfigurasi kode status HTTP, pesan respons, dan nilai header gateway API.

4.6 DEPLOY AWS API GATEWAY

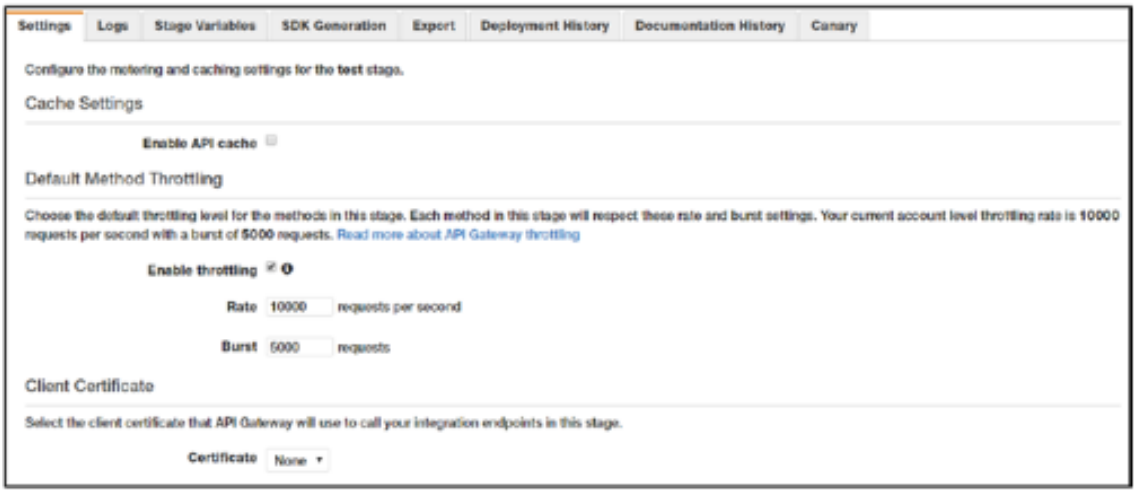
Kita harus menerapkan API untuk mengalokasikan rencana penggunaan dan menentukan kunci API. Di panel yang ditunjukkan pada Gambar 4.27, pilih tindakan Deploy API dari menu Actions. Ini akan meminta Anda untuk membuat tahapan (seperti yang

ditunjukkan pada Gambar 4.30). Tahapan adalah lingkungan yang berbeda dari API yang diterapkan. Semua metode harus memiliki integrasi; setelah kondisi ini terpenuhi, Anda dapat menerapkan API.



Gambar 4.30 Membuat Tahap dan Menyebarkan API

Setelah tahap dibuat, kita dapat mengonfigurasi lebih banyak hal di tingkat tahap. Selain itu, karena kita dapat membuat sejumlah tahap, kita dapat menyebarkan API yang sama di lingkungan yang berbeda. Gambar 4.31 menunjukkan opsi tahap. Di panel ini, bagian Pengaturan mencakup opsi untuk tingkat permintaan, penyimpanan sementara, dan sertifikat klien. Anda juga dapat memberi tag tahap dengan pasangan nilai kunci yang diperlukan.



Gambar 4.31 Pengaturan Tahap API

- *Pengaturan:* Di bagian Pengaturan, kita dapat mengonfigurasi caching. AWS API Gateway memungkinkan kita memilih ukuran cache, dan harganya pun disesuaikan. Di tingkat metode, kita dapat mengonfigurasi apakah metode tertentu harus di-cache atau tidak. Kita juga dapat mengonfigurasi pembatasan metode di bagian ini; pembatasan digunakan untuk mengamankan batas kecepatan API. Terakhir, kita dapat

mengatur sertifikat klien untuk autentikasi. Bab 5 menjelaskan batas kecepatan dan autentikasi secara lebih terperinci.

- *Log:* Di sini, kita dapat mengonfigurasi log untuk API. Di dunia AWS, ini biasanya adalah layanan CloudWatch.
- *Variabel tahap:* Kita dapat mengatur variabel sebagai pasangan nilai kunci, yang dapat diakses dalam pemetaan.
- *Pembuatan SDK:* AWS menyediakan dukungan untuk membuat SDK klien untuk API. Ini termasuk bahasa pemrograman seluler populer Android dan SWIFT, dan bahasa seperti Java dan Ruby.
- *Ekspor:* Di sini, kita dapat mengekspor definisi API dalam berbagai format.
- *Riwayat penyebaran:* Bagian ini berisi riwayat penyebaran tahapan.
- *Riwayat dokumentasi:* Bagian ini menampilkan riwayat dokumentasi tahapan yang terlampir.

Dokumen dapat dibuat, diedit, dan dipublikasikan di bagian Dokumentasi API.

- Canary: Bagian ini memungkinkan kita untuk menetapkan canary untuk tahapan. Rilis canary memastikan bagian tertentu dari lalu lintas masuk ke canary yang disebutkan, sementara sisanya masuk ke versi lama. Pertimbangkan skenario di mana versi baru dari backend tersedia; ini dapat dicapai dengan nilai header `api-version = v2`. Namun, kita tidak ingin mengarahkan semua lalu lintas ke v2, jadi kita akan menetapkan rilis canary pada tahapan dan memilih persentase lalu lintas yang harus masuk ke versi baru. Versi baru diidentifikasi oleh variabel, dan kita dapat menggantinya dalam pengaturan canary dengan mengirimkan variabel canary baru.

Membuat Rencana Penggunaan API

Selanjutnya, kita akan membuat rencana penggunaan API yang akan digunakan. Rencana penggunaan membantu mengukur penggunaan API dan mengaturnya melalui pembatasan. Kita dapat menetapkan tahapan API yang berbeda ke rencana penggunaan tunggal, dan rencana tersebut dapat berisi tahapan dari API yang berbeda. Klik opsi Rencana Penggunaan di menu utama di sisi kiri. Buat rencana penggunaan seperti yang ditunjukkan pada Gambar 4.32.

Gambar 4.32 Membuat Paket Penggunaan API

Setelah Anda memberi nama paket, Anda dapat mengaktifkan pembatasan dan menentukan permintaan kuota untuk paket tersebut. Pada langkah berikutnya, kita akan memilih API dan tahap API yang akan disertakan dalam paket. Untuk melacak pembatasan dan batas kuota, kita harus menetapkan kunci API ke paket penggunaan. AWS API Gateway menggunakan langganan untuk mengukur permintaan. Kita dapat mengimpor kunci yang sudah ada atau meminta AWS API Gateway untuk membuat kunci secara otomatis. Klik Buat Kunci API dan tambahkan ke Paket Penggunaan (lihat Gambar 4.33). Dengan asumsi Anda tidak memiliki kunci yang tersedia, di sini Anda dapat memilih untuk membuat kunci secara otomatis. Kunci baru akan ditetapkan ke paket penggunaan.

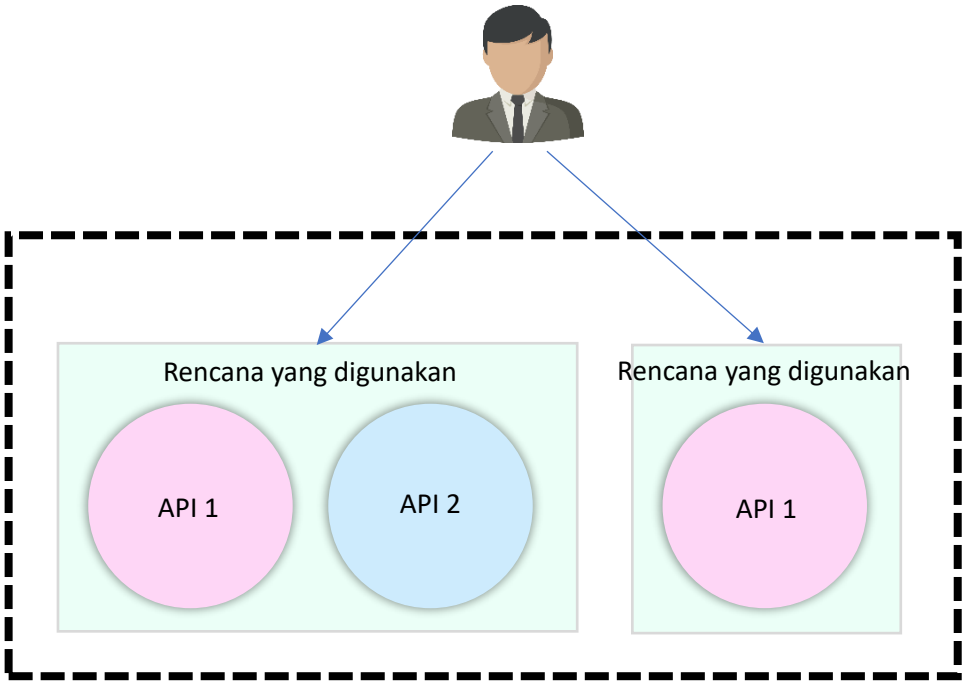
Gambar 4.33 Membuat Kunci API untuk Paket Penggunaan

Kunci yang baru dibuat dapat ditemukan di bagian Kunci API. Terakhir, Anda akan membuat paket penggunaan. Setelah dibuat, Anda dapat mengeklik paket dan mengedit propertinya. Saat memilih paket, Anda akan melihat tab bernama Marketplace, yang merupakan saluran pengiriman pengembang untuk API. Kita dapat mengaitkan paket penggunaan dengan produk AWS Marketplace menggunakan kode produk.

Struktur Komponen AWS API Gateway

AWS API Gateway dapat memiliki beberapa API, dan setiap API memiliki serangkaian

sumber daya. Setiap sumber daya dapat memiliki lebih banyak sumber daya atau metode di dalamnya, dan setiap metode memiliki integrasi terkait dengan layanan backend atau tiruan. API diterapkan secara bertahap. Satu API dapat memiliki banyak tahap, dan setiap tahap dapat memiliki pengaturan dan konfigurasinya sendiri. Paket penggunaan dapat memiliki banyak API. Paket penggunaan memiliki kuota dan pengaturan pembatasan, yang dilacak oleh kunci langganan. Pengembang mengakses paket penggunaan dari AWS Marketplace; ini adalah mekanisme monetisasi API dalam konteks AWS.



Gambar 4.34 Struktur AWS API Gateway

Ringkasan

API gateway bertindak sebagai middleware untuk layanan backend. Mereka menangani orkestrasi permintaan dan respons, kontrol aliran, dan keamanan. API gateway khusus vendor sering digabungkan sebagai layanan manajemen API, yang mencakup elemen lain dari arsitektur API dan elemen penyampaian nilai seperti pengalaman pengembang, fitur kinerja, dan fitur pemantauan. Dalam konteks arsitektur berbasis layanan modern, API gateway memainkan peran utama dalam mengatasi pemisahan dan manajemen.

Layanan API gateway berbasis internet seperti Azure API Management dan AWS API Gateway bertindak sebagai proksi yang menghadap publik untuk layanan backend. Bab ini memberikan informasi tentang gateway API, berbagai skenario pemetaan titik akhir, dan implementasi praktis gateway API dari penawaran layanan Azure dan AWS. Kami juga memperkenalkan dasar-dasar penggunaan gateway API dalam pola tiruan dan pola arsitektur lainnya.

BAB 5
KEAMANAN API

API mengekspos data dan operasi bisnis, jadi harus aman. Sebagai pengembang API, Anda harus melindungi API dari konsumen yang tidak sah, mengendalikan laju konsumsi, dan mengatur data. Desain API yang strategis membantu mencapai perlindungan dan tata kelola data sensitif bisnis. Skala cloud dan fleksibilitas yang ditawarkannya menghasilkan banyak tantangan berbeda dalam mengamankan API. Pengembang menggabungkan berbagai implementasi keamanan di berbagai tingkat sistem dalam implementasi API.

Bab ini membahas langkah-langkah keamanan umum yang diterapkan di lapisan API Azure dan AWS. Rinciannya dibahas dalam dua topik utama: implementasi keamanan berbasis permintaan dan autentikasi. Implementasi keamanan berbasis permintaan mendikte kebijakan dan batasan pada konsumsi API terutama, siapa yang dapat mengonsumsi dan berapa banyak sementara autentikasi mendikte kebijakan dan batasan pada autentikasi dan otorisasi, terutama dalam hal siapa konsumen dan apa yang dapat diakses konsumen.

5.1 KEAMANAN BERBASIS PERMINTAAN

Implementasi keamanan berbasis permintaan mengidentifikasi konsumen dan menerapkan batasan pada konsumsi mereka, baik dengan membatasi laju konsumsi atau mengizinkan atau memblokir konsumen. Dalam implementasi ini, identifikasi konsumen bersifat sederhana dan tidak mencakup mekanisme autentikasi yang rumit. Secara umum, kunci API digunakan untuk mengidentifikasi konsumen.

Dalam ekonomi API, khususnya dalam model penjualan langsung, tingkat konsumsi merupakan parameter mendasar dalam mendefinisikan SKU API yang berbeda. Dalam kasus yang jarang terjadi, kita dapat mengamati batasan pada fitur API sebagai model penjualan. Layanan manajemen API berbasis cloud seperti Azure API Management dan AWS API Gateway memiliki pengaturan ini, dan kita dapat mengonfigurasinya untuk menerapkan aturan keamanan berbasis permintaan.

Azure API Management

Azure API Management memiliki pengaturan keamanan berbasis permintaan yang dapat dikonfigurasi, yang aturannya diterapkan menggunakan kebijakan Azure API Management. Jika Anda baru mengenal kebijakan Azure API Management, saya sarankan untuk membaca bab 4 sebelum membaca bagian ini.

Langganan dan Kunci Langganan

Di Azure API Management, untuk membuat permintaan yang berhasil ke API yang ditawarkan sebagai produk Azure API Management (untuk informasi lebih lanjut tentang produk, lihat bab 4), konsumen harus memiliki kunci langganan yang valid yang terkait dengan produk tersebut. Pengembang API dapat meminta atau secara otomatis mengambil (tergantung pada pengaturan) kunci langganan untuk produk di portal pengembang. Satu langganan untuk satu produk menyediakan dua kunci langganan utama dan sekunder.

Memiliki dua kunci langganan membantu mengelola waktu henti selama pergantian kunci. Kunci langganan dikirim ke gateway Azure API Management di header permintaan ocp-apim-subscription-key. Penggunaan dilacak di tingkat langganan, apa pun kunci yang digunakan. Dalam kebanyakan kasus, kunci langganan digunakan untuk melacak penggunaan, tetapi Azure API Management memungkinkan pengembang untuk mengonfigurasi batas penggunaan berdasarkan parameter lain, seperti alamat IP atau kode respons.

Batasan Kecepatan Permintaan

Buka layanan Azure API Management dan pilih Katalog MassRover produk Azure API Management yang kami buat di bab 4. Kemudian, buka bagian kebijakan produk dan tambahkan kebijakan pembatasan kecepatan. Kebijakan tingkat produk diterapkan ke semua titik akhir secara agregat. Jika kecepatannya adalah 10 permintaan per menit untuk suatu produk dan produk tersebut memiliki dua titik akhir API, konsumen dapat membuat maksimal 10 permintaan per menit sebagai kecepatan permintaan gabungan, tetapi tidak diperbolehkan membuat 10 permintaan ke setiap titik akhir secara terpisah.

Kami dapat menerapkan kebijakan tingkat titik akhir untuk menegakkan aturan tersebut. Secara umum, kebijakan berbasis permintaan diterapkan di tingkat produk. Pernyataan kebijakan berikut di bagian masuk mengizinkan 2000 panggilan dalam 60 detik untuk titik akhir API yang disertakan dalam produk tertentu.

```
<rate-limit calls="2000" updates-period="60" />
```

Jika pemanggil melampaui batas ini, Azure API Management akan merespons dengan kode HTTP 429 (terlalu banyak permintaan). Isi pesan akan berisi pesan dan periode waktu percobaan ulang, dan pemanggil harus menunggu selama periode waktu tersebut agar permintaan dapat diproses. Isi pesan yang berisi waktu yang tersisa membantu menerapkan logika percobaan ulang yang cerdas daripada mengakses gateway secara acak. Dengan kebijakan di atas, batas kecepatan diterapkan berdasarkan kunci langganan; secara teknis, kunci langganan berfungsi sebagai kunci penghitung kenaikan untuk permintaan.

Kita dapat mengontrol kecepatan permintaan menggunakan nilai kunci yang berubah-ubah juga. Kunci dapat berupa variabel apa pun yang dapat diakses oleh Azure API Management, dan nilai kunci dapat berupa nilai apa pun yang dapat diakses dalam konteks permintaan. Contoh di bawah ini menunjukkan batas kecepatan permintaan berdasarkan alamat IP.

```
<rate-limit-by-key calls="1000"
     renewal-period="90"
     increment-condition="@(context.Response.
     StatusCode == 200)"
     counter-key="@(context.Request.IpAddress)"
/>
```

Kebijakan di atas mengizinkan 1000 permintaan, dan mengembalikan 200 dalam 90 detik untuk setiap alamat IP. Jadi, penelepon dengan alamat IP tidak dapat membuat lebih dari 1000

panggilan dalam 90 detik, yang mengembalikan 200. Contoh lain adalah menolak panggilan yang menyebabkan server mengembalikan 500. Pernyataan kebijakan di bawah ini menerapkan pembatasan berdasarkan nilai header; jika permintaan dengan nilai header tertentu menyebabkan server gagal 10 kali dalam 60 detik, maka konsumen akan diblokir hingga periode pembaruan diperbarui.

```
<rate-limit-by-key
   calls="10"
   renewal-period="60"
   increment-condition="@(context.Response.StatusCode == 500)"
   counter-key="@(context.Request.Headers.massrover_token)"/>
```

Batasan Kuota

Batasan kuota diterapkan terutama untuk mengontrol kuota permintaan. Dalam skenario praktis, ini digunakan untuk monetisasi API, di mana SKU yang berbeda dapat memiliki kuota yang berbeda. Untuk menerapkan kuota, kita dapat menggunakan kebijakan kuota.

```
<quota calls="100000" bandwidth="80000" renewal-period="3600" />
```

Dalam kebijakan di atas, kita dapat menerapkan kuota dengan periode pembaruan satu jam untuk sejumlah panggilan atau lebar pita tertentu dalam kilobyte. Ambang batas mana pun yang terpenuhi terlebih dahulu akan memicu kondisi tersebut. Kebijakan di atas melacak penggunaan berdasarkan kunci langganan. Seperti halnya kebijakan pembatasan tarif, kita dapat menerapkan batas kuota berdasarkan kunci acak.

```
<quota-by-key calls="10000" bandwidth="3000" renewalperiod="3600"
   counter-key="@(context.Request.Headers.massrover_token)"
   increment-condition="@(context.Response.StatusCode >= 200 &&
   context.Response.StatusCode < 400)”
   increment-count="@(context.Request.Method == "POST" ? 1:0)" />
```

Penghitung kebijakan kuota ini memiliki kenaikan untuk permintaan dengan nilai header massrover_token. Kenaikan permintaan dipicu untuk permintaan POST yang memiliki respons HTTP 200 atau lebih tinggi dan di bawah 400. Kebijakan akan memblokir lebih banyak saat jumlah permintaan memenuhi kondisi di atas lebih dari r 10000 atau lebar pita permintaan melebihi 3000 KB dalam satu jam mana pun kondisi yang terpenuhi terlebih dahulu.

Pembatasan IP

Azure API Management memiliki kebijakan pembatasan berbasis IP yang memungkinkan kami mengizinkan atau memblokir IP tertentu.

```
<ip-filter action="allow">
     <address-range from="10.1.1.2" to="10.1.1.16" />
</ip-filter>
```

5.2 AWS API GATEWAY

AWS API Gateway memiliki pengaturan keamanan berbasis permintaan yang dapat dikonfigurasi, yang diterapkan melalui Paket Penggunaan API. Bab 4 menjelaskan dasar-dasar AWS API Gateway dan Paket Penggunaan, jadi jika Anda baru mengenal AWS API Gateway, saya sarankan untuk membaca bab 4 sebelum membaca bagian ini.

Kunci API

Kita dapat membuat kunci untuk API di bawah "Kunci API." Kunci ini dibuat secara otomatis oleh layanan AWS API Gateway atau disediakan secara khusus.

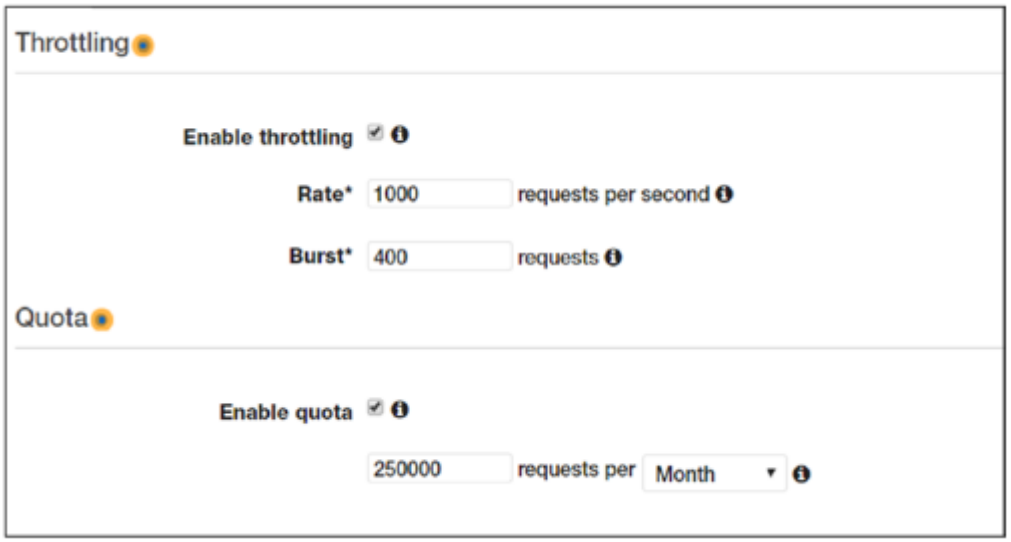
Gambar 5.1 Pembuatan Kunci API AWS API Gateway

Gambar 5.1 menunjukkan layar tempat kita dapat membuat kunci API. Kunci API dikaitkan dengan paket penggunaan API dan digunakan untuk melacak permintaan. Kita dapat menyesuaikan cara konsumen mengirim kunci API ke AWS API Gateway. Kunci API dikirim baik dalam header HTTP XAPIKEY_HEADER atau melalui otorisasi kustom yang dikonfigurasi. Konfigurasi ini dapat dilakukan di bagian Pengaturan pada API yang dipilih (Gambar 5.2).

Gambar 5.2 Pengaturan untuk Sumber Permintaan Kunci API

Batasan Kecepatan

Untuk mengonfigurasikan batasan kecepatan, kita harus mengaitkan kunci API dengan paket penggunaan. Kita menetapkan batasan kecepatan dalam paket penggunaan, dan permintaan dilacak oleh kunci API. Satu paket penggunaan dapat mencakup banyak API, dan semua API dalam paket penggunaan dibatasi oleh batasan kecepatan. Untuk menetapkan batasan kecepatan, navigasikan ke paket penggunaan yang dipilih. Anda akan melihat dua pengaturan berbeda yang tersedia, Pembatasan dan Kuota. Lihat Gambar 5.3.



Gambar 5.3 Pengaturan Paket Penggunaan AWS API Gateway

Seperti yang dapat Anda lihat, pembatasan telah diaktifkan, kecepatan ditetapkan ke 1000 permintaan per detik, dan burst ditetapkan ke 400 permintaan. Ini berarti bahwa paket penggunaan khusus ini dapat menangani 1000 permintaan per detik tanpa pembatasan apa pun, ketika 1000 permintaan tersebut didistribusikan secara merata pada satu permintaan per milidetik. Burst adalah jumlah maksimum permintaan yang dapat ditangani paket penggunaan saat permintaan tersebut tiba secara bersamaan. Contoh berikut menjelaskan dua skenario berbeda untuk pengaturan burst.

- Jika konsumen membuat 500 panggilan dalam 500 milidetik, dengan cara yang didistribusikan secara merata, maka API akan menanggapi permintaan tersebut tanpa membatasi satu pun dari permintaan tersebut. Dalam milidetik ke-501, jika penelepon membuat 405 permintaan, maka permintaan tersebut akan dibatasi, karena kecepatan burst telah terlampaui (meskipun batas kecepatan belum terlampaui).
- Jika dalam milidetik pertama konsumen membuat 400 permintaan dan kemudian penelepon mencoba membuat 1 permintaan per milidetik selama 999 milidetik yang tersisa, permintaan akan dibatasi mulai dari milidetik ke-602, karena pada saat itulah kecepatan panggilan akan melampaui batas 1000 permintaan per detik.

Batas Kuota

Paket penggunaan juga memiliki pengaturan batas kuota, yang tersedia tepat di bawah bagian pembatasan (lihat Gambar 5-3). Kita dapat mengatur kuota berdasarkan jumlah permintaan per hari, minggu, atau bulan.

5.3 AUTENTIKASI & OTORISASI

Secara sederhana, autentikasi mengidentifikasi penelepon dan otorisasi memberikan informasi tentang apakah penelepon memiliki akses ke sumber daya yang diamankan atau tidak. API dapat menerapkan autentikasi dan otorisasi dengan berbagai cara; secara umum,

API mengharapkan nilai dalam permintaan yang mencakup informasi keamanan yang diperlukan. Nilai ini dapat berisi informasi keamanan di dalamnya sendiri, atau dalam format dan skema yang dapat dipahami API, atau dapat menjadi referensi ke informasi keamanan nyata yang dapat diambil oleh API.

Ada banyak standar dan protokol yang tersedia dalam keamanan API, dan semuanya bermuara pada kriteria yang disebutkan di atas. Mengeluarkan nilai tertentu; mengamankannya dari akses dan gangguan yang tidak sah; mendaur ulang nilai; memvalidasi nilai; dan tugas manajemen lainnya yang terkait dengan pembuatan, validasi, pemeliharaan, dan penghancuran nilai dapat dilakukan dalam berbagai bentuk, tergantung pada teknologi, spesifikasi vendor, standar terbuka, dan kepatuhan.

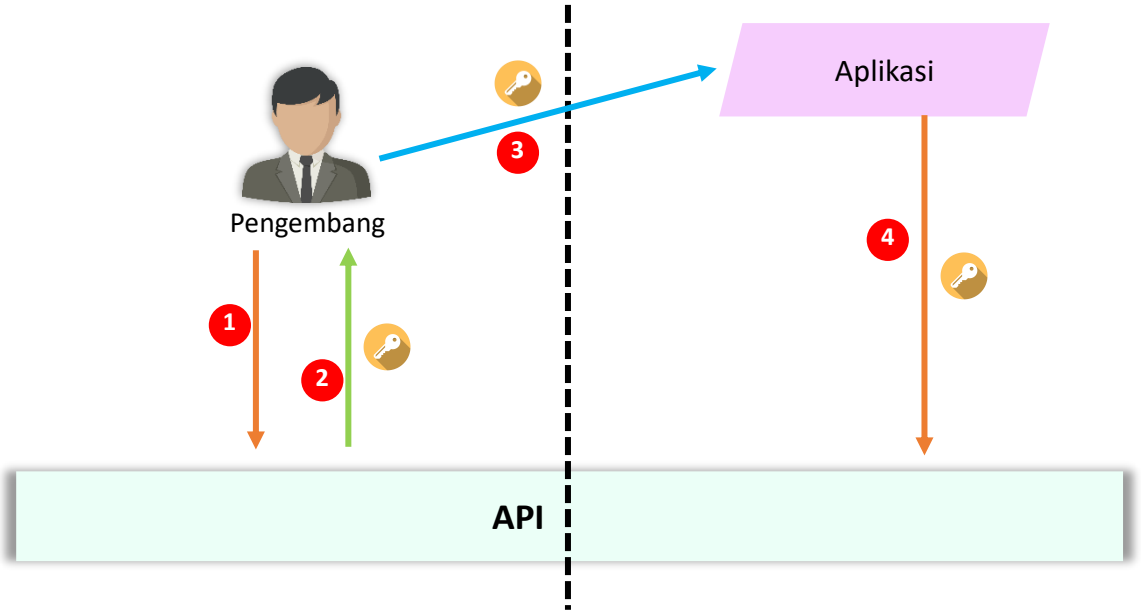
Ada banyak sekali standar dan protokol seperti itu di pasaran. Beberapa sangat umum dan banyak digunakan, dan beberapa sangat spesifik untuk vendor atau teknologi tertentu. Rincian standar dan protokol tersebut berada di luar cakupan buku ini, tetapi bagian ini membahas skenario autentikasi dan otorisasi API berbasis cloud yang paling umum, terutama yang menargetkan Azure dan AWS.

Desain Keamanan API

Tidak ada satu ukuran yang cocok untuk semua, jadi tidak ada satu desain keamanan yang sesuai untuk semua kasus bisnis, tetapi seperti yang dinyatakan sebelumnya, ada baiknya untuk mempertimbangkan beberapa pola implementasi utama. API tidak memiliki status, artinya tidak menyimpan informasi status di antara permintaan, jadi setiap permintaan harus berisi semua elemen yang diperlukan untuk pemenuhan. Informasi keamanan juga harus menjadi bagian dari setiap permintaan, dengan asumsi API memerlukan informasi ini untuk memenuhi permintaan. Informasi tersebut dapat berada di badan permintaan, string kueri, atau tajuk permintaan.

Kunci API

Kunci API adalah pengenal sederhana dari pemanggil. Sebagai pengembang, Anda mendaftarkan diri Anda dengan penyedia API atau membuktikan identitas Anda kepada penyedia API untuk mendapatkan kunci untuk mengakses API. Misalnya, dalam API Google Maps, pengembang membuktikan identitas mereka melalui login Google sebelum menerima kunci. Kunci ini adalah string acak yang dikeluarkan oleh penyedia API setiap permintaan harus berisi kunci. Penyedia API mengidentifikasi pemanggil dan batasan akses. Umumnya, implementasi ini digunakan jika tidak ada persyaratan khusus untuk mengidentifikasi pengguna, tetapi penggunaannya perlu dipantau. Model API penjualan langsung menggunakan implementasi kunci API.

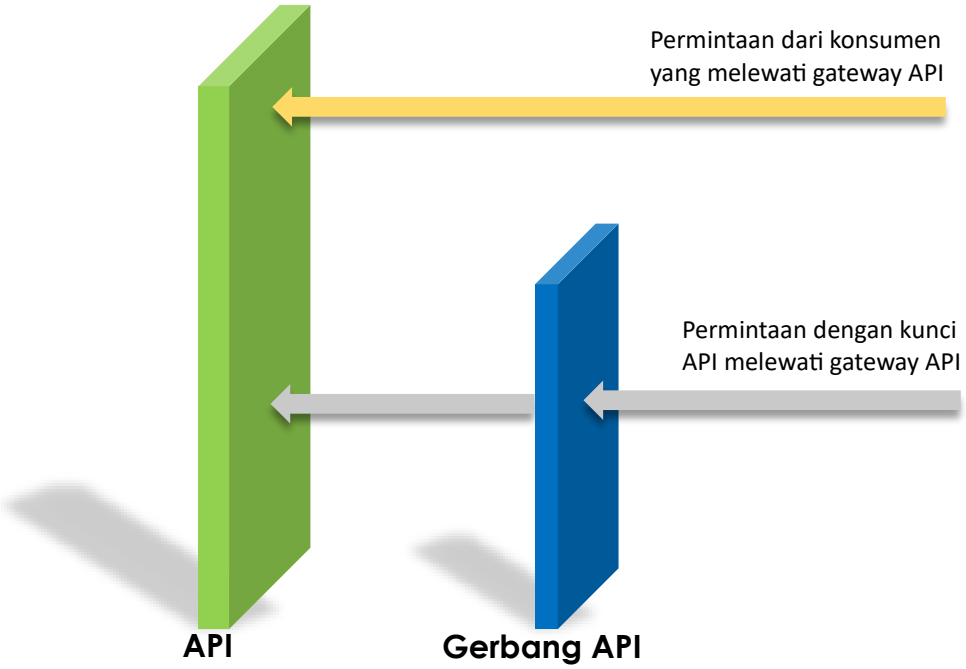


Gambar 5.4 Keamanan Berbasis Kunci API

API publik sering mengikuti pola ini; manajemen kunci penuh dikontrol oleh API atau ditangani di layanan gateway. Ingat kunci langganan Azure API Management dan kunci API AWS API Gateway. Penerbitan dan pengelolaan kunci ditangani di layanan gateway, dan layanan backend tidak mengetahui kunci tersebut. Dalam model API penjualan langsung, sebagian besar kunci tidak kedaluwarsa, tetapi dapat dibatalkan oleh API karena terlalu banyak permintaan, dugaan serangan berbasis permintaan, atau pembayaran yang terlewat. Namun, konsumen API tidak diperbolehkan mendaur ulang kunci mereka karena alasan keamanan yang jelas (lihat Gambar 5.4).

1. Pengembang meminta kunci API dari penyedia API/API. Permintaan ini mungkin memiliki prasyarat, seperti validasi atau pembayaran tertentu.
2. Ketika prasyarat terpenuhi, pengembang menerima kunci dari penyedia API/API.
3. Pengembang menyimpan kunci ini di aplikasi.
4. Aplikasi mengirimkan kunci ke API dalam permintaan.

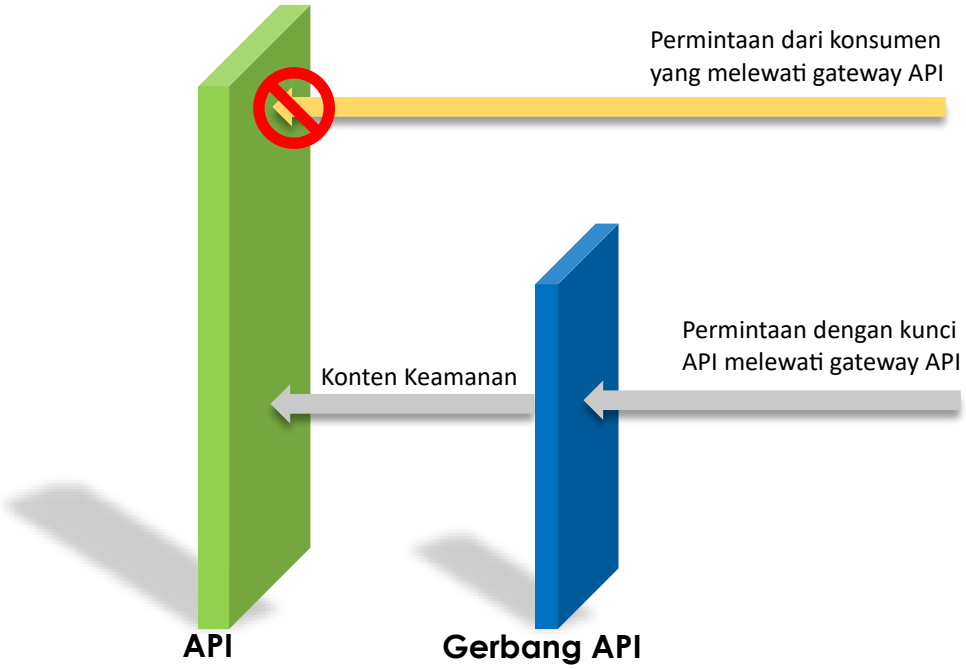
Pengembang dapat menggunakan kunci yang sama di berbagai aplikasi; API tidak mengetahuinya. Satu-satunya informasi yang diperhatikan API adalah apakah kunci tersebut valid dalam hal kondisi dan batasan penggunaan. Seperti yang dibahas, di Azure dan AWS, layanan gateway menangani manajemen kunci. Ini memindahkan logika manajemen kunci dari implementasi API, tetapi jika API itu sendiri terbuka dan memungkinkan akses langsung, maka pemanggil dapat berhasil membuat permintaan langsung ke API. Gambar 5-5 menggambarkan implementasi kunci API di gateway API.



Gambar 5.5 Kunci API Tingkat Gateway API

Seperti yang Anda lihat, gateway API memiliki konteks kunci yang lengkap, dan layanan backend tidak mengetahui implementasi kunci. Implementasi kunci dapat dibawa ke layanan backend dan digunakan dalam logika bisnis, tetapi desain semacam ini memperkenalkan banyak logika manajemen kunci yang harus ditulis dalam API itu sendiri, sehingga mencegah manfaat manajemen kunci gateway API. Di sisi lain, jika API backend terbuka dan konsumen mengetahui titik akhir layanan backend, struktur, dan detail lainnya maka permintaan dapat dibuat langsung ke layanan backend dengan melewati gateway API. Ini adalah kelemahan umum dalam implementasi API berbasis cloud.

Menyebarkan API di layanan Internet terbuka seperti AWS Beanstalk atau Azure API Apps dan menganggap bahwa layanan gateway API (Azure API Management atau AWS API Gateway) akan menjaga keamanan hanya karena kunci API dikonfigurasi merupakan celah keamanan yang besar. Implementasi yang sebenarnya harus memiliki keamanan yang diterapkan di layanan backend, dan akses publik ke layanan backend harus dikunci. Mencegah akses publik ke layanan backend dapat dicapai dengan konfigurasi infrastruktur dan implementasi perangkat lunak. Masalah di atas dapat dengan mudah diperbaiki dengan memperkenalkan beberapa konteks antara layanan backend dan gateway API. Gambar 5-6 menggambarkan hal ini.



Gambar 5.6 Kepercayaan Antara Gateway API dan Layanan Backend

Seperti yang ditunjukkan dalam bab sebelumnya, gateway API dapat memanipulasi permintaan dengan menyuntikkan konteks keamanan yang disepakati ke dalam permintaan yang mereka buat ke layanan backend. Layanan backend mencari konteks keamanan yang tepat dalam setiap permintaan yang diterimanya, dan menolak permintaan yang tidak memenuhi harapan ini. Konteks keamanan ini dapat mengambil bentuk yang berbeda; berikut ini adalah yang paling umum:

- Kunci bersama (string statis) antara gateway API dan layanan backend. Layanan backend mengharapkan nilai ini sebagai pengenal kunci simetris dari layanan gateway API.
- Autentikasi sertifikat antara gateway API dan layanan backend, implementasi asimetris karena layanan backend mempercayai autentikasi sertifikat tertentu.
- Layanan backend itu sendiri, atau penyedia autentikasi tepercaya lainnya, mengeluarkan token untuk komunikasi antara gateway API dan layanan backend.
- Campuran dari hal di atas, dengan implementasi keamanan berbasis jaringan lainnya, seperti firewall dan pengaturan IP, yang hanya mengizinkan panggilan dari gateway API.

Kunci API di gateway tidak menawarkan keamanan yang komprehensif, jadi menggabungkannya dengan salah satu metode di atas mengamankan layanan backend sekaligus menawarkan fleksibilitas manajemen kunci API dari layanan gateway API yang tersedia.

5.4 OPENID DAN OAUTH

Jika Anda seorang pengembang web atau bekerja di bidang pengembangan API, kemungkinan besar Anda belum pernah mendengar istilah OpenID dan OAuth. Meskipun sangat umum, terdapat banyak kebingungan di antara pengembang dan pengguna antara

kedua standar ini. Bagian ini menjelaskan perbedaan di antara keduanya, karena ini merupakan pengetahuan yang diperlukan untuk memahami bagian selanjutnya. Ya, sesederhana itu, tetapi sulit untuk membedakannya dalam implementasi di dunia nyata karena otorisasi mencakup autentikasi, yang berarti bahwa OAuth bergantung pada OpenID. Namun, OAuth sebagai standar tidak menentukan autentikasi tertentu yang harus dilakukan melalui OpenID agar berfungsi. Dalam hal implementasi teknis, ada beberapa kesamaan. Misalnya, OpenID dan OAuth bergantung pada pengalihan peramban.

OpenID mengidentifikasi pengguna. Misalnya, bayangkan Anda menggunakan login Facebook untuk mengakses situs web (sebut saja example.com). Saat Anda menavigasi ke example.com dan melihat opsi untuk "Login dengan Facebook", Anda mengklik tombol tersebut untuk menghindari pengisian formulir pendaftaran yang panjang. Saat Anda melakukan ini, jika Anda sudah login ke Facebook, Anda tidak perlu memasukkan kredensial Facebook Anda lagi. Ini adalah OpenID; ini memfasilitasi autentikasi pengguna dari penyedia yang sama ke beberapa entitas. Prinsip dasar single sign-on didasarkan pada OpenID.

Terkadang Facebook akan merespons dengan perintah, yang mengatakan bahwa example. com mencoba mengakses email, foto, atau informasi lain milik pengguna. Ini adalah OAuth. Dalam kasus ini, example.com perlu mengakses beberapa informasi dari Facebook, dan meminta izin dengan meminta izin kepada Anda, karena Anda mengizinkan example.com untuk membaca data Anda yang tersimpan di Facebook. Dengan memberikan izin, Anda mengizinkan example.com untuk membaca data Anda.

Secara teknis, Facebook memberikan beberapa akses kepada example.com untuk membuat permintaan ke sumber dayanya. Setelah Anda memberikan izin, komunikasi selanjutnya dapat dilakukan antara example.com dan Facebook berdasarkan izin yang diizinkan. Ini juga OAuth. Di bagian selanjutnya, kita akan melihat bagaimana OpenID dan OAuth digunakan dalam mengamankan API.

Mengamankan API dengan Azure Active Directory V2

Jika Anda berkecimpung dalam pengembangan aplikasi Azure, mengamankan layanan dengan Azure Active Directory (AAD) merupakan persyaratan umum. AAD memiliki versi baru API autentikasi, yang dikenal sebagai v2, yang memiliki beberapa perubahan signifikan dibandingkan dengan versi sebelumnya. Di bagian ini, kita akan melihat cara mengamankan API menggunakan AAD v2. Anda dapat membaca selengkapnya tentang perbedaan antara v1 dan v2 di sini: https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-v2-compare.

AAD menawarkan manajemen identitas berbasis cloud yang komprehensif, yang melayani berbagai alur dan protokol autentikasi yang kompleks. Hal ini berada di luar cakupan buku ini. Untuk wawasan yang lebih mendalam tentang AAD. Untuk mengamankan API menggunakan penyedia identitas apa pun, klien harus terlebih dahulu memperoleh identitas yang valid dari penyedia identitas. Selanjutnya, identitas yang diperoleh dikirim ke API dalam permintaan, dan API memvalidasi identitas dan melayani permintaan tersebut. Untuk memvalidasi identitas, API harus mengetahui penyedia identitas dan mekanisme yang digunakan untuk memvalidasi identitas.

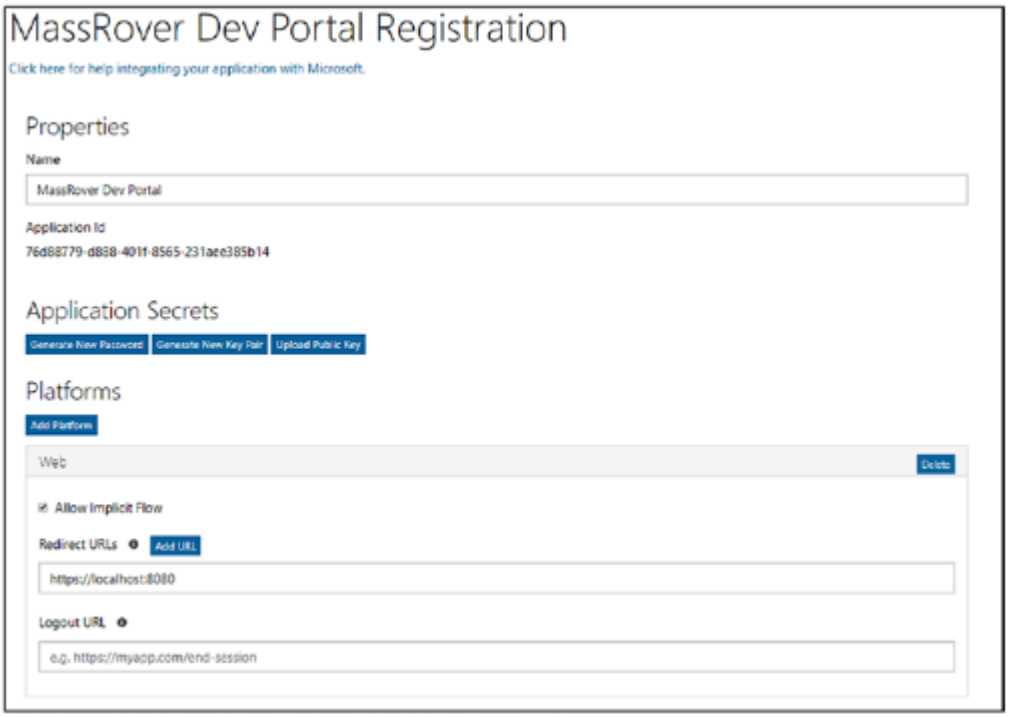
Sekarang, mari kita lihat pengaturan untuk mengamankan API menggunakan AAD. Ada dua langkah utama yang terlibat:

1. Memfasilitasi klien untuk mendapatkan token (identitas)
2. Memfasilitasi API untuk memvalidasi token

Pertama, buat pengaturan agar klien dapat meminta token dari AAD v2. Ini dilakukan melalui titik akhir otorisasi AAD v2 OAuth. Klien harus membuat permintaan ke titik akhir otorisasi ini, mengirimkan kredensial mereka, dan mendapatkan OpenID. Berikut adalah contoh permintaan dari klien.

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=[
clientid]&response_type=id_token&redirect_uri=[redirect
```

Seperti yang dapat Anda lihat, untuk membuat permintaan ini, klien memerlukan, minimal, ID klien terdaftar dan URI pengalihan yang dikonfigurasi, tempat AAD akan mengalihkan jenis respons yang diminta. Kita perlu mendaftarkan aplikasi klien di AAD dan mengatur informasi tersebut. Kunjungi https://apps.dev.microsoft.com/ (yang merupakan titik akhir untuk mendaftarkan aplikasi AAD v2) dan navigasikan ke Portal Pendaftaran Aplikasi yang baru. Masuk dengan kredensial AAD atau akun Microsoft Anda dan buat aplikasi AAD v2 baru, seperti yang ditunjukkan pada Gambar 5.7.



Gambar 5.7 Pendaftaran Aplikasi AAD V2

Tentukan platform aplikasi sebagai web dan tetapkan URL pengalihan (dalam kasus ini, localhost ditetapkan, jadi kita dapat mengujinya tanpa harus menggunakan aplikasi yang sebenarnya). Klien harus menentukan ID klien dan URL balasan yang dikonfigurasi dalam

permintaan, seperti yang ditunjukkan di bawah ini.

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=7
6d88779-d888-401f-8565-231aee385b14&response_type=id_token&redirect_uri=
https://localhost:8080&scope=openid &nonce=3c9d2ab9-2d3b-4
```

ID klien dan URI pengalihan harus sesuai dengan nilai yang dikonfigurasi dalam aplikasi AAD v2. Karena persyaratannya adalah memperoleh identitas, OpenID ditetapkan dalam cakupan. Anda dapat mengujinya menggunakan URL di atas dan mengautentikasi dengan akun organisasi atau Microsoft Anda dan mengambil id_token (jenis respons). Token akan dikirimkan ke URL pengalihan (https://localhost:8080).

Aplikasi di atas terdaftar di bawah akun AAD penulis dan mengalihkan token ke localhost, sehingga Anda dapat mengujinya dengan aman hanya dengan menempelkannya di browser Anda. URL permintaan di atas menerima autentikasi AAD yang valid karena URL otorisasi mengarah ke titik akhir yang sama. Ini berlaku untuk akun Microsoft dan akun organisasi. Jika Anda hanya ingin menerima autentikasi organisasi, ganti "umum" dengan "organisasi" seperti di bawah ini.

```
https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize?rest
-of-the-url
```

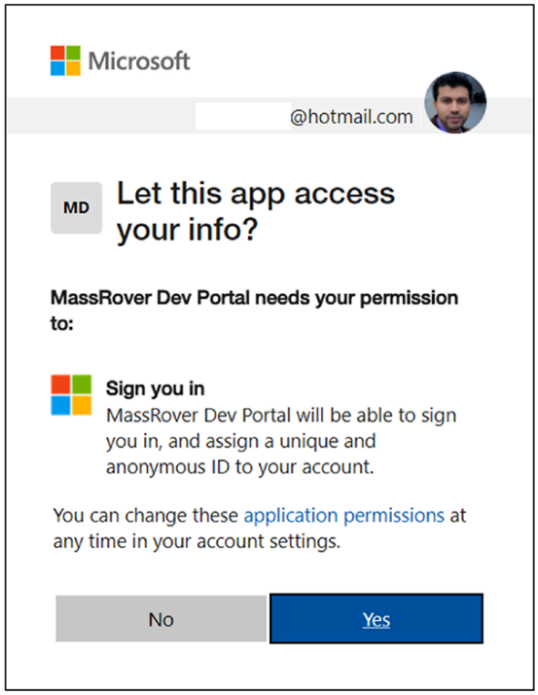
Jika Anda merancang aplikasi penyewa tunggal, yang mengharapkan autentikasi dari satu penyewa AAD, Anda dapat menentukan ID penyewa di URL permintaan.

```
https://login.microsoftonline.com/[tenantid]/oauth2/v2.0/authorize?rest-
of-the-url
```

Konsumen dapat membuat salah satu permintaan di atas, mengautentikasi diri mereka sendiri dengan kredensial AAD, dan memperoleh informasi OpenID. Dalam konteks yang dijelaskan di atas, informasi OpenID ada di id_token yang dikeluarkan oleh titik akhir AAD v2 ke URL pengalihan yang diinstruksikan. Saat Anda menguji URL di bawah ini:

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?client_id=7
6d88779-d888-401f-8565-231aee385b14&response_type=id_token&redirect_uri
=https://localhost:8080&scope=openid &nonce=3c9d2ab9-2d3b-4
```

Saat Anda masuk untuk pertama kalinya, Anda akan melihat layar persetujuan, seperti yang ditunjukkan pada Gambar 5-8. Ini adalah permintaan dari AAD v2 untuk mendapatkan persetujuan dari pengguna dan menerbitkan informasi OpenID kepada klien yang meminta. Upaya masuk berikutnya tidak akan meminta persetujuan ini.



Gambar 5.8 Layar Persetujuan AAD V2

Setelah berhasil masuk dan memperoleh persetujuan, AAD v2 akan dialihkan ke URL yang ditentukan dengan id_token, seperti di bawah ini.

```
https://localhost:8080/#id_token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUz
I1NiIsImtpZCI6IjFMVE16YWtpaGlSbGFfOHoyQkVKVlhlV01xbyJ9.eyJ2ZXI
iOiIyLjAiLCJpc3MiOiJodHRwczovL2xvZ2luLm1pY3Jvc29mdG9ubGluZS5jb
20vOTE4ODA0MGQtNmM2Ny00YzViLWIxMTItMzZhMzA0YjY2ZGFkL3YyLjAiLCJz
dWIiOiJBQUFBQUFBQUFBQUFBQUFBQUFBQUFKNFk4NE56dWdlYl8yTFBWcFlkbzN
jIiwiYXVkIjoiNzZkODg3NzktZDg4OC00MDFmLTg1NjUtMjMxYWVlMzg1YjE0Ii
wiZXhwIjoxNTIxNDY5NTE5LCJpYXQiOjE1MjEzODI4MTksIm5iZiI6MTUyMTM4M
jgxOSwidGlkIjoiOTE4ODA0MGQtNmM2Ny00YzViLWIxMTItMzZhMzA0YjY2ZGFk
Iiwibm9uY2UiOiIzYzlkMmFiOS0yZDNiLTQiLCJhaW8iOiJEVEghayFwMzdNamh
jcE8qWkF6aTBKb3Z0b2x4RDhVZDk5R3Ria2RaSFAqbFRVU01wbUp1Q2h1IVVtbV
hzbEoqYkFkTWU3bEtTY2JYbjlIWTFNc0tUNUxBMEM5anJBdjBaTE1XV3oyZXVnS
mZSbGdRMGNZZWlnMHd5S0piakVldE13JCQifQ.HZbpQmkdi-2yOHtwtF-zFJhz7
RJe3_GIkcmS5u5EwIV7U_5x8S_2_o6JfQ0KBpnhzop5UijP99Rjan0dTtfat2Bs
TnWZloLbKy9X30XwJzrd-8WU2Nz7zwg24rMKEu0t6c8-uR2ze-U1dhogGQZj6eu
rvnsedL4ET9eYehvPxV18U3AsSkZ2LAERpEiZeu16G0ORWpwGBI5NvogYhkRxzY
iZGlC5MsmkvZa4VdyVce_zJ-AnGrgwBvi5oL083RFGNMKUDevHDGpefO-UW3XOq
D3WiJmvdYx3g4ZPyamYH7UgyR0DIgOXnuLIoPXEop8AgrbJkPjJRbIkRVDMqjFAww
```

Anda dapat menggunakan alat dekode JWT apa pun yang tersedia untuk melihat detailnya. `http://calebb.net/` adalah alat daring yang bagus yang digunakan untuk melihat bagian dalam JWT, dan JWT di atas didekodekan sebagai berikut:

```
{
  typ: "JWT",
  alg: "RS256",
```

```
    kid: "1LTMzakihiRla_8z2BEJVXeWMqo"
  }.
  {
    ver: "2.0",
    iss: "https://login.microsoftonline.com/9188040d-6c67-4c5bb112-
    36a304b66dad/v2.0",
    sub: "AAAAAAAAAAAAAAAAAAAAAJ4Y84Nzugeb_2LPVpYdo3c",
    aud: "76d88779-d888-401f-8565-231aee385b14",
    exp: 1521469519,
    iat: 1521382819,
    nbf: 1521382819,
    tid: "9188040d-6c67-4c5b-b112-36a304b66dad",
    nonce: "3c9d2ab9-2d3b-4",
    aio: "DTH!k!p37MjhcpO*ZAzi0JovtolxD8Ud99GtbkdZHP*lTUSMpm
    JuChu!UmmXslJ*bAdMe7lKScbXn9HY1MsKT5LA0C9jrAv0ZLMWWz2eugJfRlgQ0
    cYeig0wyKJbjEetMw$$"
  }.
```

Setelah memperoleh token, konsumen akan mengirimkannya ke API, yang seharusnya dapat memvalidasi token dan mengambil informasi yang akan digunakan dalam logika bisnis. Dalam aplikasi ASP.NET Core, Anda dapat menggunakan kode berikut untuk melakukan validasi token dan meminta klaim dari token.

```
  private async Task<System.IdentityModel.Tokens.Jwt.JwtSecurityToken>
    ValidateAADIdTokenAsync(string idToken)
  {
    var stsDiscoveryEndpoint =
    "https://login.microsoftonline.com/common/v2.0/.well-known/openid-
    configuration";

    var configRetriever = new
    Microsoft.IdentityModel.Protocols.OpenIdConnect.OpenIdConnectConfigurati
    onRetriever();

    var configManager = new
    Microsoft.IdentityModel.Protocols.ConfigurationManager<OpenIdConnect
    Configuration>
    (stsDiscoveryEndpoint, configRetriever);

    var config = await configManager.GetConfigurationAsync();
    var tokenValidationParameters = new
    Microsoft.IdentityModel.Tokens.TokenValidationParameters
    {
      IssuerSigningKeys = config.SigningKeys,
    };
    var tokenHandler = new JwtSecurityTokenHandler();
      tokenHandler.ValidateToken(idToken,tokenValidationParameters, out var
      validatedToken);
    return validatedToken as JwtSecurityToken;
  }
```

Metode ini menerima token sebagai string dan memperoleh kunci penandatanganan token dari penyedia (AAD v2) melalui titik akhir layanan token aman https://login.microsoftonline.com/common/v2.0/.well-known/openid-configuration. URL ini akan digunakan oleh `OpenIdConfiguration` untuk memperoleh informasi penandatanganan. Anda dapat mengubah URL ini dengan mengganti "common" dengan "organizations" atau "tenant id" sesuai pola URL permintaan yang dibahas sebelumnya. Kunci penandatanganan yang diperoleh dan pengaturan validasi lainnya digunakan untuk membuat `TokenValidationParameters`. Akhirnya token divalidasi, dan output akan dikonversi ke `JwtSecurityToken`, yang berisi klaim dalam token untuk akses terprogram.

Cuplikan kode di atas "`TokenValidationParameters`" memiliki konfigurasi validasi minimum, yang berarti memeriksa apakah token dikeluarkan oleh entitas tepercaya yang benar dengan memvalidasi kunci penandatanganan. Dalam kasus implementasi di dunia nyata, aturan validasi yang lebih kompleks akan digunakan untuk melakukan validasi, bersama dengan aturan penandatanganan seperti memvalidasi penerbit, audiens, dan kedaluwarsa. Token dan autentikasi berbasis klaim berjalan beriringan dalam implementasi. Ada banyak jenis token, masing-masing dengan cakupan berbeda dan berisi berbagai informasi berdasarkan permintaan autentikasi dan alur autentikasi.

Namun secara umum, token adalah string bertanda tangan yang berisi potongan informasi (klaim). JWT adalah format token standar, yang banyak digunakan oleh banyak penyedia. Dalam implementasi praktis, cuplikan kode validasi token di atas akan menjadi filter di ASP.NET Core. Setiap header permintaan dari klien akan berisi id_token dan divalidasi melalui filter. Klaim tertentu akan diambil dari id_token untuk menjalankan logika bisnis. Penyedia identitas publik seperti Google dan Facebook dapat digunakan untuk mengautentikasi pengguna.

Sebagian besar klaim dalam informasi OpenID tersebut tidak dapat dikaitkan dengan logika bisnis khusus dari aplikasi lini bisnis perusahaan yang umum, karena logika bisnis sebagian besar menangani peran dan izin khusus aplikasi, yang berada di luar konteks penyedia identitas yang disebutkan. Untuk mengatasi hal ini, kita dapat menerbitkan token khusus dari aplikasi setelah memvalidasi identitas dari penyedia eksternal. Dalam mode ini, kita bergantung pada penyedia identitas eksternal untuk autentikasi dan menerbitkan token khusus untuk otorisasi. Bagian berikutnya menjelaskan proses penerbitan token khusus secara lebih rinci.

5.5 MENERBITKAN TOKEN JWT KHUSUS

Seperti yang saya jelaskan, ada beberapa kasus di mana kita perlu membuat dan menerbitkan token JWT kita sendiri. Misalnya, katakanlah Anda sedang mengembangkan aplikasi SaaS yang mengandalkan beberapa penyedia identitas. Penyedia identitas ini membantu pengguna mengautentikasi ke aplikasi dengan lebih sedikit gesekan, dan OpenID memainkan peran penting dalam membangun pengalaman masuk tunggal, tetapi setelah pengguna diautentikasi, aplikasi harus mengetahui informasi otorisasi pengguna, termasuk peran dan izin.

Secara sederhana, Google atau Facebook tidak dapat menyimpan detail pengguna, meskipun pengguna adalah admin aplikasi kustom Anda. Merupakan tanggung jawab layanan/aplikasi backend untuk mengelola otorisasi. Di bagian sebelumnya, saya menjelaskan cara mendapatkan token JWT dengan klaim OpenID dari AAD dan memvalidasinya. Ini akan membantu mengamankan API menggunakan AAD dari sudut pandang autentikasi, tetapi setelah pengguna masuk, API harus menentukan otorisasi pengguna untuk memutuskan apa yang dapat dilakukan pengguna di dalam aplikasi. Cuplikan kode di bawah menunjukkan cara menerbitkan token JWT kustom menggunakan kunci penandatanganan simetris.

```
private string IssueJwtToken(JwtSecurityToken aadToken)
{
     var msKey = GetTokenSignKey();
     var msSigningCredentials = new
     Microsoft.IdentityModel.Tokens.SigningCredentials
       (msKey, SecurityAlgorithms.HmacSha256Signature);

     var claimsIdentity = new ClaimsIdentity(new List<Claim>()
     {
       new Claim(ClaimTypes.NameIdentifier,"thuru@massrover.com"),
       new Claim(ClaimTypes.Role, "admin"),
     }, "MassRover.Authentication");

     var msSecurityTokenDescriptor = new
     Microsoft.IdentityModel.Tokens.SecurityTokenDescriptor()
     {
       Audience = "massrover.client",
       Issuer = "massrover.authservice",
       Subject = claimsIdentity,
       Expires = DateTime.UtcNow.AddHours(8),
       SigningCredentials = msSigningCredentials
     };

   var tokenHandler = new JwtSecurityTokenHandler();
   var plainToken = tokenHandler.CreateToken(msSecurityTokenDescriptor);
   var signedAndEncodedToken = tokenHandler.WriteToken(plainToken);

   return signedAndEncodedToken;
}
```

Kode penerbitan JWT di atas memperoleh kunci penandatanganan dari metode pribadi ini.

```
private Microsoft.IdentityModel.Tokens.SymmetricSecurityKeyGetTokenSignKey()
{
   var plainTextSecurityKey = "massrover secret key";

   var msKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey
   (Encoding.UTF32.GetBytes(plainTextSecurityKey));

   return msKey;
```

```
}
```

Kode penerbitan token menggunakan `ClaimsIdentity` untuk menyertakan klaim khusus aplikasi dalam subjek token. Untuk langkah berikutnya, Microsoft. `IdentityModel.Tokens.SecurityTokenDescriptor` digunakan untuk membuat token JWT lengkap beserta klaim khusus dalam subjek. Objek ini digunakan untuk membuat token JWT menggunakan metode `CreateToken` dari `JwtSecurityTokenHandler`, dan token dikembalikan sebagai string. Saat kami menerbitkan token khusus, API juga harus dapat memvalidasi token, saat dikembalikan dari pemanggil dalam permintaan. Cuplikan kode di bawah ini menunjukkan kode validasi token.

```
public bool ValidateMassRoverToken(string token)
{

   var tokenValidationParameters = new
   Microsoft.IdentityModel.Tokens.TokenValidationParameters()
   {

      ValidAudiences = new string[]
      {
         "massrover.client",
      },

      ValidIssuers = new string[]
      {
         "massrover.authservice",
      },

      ValidateLifetime = true,
      IssuerSigningKey = GetSignTokenSignKey()
   };

   var tokenHandler = new JwtSecurityTokenHandler();
      tokenHandler.ValidateToken(token,tokenValidationParameters,  out  var
      validatedToken);

   return true;
}
```

Kode tersebut menggunakan objek `TokenValidationParameters`, yang mencakup logika validasi token beserta kunci penandatanganan (diperoleh dari metode privat yang sama yang digunakan untuk menerbitkan token). Metode `ValidateToken` dari `JwtSecurityTokenHandler` memvalidasi token menggunakan objek `TokenValidationParameters` yang telah dibangun. Perhatikan bahwa cuplikan kode di atas belum siap untuk produksi, dan alur token dalam aplikasi produksi memerlukan implementasi perangkat lunak beserta dukungan TLS. Potongan kode di atas menjelaskan dasar-dasar penerbitan dan validasi token JWT di ASP.NET Core.

Ada sejumlah pustaka dan kerangka kerja yang tersedia yang menyediakan implementasi manajemen token yang aman dan komprehensif dalam aplikasi bisnis. Pustaka ini mencakup banyak skenario autentikasi yang berbeda. Dalam implementasi di dunia nyata, Anda akan menggunakan salah satu pustaka atau kerangka kerja tersebut untuk menangani manajemen token autentikasi, daripada menulis sendiri seluruh kode. Identity Server adalah kerangka kerja autentikasi yang populer di dunia .NET. Setelah alur token JWT berjalan, konsumen akan mengirim token dalam setiap permintaan ke API.

Dalam skenario umum, token dikirim dalam header Otorisasi. Layanan backend menerima token dari permintaan HTTP dan memvalidasi serta memperoleh informasi darinya untuk mengoordinasikan persyaratan logika bisnis. Dalam beberapa kasus, token tidak berisi informasi apa pun selain pengenal, tetapi layanan backend tahu untuk mendapatkan informasi yang diperlukan menggunakan pengenal ini. Jenis token ini dikenal sebagai token referensi. Di bagian berikutnya, kita akan membahas cara menggunakan Azure API Management untuk melakukan pra-autentikasi permintaan yang berisi token JWT.

Pra-Autentikasi dalam Azure API Management

Dalam banyak contoh di atas, kita melihat bahwa Azure API Management dapat memproses informasi permintaan dan respons. Pra-autentikasi di gateway merupakan praktik yang baik dan mencegah masuknya layanan backend untuk semua permintaan. Namun, sangat disarankan untuk memvalidasi token di layanan backend dan memperoleh informasi yang tersimpan di sana. Kita akan menggunakan kebijakan Validasi JWT untuk langkah pra-autentikasi ini. Cuplikan di bawah ini menunjukkan implementasi kebijakan validasi JWT dasar.

```
<validate-jwt
   header-name="Authorization" failed-validation-httpcode="401 "
   failed-validation-error-message="Unauthorized"
   require-expiration-time="true"
   require-scheme="scheme"
   require-signed-tokens="true">
<audiences>
   <audience>76d88779-d888-401f-8565-231aee385b14</audience>
</audiences>
<required-claims>
   <claim name="massrover-role" match="any">
      <value>admin</value>
      <value>user</value>
   </claim>
</required-claims>
   <openid-config url=" https://login.microsoftonline.com/common/.well-
   known/openid-configuration" />
</validate-jwt>
```

Dalam cuplikan di atas, saya telah menyediakan URL openid-config dan perbandingan validasi klaim dasar. Kode tersebut juga memeriksa klaim kustom yang disebut massrover-role dan memastikan keberadaannya dalam token JWT dan nilai yang dapat diambil oleh admin atau pengguna. Perlu dicatat juga bahwa require-expiration-time disetel ke true. Untuk melewati

validasi ini, token JWT yang masuk harus berisi klaim exp. Jika klaim exp tidak ada, validasi akan gagal. Kebijakan validasi JWT Azure API Management mendukung algoritme penandatanganan HS256 dan RS256. Untuk HS256, keduanya harus disediakan dalam kebijakan itu sendiri sebagai string berkode base64, seperti di bawah ini.

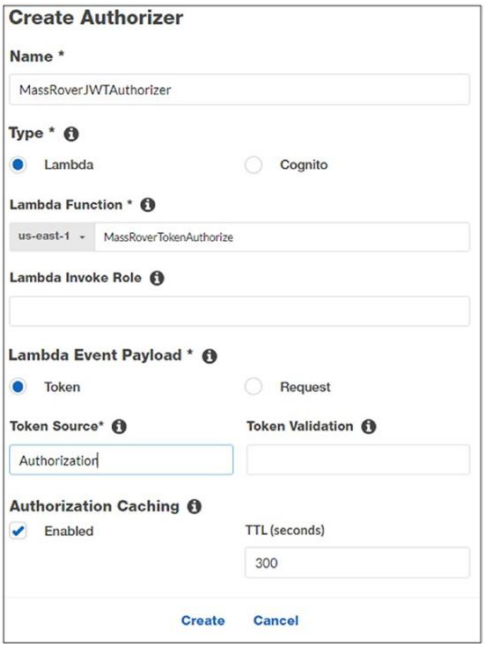
```
<issuer-signing-keys>
  <key>base64 encoded key</key>
</issuer-signing-keys>
```

Untuk RS256, kunci harus disediakan melalui titik akhir konfigurasi OpenID, seperti yang ditunjukkan dalam contoh kebijakan di atas. Anda dapat menambahkan lebih banyak aturan validasi ke kebijakan ini, termasuk memvalidasi klaim kustom, seperti klaim massrover-role. Ada kekhawatiran tentang menghadirkan lebih banyak konteks bisnis ke gateway API, yang disebut sebagai gateway API yang terlalu ambisius. Baca selengkapnya tentang ini di sini: https://www.thoughtworks.com/radar/platforms/overambitious-api-gateways.

Dalam kasus di atas, memvalidasi klaim kustom (massrover-role) dan mengendalikan akses dapat dianggap sebagai praktik yang buruk, karena ini membawa logika bisnis ke gateway. Pada saat yang sama, sebagai alat, API Gateway menawarkan banyak fitur seperti ini. Anda dapat membaca lebih lanjut tentang keamanan yang sangat dapat disesuaikan di API Gateway dari posting ini: https://thuru.net/2018/02/28/overambitious-api-gateways-security-at-api-gateway-with-azure-api-management/

5.6 PEMBERI WEWENANG DI AWS API GATEWAY

Pemberi wewenang disiapkan di AWS API Gateway untuk mengautentikasi permintaan yang masuk. AWS API Gateway menangani ini dengan fungsi Lambda (lebih lanjut tentang serverless di bab 6) atau AWS Cognito. Lambda adalah platform serverless AWS, dan Cognito adalah layanan kontrol akses berbasis AWS.



Gambar 5.9 Buat AWS API Gateway Authorizer

Untuk menyiapkan pemberi wewenang, pertama-tama kita harus membuat pemberi wewenang di bawah API yang dipilih. Navigasi ke API, pilih Pemberi wewenang, dan klik Buat Pemberi wewenang Baru. Anda akan melihat panel seperti yang ditunjukkan pada Gambar 5-9. Di panel, berikan nama untuk pemberi wewenang dan pilih jenis sebagai Lambda, yang dapat kita tulis kode untuk melakukan logika validasi autentikasi.

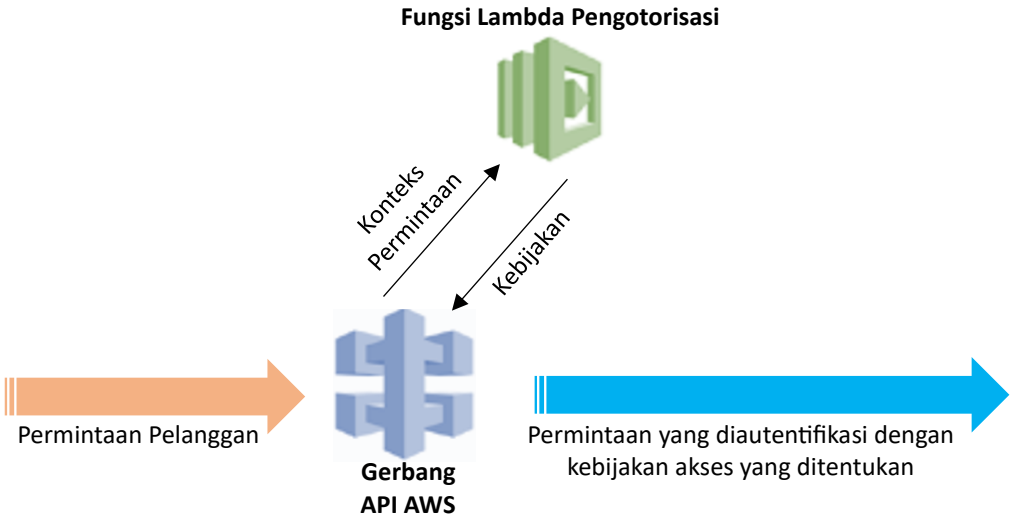
Sebagai Azure API Management, ini dilakukan melalui kebijakan di AWS API Gateway, ini dilakukan melalui fungsi serverless eksternal dengan kode kustom. Selanjutnya, pilih fungsi Lambda, yang memiliki logika pemberi wewenang. Untuk melakukannya, Anda harus memiliki fungsi dan implementasi Lambda yang ada, atau memberikan nama (dimasukkan dalam kotak teks) fungsi tersebut, yang akan digunakan nanti. Lihat bab 6 untuk petunjuk tentang cara membuat dan menggunakan fungsi Lambda.

Pilih Token untuk Lambda Event Payload. Ini memastikan token akan ada di header yang ditentukan. Pilih Request jika token ada di badan `request/header/string` kueri event payload dengan nilai yang ditentukan. Tentukan nilai terkait yang memiliki token di Token Source. Seperti di atas, konteks Lambda akan mencari token di header Authorization. Aturan untuk mencari token di header berasal dari jenis Lambda Event Payload, dan nilainya berasal dari Token Source. Hal yang sama akan berlaku jika Anda memilih Request sebagai Lambda Event Payload; pencarian akan terjadi di berbagai tempat, termasuk header, parameter string kueri, variabel tahap, dan parameter konteks untuk kunci Token Source yang ditentukan.

Jika Anda mau, Anda dapat menentukan validasi regex di bagian Token Validation, yang mencegah masuknya Lambda untuk token yang tidak dalam format yang tepat. Caching otorisasi menunjukkan apakah akan menyimpan dokumen kebijakan otorisasi dalam cache atau tidak. Dokumen kebijakan otorisasi dibuat oleh pemberi otorisasi untuk token yang ditentukan. Pemberi otorisasi AWS API Gateway mengembalikan kebijakan otorisasi ke AWS API Gateway dengan aturan akses dan penolakan yang ditentukan.

Anda dapat membaca lebih lanjut tentang kebijakan ini di sini: https://docs.aws.amazon.com/AmazonS3/latest/dev/access-policy-language-overview.html.

Kebijakan ini menentukan apakah AWS API Gateway memiliki akses ke sumber daya AWS tertentu. Pembuatan dokumen kebijakan merupakan logika dalam fungsi Lambda pemberi otorisasi. Seperti disebutkan di atas, dokumen kebijakan ini mengontrol akses ke sumber daya AWS, dan dalam hal logika bisnis aplikasi, Lambda pemberi otorisasi harus memiliki logika validasi token (mirip dengan logika validasi JWT yang ditunjukkan di bagian Menerbitkan Token JWT Kustom di atas). Setelah validasi, Lambda akan menyusun dokumen kebijakan dan mengembalikannya ke AWS API Gateway, yang akan menyimpan dokumen dalam cache selama periode yang ditentukan jika penyimpanan dalam cache diaktifkan. Gambar 5-10 mengilustrasikan alur ini.



Gambar 5.10 Alur Autentikasi Pemberi Otorisasi AWS API Gateway

Setelah membuat pemberi otorisasi Lambda, kami akan menerapkan logika yang memvalidasi pembuatan token dan dokumen kebijakan. Kode berikut menunjukkan cara melakukannya:

```
public class Function
{
   public Policy FunctionHandler(APIGatewayCustomAuthorizer
   Request authRequest,ILambdaContext context)
   {
      var token = authRequest.AuthorizationToken;
      Policy policy;

      if (ValidateToken(token))
         {
         var statement = new Statement(Statement.StatementEffect.Allow);
         var policyStatements = new List<Statement> {
            statement };

         policy = new Policy("TokenValidationPassed",policyStatements);
         }
      else
         {
         var statement = new Statement(Statement.StatementEffect.Deny);
         var policyStatements = new List<Statement> {
         statement };

         policy = new Policy("TokenValidationFailed",
         policyStatements);
         }
      return policy;
      }
   private bool ValidateToken(string token)
   {
```

```
    // JWT token validation here
    return true;
    }
}
```

Di sini, dokumen kebijakan hanya menunjukkan apakah akses diizinkan atau tidak, dan tidak berisi kebijakan akses khusus apa pun ke sumber daya AWS tertentu. Kita dapat membuat implementasi kebijakan khusus, tetapi implementasi ini cukup memadai untuk mengalirkan permintaan ke layanan backend.

Ringkasan

Keamanan API adalah bidang yang luas, dan merupakan salah satu topik keamanan yang semakin mendapat perhatian dan permintaan. Bab ini membahas beberapa area minat umum dalam pengembangan aplikasi modern. Perhatikan bahwa bab ini tidak menyediakan panduan komprehensif untuk keamanan API maupun implementasi alur penuhnya di platform yang dipilih. Bab ini menyediakan detail dasar dan informasi implementasi yang cukup memadai untuk memulai dan menemukan lebih banyak, berdasarkan persyaratan bisnis yang ada.

Seperti yang dinyatakan, keamanan API melibatkan berbagai lapisan implementasi, dan bab ini membahasnya dalam dua bagian utama: keamanan berbasis permintaan dan keamanan berbasis autentikasi dan otorisasi. Keamanan berbasis permintaan mencakup batasan panggilan, batasan kecepatan, pembatasan IP, dan implementasi kunci API. Keamanan berbasis autentikasi dan otorisasi mencakup implementasi kunci API, implementasi OAuth dari penyedia yang tersedia di Azure dan AWS, dan implementasi manajemen JWT khusus tanpa pustaka pihak ketiga.

BAB 6
API SERVERLESS

Serverless adalah model komputasi yang bersifat sementara; titik akhir serverless dipicu berdasarkan satu peristiwa. Peristiwa ini memunculkan komputasi dan menjalankan logika yang ditentukan. Baik pemanggil maupun pengembang tidak menyadari infrastruktur komputasi yang muncul untuk melayani peristiwa tersebut. Infrastruktur tersebut benar-benar tersembunyi dan membuat keberadaan infrastruktur komputasi tidak terlalu terlihat.

Hal yang sama berlaku untuk pengembangan dan penerapan, karena tidak ada definisi server yang eksplisit dan paket kode infrastruktur diberikan ke platform serverless dan dijalankan. Karakteristik ini, dan sifat komputasi yang berumur pendek, telah menghasilkan istilah "serverless" dalam industri. Dalam bab ini, kami akan fokus pada teknologi dan ekonomi serverless, platform serverless yang tersedia di Azure dan AWS, dan cara menggunakannya dalam mengembangkan API.

6.1 KOMPUTASI SERVERLESS

Serverless adalah topik yang sangat populer di kalangan pengembang. Penawaran serverless hadir dalam dua jenis utama: Functions as a Service (FaaS) dan Backend as a Service (BaaS). Model FaaS memungkinkan pengembang untuk menulis logika kustom, membuat paket kode, dan menjalankannya dalam konteks tanpa server. Fungsi-fungsi ini memerlukan pemicu peristiwa untuk dapat dijalankan. Platform tanpa server utama mendukung berbagai bahasa dan runtime dalam model FaaS. Azure Functions dan AWS Lambda masing-masing merupakan penawaran FaaS dari Azure dan AWS.

Model BaaS memungkinkan pengembang untuk dengan mudah membuat integrasi dan alur kerja yang memicu penggunaan dengan menghubungkan layanan yang tersedia dari penyedia yang berbeda. Layanan seperti autentikasi, pencarian, transformasi data, penyimpanan sementara, integrasi, pembayaran, akuntansi, dan basis data yang tersedia sebagai layanan memungkinkan pengembang untuk menggunakannya kembali, sehingga mengurangi biaya desain dan pengembangan aplikasi. Integrasi antara berbagai layanan dan koordinasi alur logika memungkinkan kita untuk membangun aplikasi yang kuat dengan cepat.

Aplikasi Logika dan Layanan Alur Kerja Sederhana (SWF) masing-masing merupakan penawaran BaaS dari Azure dan AWS, meskipun SWF sebagian besar digunakan dalam skenario orkestrasi sumber daya internal. AWS mulai menawarkan model tanpa server dengan Lambda, dan segera penyedia cloud lainnya ikut terjun ke pasar. Sekarang, hampir semua vendor cloud publik terkemuka menyediakan platform tanpa server. Selain penawaran cloud, ada banyak platform tanpa server di tempat yang tersedia. Janji ekonomi tanpa server dicontohkan oleh dua manfaat utama. Salah satunya adalah waktu pengembangan.

Model BaaS telah merangkul layanan pihak ketiga dan mengurangi waktu pengembangan secara dramatis, yang memungkinkan pengembang untuk fokus pada penyampaian nilai. Selain itu, serverless menyembunyikan sebagian besar penerapan dan

pengelolaan aplikasi. Kedua, biaya menjalankan aplikasi dalam serverless jauh lebih murah, dan meskipun ini sebagian bergantung pada sifat aplikasi dan cara penggunaannya, penawaran awal serverless dari vendor cloud utama tampak sangat menjanjikan dalam hal biaya. Namun, serverless mungkin tidak selalu murah dibandingkan dengan penawaran PaaS dan IaaS.

Dari sudut pandang teknis, serverless menawarkan fleksibilitas untuk mengembangkan dan mengirimkan dengan cepat, karena menghilangkan overhead penyediaan sumber daya dan merangkul layanan pihak ketiga sebanyak mungkin. Serverless adalah kandidat yang baik untuk dipasang ke sistem yang ada karena bekerja berdasarkan peristiwa. Suatu peristiwa dapat dimulai oleh sistem yang ada, dengan lebih sedikit kabel dan logika serverless dapat mengambilnya dan menjalankan logika tersebut. Struktur ini telah dianut oleh pengembang, dan serverless digunakan dengan berbagai sistem lama untuk mengembangkan fitur baru.

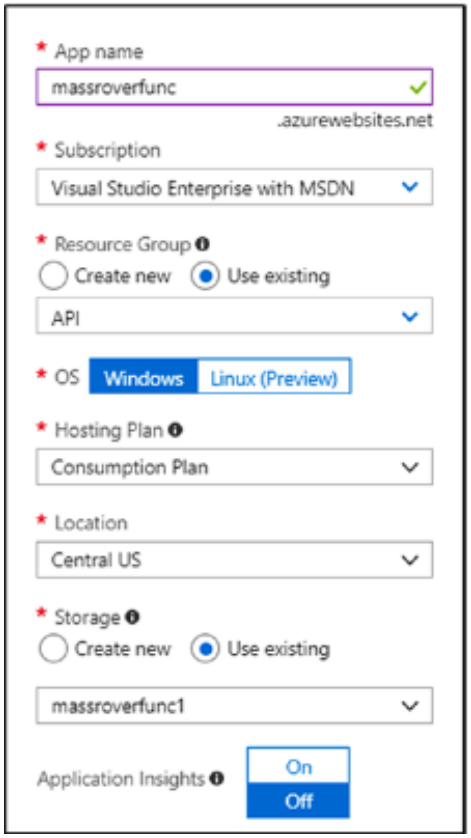
API tanpa server adalah fungsi API yang didukung oleh platform tanpa server. Sebagian besar platform tanpa server dapat menerima peristiwa HTTP sebagai pemicu, sehingga siap digunakan sebagai API (Azure Functions adalah contoh yang baik untuk ini). Dalam beberapa kasus, teknologi API lain digunakan untuk mengintegrasikan eksekusi tanpa server dengan jalur HTTP. Misalnya, AWS API Gateway digunakan dengan integrasi proksi dengan Lambda untuk melayani backend API menggunakan model tanpa server.

6.2 API TANPA SERVER DI AZURE

Azure memiliki banyak penawaran tanpa server, seperti Azure Functions, Logic Apps, Azure Storage, Cloud Messaging (Event Grid), Azure Bot Services, dua penawaran tanpa server utama, Azure Functions, dan Logic Apps, tetapi sesuai tren dan permintaan pasar saat ini, kami akan menjelajahi Azure Functions dan Logic Apps. Azure Functions adalah penawaran FaaS—pengembang dapat menulis logika kustom berdasarkan pemicu yang tersedia dan menjalankan logika tersebut. Logic Apps adalah penawaran BaaS tempat berbagai tindakan dan kondisi alur kerja tersedia untuk menjalankan logika berdasarkan pemicu peristiwa. Baik Azure Functions maupun Logic Apps mendukung peristiwa HTTP, yang berarti keduanya dapat bertindak sebagai API mandiri.

Azure Functions

Azure Functions adalah penawaran di bawah Azure App Service. Mari kita lihat cara memulai dengan Azure Functions dan membuat pemicu HTTP. Kita dapat membuat Azure Functions di portal menggunakan Function Apps. Membuat Azure Function App Navigasi ke portal Azure, cari Function Apps, dan buat layanan Function App. Gambar 6-1 memperlihatkan bilah pembuatan Function App.



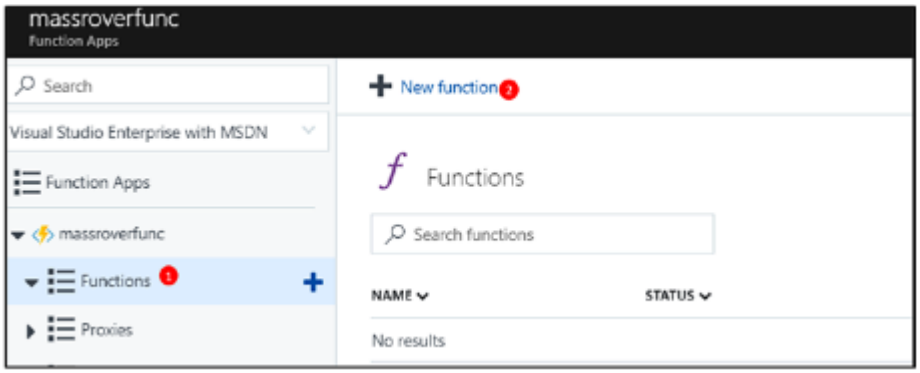
Gambar 6.1 Azure Function App Creation Blade

Kita akan melihat dua pengaturan di sini paket hosting dan penyimpanan. Paket hosting memiliki dua opsi Paket Konsumsi dan Paket Layanan Aplikasi. Kita juga harus menentukan akun penyimpanan serbaguna untuk Aplikasi Fungsi.

- Paket Konsums*i:* Instans host Azure Function disediakan dan dihapus alokasinya secara dinamis berdasarkan pemicu peristiwa. Ini memiliki penskalaan otomatis dan batasan teoretis untuk eksekusi bersamaan yang tidak terbatas, karena infrastruktur yang mendasarinya sepenuhnya dinamis. Paket ini diberi harga berdasarkan penggunaan. Durasi maksimum untuk eksekusi fungsi adalah 10 menit, dan pengaturan default adalah 5 menit.
- *Paket Layanan Aplikasi:* Host Azure Function berjalan pada SKU Paket Layanan Aplikasi yang dialokasikan. Host Fungsi memiliki sumber daya khusus yang dialokasikan, dan penskalaan dapat disiapkan dengan aturan penskalaan otomatis Paket Layanan Aplikasi.
- *Fungsi dapat berjalan selama lebih dari 10 menit*. Ini adalah pilihan yang baik untuk dipertimbangkan untuk kasus-kasus ketika peristiwa hampir terus-menerus dipicu atau waktu eksekusi fungsi memerlukan lebih dari 10 menit atau pengaturan VNET/VPN.

Azure Function memerlukan akun penyimpanan serbaguna, yang digunakan untuk mengelola pemicu dan log peristiwa. Saat menggunakan Consumption Plan, kode dan konfigurasi juga disimpan di akun yang dipilih. Buat Aplikasi Azure Function dan buka. Di bawah Aplikasi Function (massroverfunc), Anda akan melihat tiga opsi berbeda: Functions, Proxies, dan Slots. Satu Aplikasi Function dapat memiliki banyak fungsi; setiap fungsi dapat memiliki pemicu

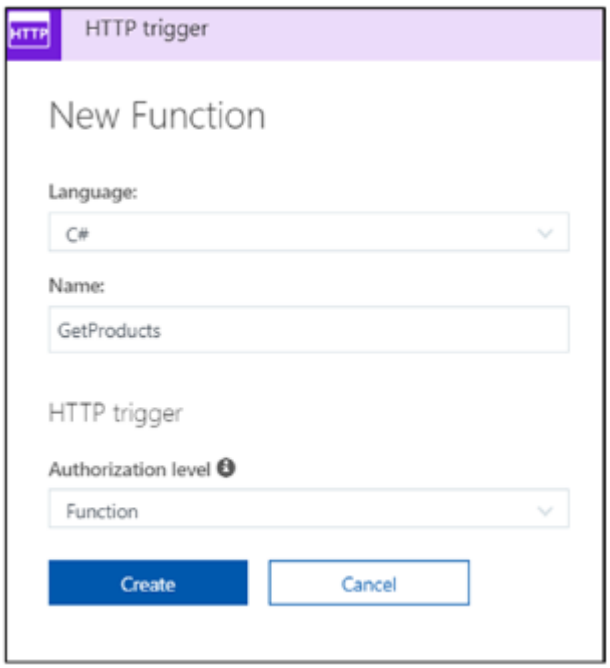
peristiwanya sendiri. Buku ini berfokus pada pemicu peristiwa HTTP. Klik "Functions," lalu klik "New Function." (Gambar 6.2).



Gambar 6.2 Buat Fungsi Baru

Fungsi Pemicu HTTP

Ini akan membuka daftar opsi. Pilih "Pemicu HTTP", yang akan membuka bilah pembuatan fungsi pemicu HTTP (Gambar 6.3).



Gambar 6.3 Bilah Pembuatan Fungsi Pemicu HTTP

Pilih bahasa dan beri nama fungsi. Selain itu, untuk pemicu HTTP, kita harus memilih tingkat Otorisasi. Pengalaman pengembangan yang tersedia di portal tidak memadai dalam sebagian besar kasus, jadi pengembang menggunakan Azure Functions SDK dengan IDE seperti Visual Studio untuk pengembangan di dunia nyata. Buku ini tidak membahas aspek pengembangan fungsi Azure, tetapi mencakup beberapa poin penting untuk dipertimbangkan. Salah satunya adalah tingkat Otorisasi untuk pemicu HTTP. Pemicu dapat ditulis dalam C#, F#, dan JavaScript. Fungsi selalu berjalan sebagai mode asinkron.

6.3 TINGKAT OTORISASI FUNGSI PEMICU HTTP

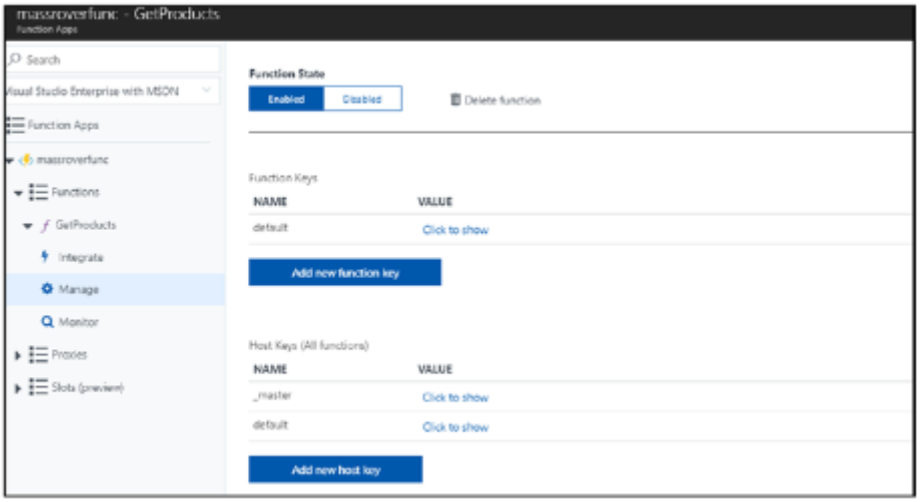
Portal menawarkan tiga opsi untuk tingkat otorisasi pemicu HTTP, tetapi saat Anda membuat pemicu HTTP menggunakan SDK, Anda akan memiliki lima opsi.

1. *Fungsi:* Otorisasi didasarkan pada pasangan nilai kunci tingkat fungsi. Kunci ini bertindak sebagai kunci API untuk fungsi tersebut. Kunci dapat diteruskan dalam permintaan HTTP, baik sebagai parameter kueri atau dalam header. Saat meneruskan kunci dalam kueri, gunakan kode string kueri, dan saat meneruskannya melalui header, gunakan header x-functions-key. Otorisasi tingkat fungsi dicakup untuk setiap fungsi individual, yang menghasilkan kunci yang berbeda untuk setiap fungsi. Ini adalah opsi default. String kueri yang dikirim adalah sebagai,
   `https://<app>.azurewebsites.net/api/<funcname>?code=<key>`
2. *Anonim:* Mengizinkan pemicu peristiwa HTTP tanpa kunci apa pun.
3. *Admin:* Memerlukan kunci host. Kunci host dibuat pada cakupan Aplikasi Fungsi. Jadi, satu kunci host dapat digunakan di beberapa fungsi yang memiliki level otorisasi yang ditetapkan ke Admin.
4. *Sistem*: Memerlukan kunci utama. Kunci utama adalah kunci host khusus yang dapat digunakan untuk mengelola host Fungsi. Kecuali fungsi Anda melakukan eksekusi logika manajemen, level otorisasi ini tidak diperlukan, dan tidak boleh digunakan.
5. *Pengguna*: Level otorisasi ini tidak berbasis kunci, tetapi memerlukan header keamanan yang valid (seperti header otorisasi) dalam permintaan.

Level otorisasi di atas (kecuali otorisasi level Pengguna) berfungsi menggunakan dua jenis kunci yang berbeda.

1. *Kunci host:* Kunci host dicakup pada level Aplikasi Fungsi. Kunci utama (diberi nama _master) adalah kunci host khusus yang tidak dapat dicabut tetapi dapat diperbarui.
2. *Kunci fungsi*: Kunci fungsi dicakup pada fungsi individual. Satu fungsi dapat memiliki banyak kunci fungsi.

Gambar 6.4 menunjukkan kunci ini, di bawah bagian Kelola dari fungsi yang dipilih. Meskipun kunci host dikelola di tingkat Aplikasi Fungsi, kunci tersebut juga ditampilkan di bawah fungsi yang dipilih.



Gambar 6.4 Tombol Aplikasi Fungsi

Mengonfigurasi Fungsi

Arahkan ke bagian Integrasi fungsi, yang ditunjukkan pada Gambar 6.5.

Gambar 6.5 Bagian Integrasi Fungsi

Di bagian ini, kita dapat mengatur nama parameter permintaan HTTP, tingkat otorisasi, metode HTTP yang berlaku untuk memanggil fungsi, mode pemicu HTTP (apakah itu pemicu HTTP sederhana atau web hook), dan templat rute. Templat rute mudah; kita dapat menentukan parameter menggunakan kurung kurawal. Jika Anda mengklik editor Lanjutan, Anda dapat mengedit dokumen JSON untuk konfigurasi yang ditentukan.

```
{
  "bindings": [
    {
      "authLevel": "function",
      "name": "req",
      "type": "httpTrigger",
      "direction": "in",
      "methods": [
        "get",
        "post"
      ]
    },
    {
      "name": "$return",
      "type": "http",
      "direction": "out"
    }
  ],
  "disabled": false
}
```

Dokumen JSON ini mendefinisikan konfigurasi dalam struktur yang lebih ramah pengembang. Kita dapat mengedit dokumen ini untuk membuat perubahan konfigurasi pada fungsi.

Terapkan versi fungsi yang berfungsi, yang akan mengembalikan beberapa data dalam respons HTTP. Anda dapat memilih untuk menulis logika di portal atau menggunakan IDE. Cuplikan kode di bawah ini mengembalikan beberapa produk (seperti API MassRover di bab sebelumnya). Kode ditulis dalam C# menggunakan Visual Studio.

Pengembangan dan penerapan Azure Functions berada di luar cakupan buku ini. Cara termudah untuk mengembangkan dan menerbitkan fungsi Azure adalah dari Visual Studio itu sendiri. Untuk melakukannya, pengembang harus memasang alat Azure Function dan WebJobs. Ini dapat dilakukan menggunakan alat dan ekstensi Visual Studio.

```
public static class GetProducts
{
   [FunctionName("GetProducts")]
   public static IActionResult Run(
      [HttpTrigger(AuthorizationLevel.Function, "get","post", Route = null)]
      HttpRequest req, TraceWriter log)
   {
      var products = new List<Product>
      {
         new Product { Id = 1, Name = "Lithim L2"},
         new Product { Id = 2, Name = "SNU 61" }
      };
      var data = JsonConvert.SerializeObject(products);
      return (ActionResult)new OkObjectResult(data);
   }
}

public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public DateTime? ModifiedDate { get; set; }
}
```

Setelah Anda menerbitkan, Anda bisa mendapatkan URL fungsi di portal. Karena level otorisasi ditetapkan ke Fungsi, kita harus meneruskan kunci fungsi baik dalam kueri maupun di header. Contoh URL akan berada dalam format berikut:

```
https://<nama aplikasi fungsi>.azurewebsites.net/api/<nama fungsi>?code={key}
```

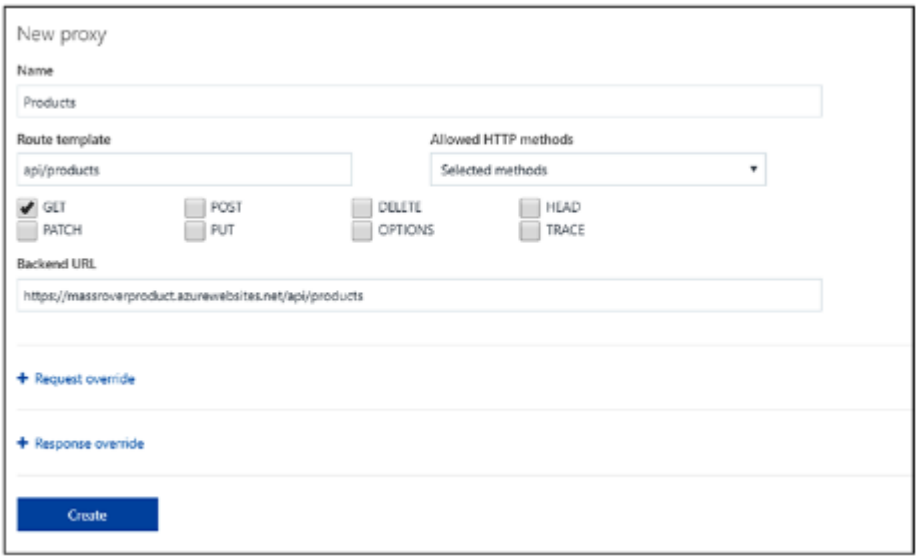
Perhatikan juga bahwa satu fungsi pemicu HTTP dapat menerima beberapa kata kerja HTTP. Opsi ini dapat digunakan untuk membuat titik akhir CRUD cepat dari suatu entitas dengan pernyataan switch, yang dapat melayani permintaan CRUD untuk suatu entitas.

6.3 AZURE FUNCTION PROXIES

Azure Functions Proxies menyediakan serangkaian inti alat pengembangan API yang secara khusus cocok untuk pengembang API tanpa server. Azure Functions Proxies memungkinkan Anda untuk menggabungkan beberapa API di seluruh fungsi dan layanan

menjadi satu permukaan API terpadu. Azure Functions Proxies dapat menautkan ke layanan backend eksternal atau Azure Functions lainnya, atau memberikan hasil dari titik akhir tiruan. Pengalaman portal Azure Functions Proxies tampaknya terbatas, tetapi Azure menyediakan dokumen JSON untuk mendefinisikan Azure Functions Proxies.

Pertama, kita akan membuat proksi Fungsi Azure yang secara langsung memanggil layanan backend yang ada. Kemudian, kita akan mengedit dokumen JSON untuk menyertakan lebih banyak titik akhir permukaan API. Navigasi ke Aplikasi Fungsi dan klik Proksi untuk membuat Proksi Fungsi Azure. Anda akan melihat bilah pembuatan Proksi Fungsi Azure seperti yang ditunjukkan pada Gambar 6.6.



Gambar 6.6 Bilah Pembuatan Proksi Fungsi Azure

Pada tangkapan layar yang ditunjukkan pada Gambar 6.6, Proksi Fungsi Azure dibuat untuk layanan backend. Templat rute menentukan jalur perutean, dan kita memiliki opsi untuk mengganti permintaan dan respons. Klik Buat dan Proksi Fungsi Azure akan dibuat, dan Anda akan melihat URL proksi. Sekarang kita telah membuat proksi untuk URL layanan backend.

Kita dapat memiliki beberapa proksi dan permintaan serta respons yang disesuaikan, tetapi kita akan melakukannya di dokumen proxies.json daripada di portal. Klik Editor lanjutan ini akan membuka jendela editor di browser, dan file proxies.json akan terbuka. Contoh kode di bawah ini menunjukkan proxies.json dengan proksi tambahan yang ditambahkan sebagai titik akhir tiruan.

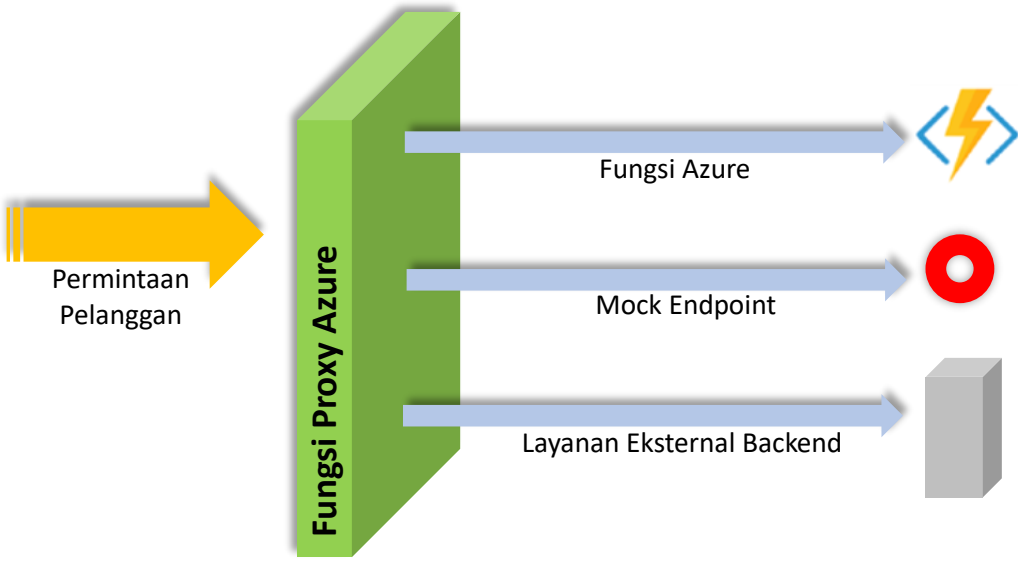
proxy.json

```
{
  "$schema": "http://json.schemastore.org/proxies",
  "proxies": {
    "Products": {
      "matchCondition": {
        "route": "/api/products"
```

```
        },
        "backendUri": "https://massroverproduct.azurewebsites.net/api/products"
        },
      "Customers": {
        "matchCondition": {
          "route": "/api/customers",
          "methods":[
            "GET",
            ]
          },
        "requestOverrides":{
          "request.header":{
            "api-version":"v1"
          }
        },
        "responseOverrides":{
          "response.statusCode": "200",
          "response.statusReason": "OK",
          "response.body":"[{\"name\":\"customer1\"},
            {\"name\":\"customer2\"}]"
          }
        },
        "Partners":{
          "matchCondition":{
            "route":"api/partners",
              "methods":[
                "GET"
              ],
          },
        "backendUri":"https://localhost/api/HttpTriggerCSharp1"
      }
    }
}
```

Seperti yang dapat Anda lihat, dokumen di atas memiliki tiga proksi: Produk, Pelanggan, dan Mitra. Proksi Produk memancarkan pengaturan yang kami pilih pada Gambar 6-6, yang mendengarkan `rute api/products` dan memanggil URL layanan backend eksternal. URL layanan backend ini juga dapat berupa Fungsi Azure lainnya. Proksi Pelanggan memaparkan titik akhir tiruan pada rute api/customers, dengan pengaturan tambahan untuk penggantian permintaan dan respons.

Proksi Mitra memanggil Fungsi dalam Aplikasi Fungsi yang sama dengan demikian, penggunaan localhost juga berfungsi di Azure. Jika Fungsi tersebut milik Aplikasi Fungsi yang berbeda, maka fungsi tersebut harus dipanggil dengan URL lengkapnya. Model di atas menyediakan satu konsumen permukaan API, dan Proksi Fungsi Azure mengabstraksikan detail implementasi. Gambar 6.7 mengilustrasikan hal ini.



Gambar 6.7 Implementasi Azure Function Proxy

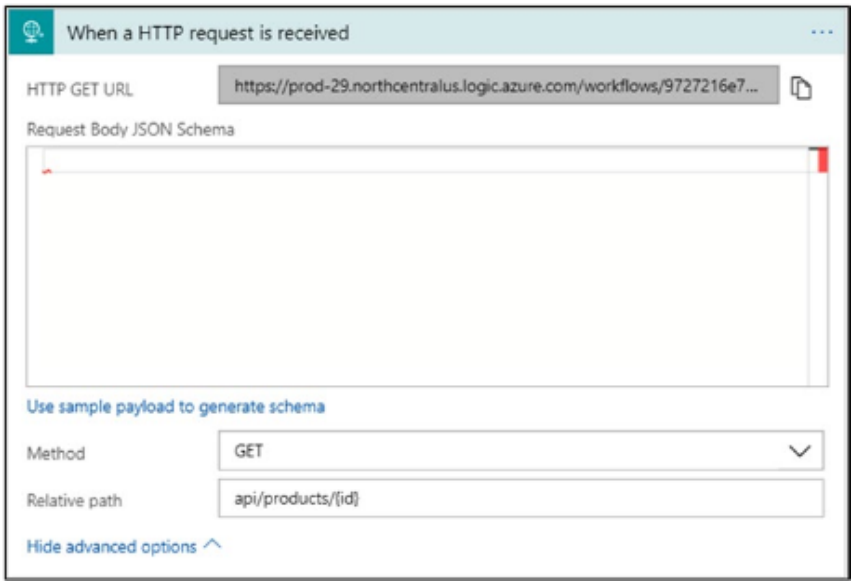
Perhatikan bahwa Azure Function Proxy berperan sebagai API Gateway, dan dapat digunakan untuk meniru API juga. Dalam konteks ini, Azure Function Proxy menyediakan beberapa fungsi Azure API Management, tetapi Azure API Management adalah layanan platform manajemen API yang lebih kaya dan berfungsi penuh, yang menyediakan kontrol tingkat lanjut, fitur, dan alur kerja manajemen API tambahan seperti portal pengembang dan manajemen langganan. Jika Anda mengintegrasikan beberapa API dengan kontrol sederhana atas permintaan dan respons HTTP, Azure Function Proxy lebih disukai daripada Azure API Management (dan lebih murah).

6.4 AZURE LOGIC APPS

Azure Logic App adalah penawaran BaaS tanpa server, yang berguna untuk membuat integrasi dan alur kerja. Aplikasi ini memiliki banyak konektor bawaan untuk aplikasi dan layanan dengan blok pernyataan bersyarat. Untuk kustomisasi terperinci, aplikasi ini menawarkan bahasa logis. Aplikasi Logika berguna dalam pengembangan API karena memiliki pemicu HTTP yang dapat bertindak sebagai titik akhir API.

Navigasi ke portal Azure, cari Aplikasi Logika, dan buat satu.

Proses pembuatannya mudah. Anda harus memberi nama aplikasi dan memilih langganan, grup sumber daya, dan lokasi. Setelah aplikasi dibuat, navigasikan ke Aplikasi Logika. Ada banyak templat yang dapat dipilih, tetapi untuk latihan ini, pilih templat Aplikasi Logika Kosong untuk memulai dari awal. Perancang portal menyediakan sejumlah pemicu yang berbeda. Cari "HTTP" dan Anda akan menemukan pemicu permintaan HTTP.



Gambar 6.8 Pemicu Permintaan HTTP Aplikasi Logika Azure

Kita dapat mengatur metode HTTP dan jalur relatif untuk permintaan, dan kita juga dapat mengatur skema JSON untuk badan permintaan. Kita dapat menghubungkan tindakan ke pemicu, dan menambahkan tindakan ke pemicu permintaan HTTP. Dalam tindakan, Anda dapat memperoleh masukan yang mungkin dari langkah sebelumnya, dalam hal ini badan permintaan dan parameter ID. Terakhir, saat Anda menyimpan Aplikasi Logika di perancang, ia akan menghasilkan URL lengkap untuk pemicu HTTP. URL akan mirip dengan yang di bawah ini.

```
https://prod-29.northcentralus.logic.azure.com/workflows/97272
16e7a4442ccb4b3f96e863374be/triggers/manual/paths/invoke/api/
products/{id}?api-version=2016-10-01&sp=%2Ftriggers%2Fmanual%2F
run&sv=1.0&sig=EuqlcN4fV8qKIfLQbteKaTkE3V5v_OkM5w-FgShMnXw
```

Anda mungkin memperhatikan bahwa URL berisi jalur relatif setelah segmen manual/jalur/invoke, yang diisi dalam permintaan. Selain itu, ada beberapa kunci dan ID untuk Aplikasi Logika guna mengidentifikasi permintaan. Meskipun URL untuk pemicu HTTP Azure Logic terlihat rumit, URL ini dapat diintegrasikan dengan Azure API Management atau Azure Function Proxies, sehingga menghasilkan permukaan API terpadu dengan fragmen URI yang ramah pengguna.

API Tanpa Server di AWS

Tanpa server adalah istilah asli untuk AWS, dan AWS memiliki penawaran FaaS tanpa server yang disebut Lambda. Di bagian ini, kita akan membangun layanan backend menggunakan Lambda dan mengirimkan permukaan API melalui AWS API Gateway.

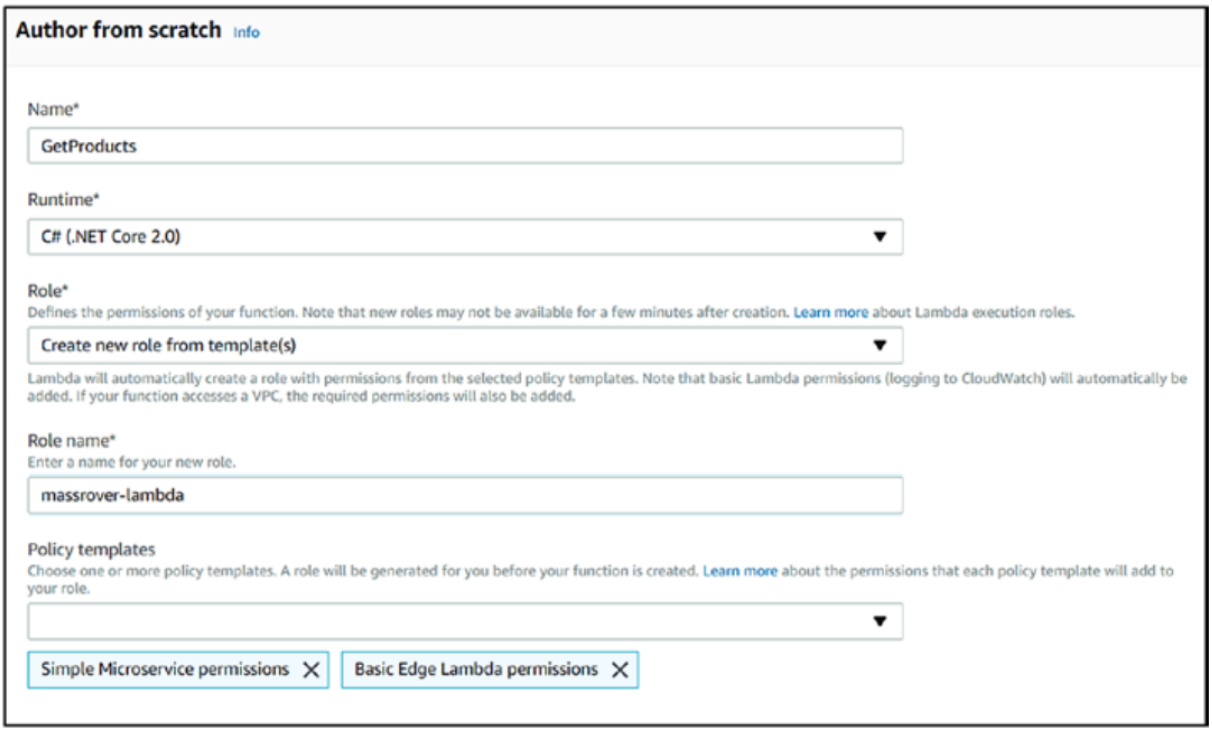
6.5 AWS LAMBDA

AWS Lambda adalah penawaran FaaS dari AWS. Lambda sendiri mempopulerkan model tanpa server dan membuka jalan bagi daya tarik tanpa server dalam pengembangan

perangkat lunak komersial. Ia mendukung berbagai bahasa dan platform. Fungsi AWS Lambda sendiri tidak mendukung pemicu HTTP, jadi kita perlu melengkapi API tanpa server melalui AWS API Gateway dan menerima permintaan HTTP serta meneruskannya ke AWS Lambda. Di bagian ini, kita akan melihat cara membuat dan mengonfigurasi AWS Lambda, dan di bagian berikutnya kita akan fokus pada pengintegrasiannya dengan AWS API Gateway.

Membuat Fungsi AWS Lambda

Buka portal AWS dan cari Lambda untuk membuat fungsi. Kita dapat membuat fungsi dari awal, dari templat yang tersedia, atau dari repositori aplikasi tanpa server. Di bagian ini, mari kita buat fungsi Lambda dari awal. Pilih Penulis dari awal dan masukkan detail yang diperlukan. Gambar 6.9 mengilustrasikannya.



Gambar 6.9 AWS Membuat Lambda dari Awal

Masukkan nama fungsi Lambda, pilih runtime, dan pilih peran untuk fungsi Lambda. Peran ini menentukan izin untuk fungsi tersebut. Anda dapat menggunakan peran yang sudah ada, membuat peran baru, atau membuatnya dari templat kebijakan. Dalam kasus ini, peran bernama massrover-lambda telah dibuat dari dua templat kebijakan: izin Simple Microservices dan izin Basic Edge Lambda. Setelah peran dibuat, AWS akan membuka jendela konfigurasi fungsi. Jendela tersebut memiliki banyak bagian, dan buku ini menjelaskannya secara singkat. Gambar 6.10 menunjukkan perancang AWS Lambda.

Gambar 6.10 Perancang AWS Lambda

Pada layar yang ditunjukkan pada Gambar 6.10, jika Anda menggulir ke bawah, Anda akan melihat opsi konfigurasi lainnya:

- *Kode fungsi:* Ini adalah bagian tempat kita mengunggah kode Lambda sebagai file zip, atau kita dapat mengarahkan ke file/folder di AWS S3. Di bagian ini, kita juga dapat mengatur runtime Lambda. Handler menentukan nilai pengendali Lambda; dalam konteks bahasa C#, ini dalam bentuk yang ditunjukkan di bawah ini:

  ```
  assembly::namespace.class-name::method-name untuk runtime C#.
  ```

- *Variabel lingkungan:* Ini menetapkan variabel lingkungan apa pun yang diperlukan untuk menjalankan fungsi Lambda. Dalam konteks eksekusi Lambda di C#, variabel lingkungan dapat dibaca dari metode assembly sistem, seperti di bawah ini:

  ```
  Environment.GetEnvironmentVariable("key");
  ```

  Di bagian variabel lingkungan, kita dapat menetapkan AWS KMS untuk mengenkripsi variabel lingkungan yang sensitif. Anda juga dapat menggunakan variabel lingkungan untuk menyimpan string koneksi dan variabel eksekusi lainnya.
- *Tag:* Ini adalah pasangan nilai kunci yang digunakan untuk mengelompokkan dan memfilter fungsi Lambda. Fungsi Lambda yang terkait dengan satu unit kerja logis dapat ditandai dengan nilai yang sama dan difilter.
- *Peran eksekusi:* Peran eksekusi adalah izin untuk fungsi Lambda. Ini ditetapkan pada saat pembuatan (Gambar 6-9), dan kita dapat mengubahnya di sini.
- *Pengaturan dasar:* Di bagian ini, kita dapat mengatur memori maksimum untuk fungsi Lambda. Rentang yang tersedia adalah 128 MB hingga 3008 MB, dan CPU dialokasikan secara proporsional dengan memori. Memori dialokasikan dalam potongan 64 MB. Kita juga dapat mengatur batas waktu fungsi Lambda. Waktu eksekusi Lambda maksimum adalah 300 detik.
- *Jaringan:* Fungsi Lambda AWL berjalan dalam VPC default yang dikelola sistem. Di bagian ini, kita dapat mengonfigurasi VPC AWS kustom, yang membuatnya cukup aman sehingga dapat mengakses sumber daya seperti basis data dalam VPC yang ditentukan.
- *Debugging dan penanganan kesalahan:* Jika fungsi Lambda gagal selama eksekusi, fungsi tersebut akan mencoba lagi dua kali secara default. Jika eksekusi tidak berhasil

setelah percobaan ulang, kita dapat mengonfigurasi fungsi tersebut untuk menempatkan pesan dalam topik AWS SNS atau AWS SQS. Opsi konfigurasi ini dikenal sebagai DLQ (Dead Letter Queue). Kita dapat menentukan ARN dari SNS atau SQS untuk bertindak sebagai DLQ.

- *Konkurensi:* Konfigurasi ini berkaitan dengan konkurensi eksekusi fungsi Lambda. Kita dapat menetapkan angka tertentu atau menggunakan semua konkurensi akun yang tidak dicadangkan. Akun dicadangkan dengan 1000 eksekusi konkurensi.
- *Audit dan kepatuhan:* Di bagian ini, kita dapat mengaktifkan jejak audit eksekusi Lambda menggunakan AWS CloudTrail.

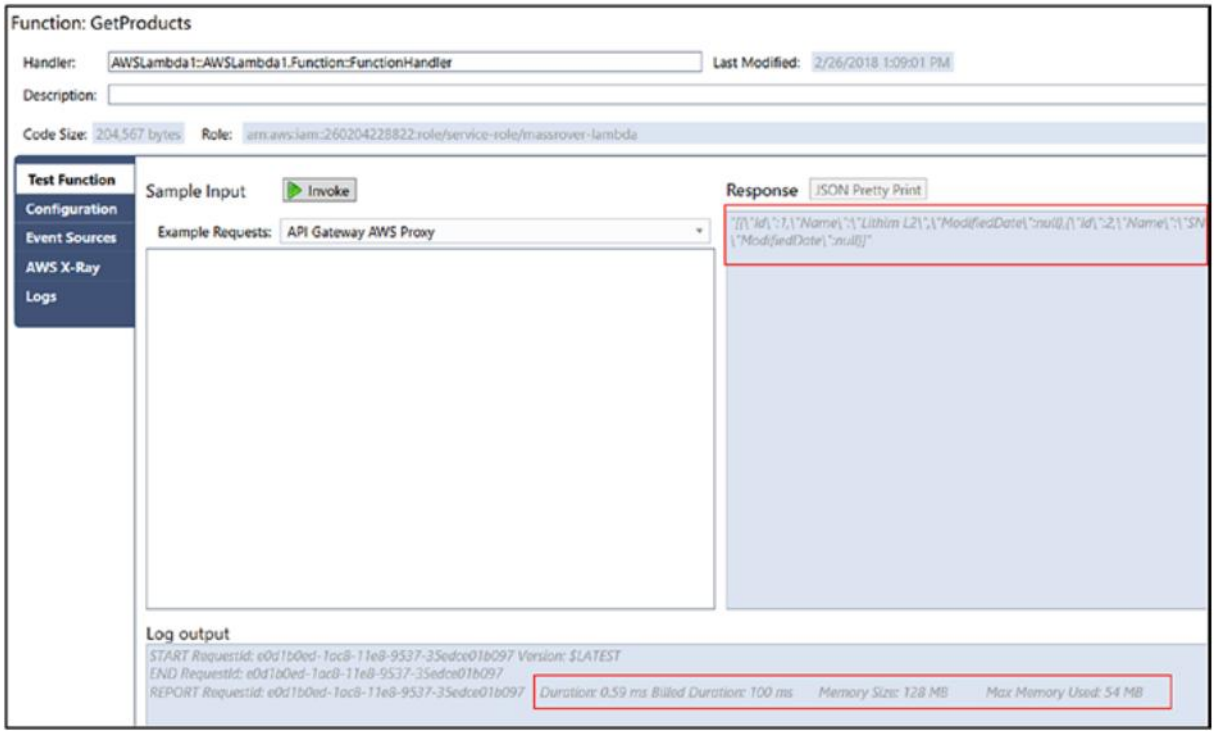
Dengan memilih fungsi Lambda, kita dapat mengunjungi desainer dan mengonfigurasi pengaturan di atas. Setelah pembuatan AWS Lambda, kita dapat menulis kode menggunakan IDE dan menyebarkannya, atau mengunggah paket kode melalui Konsol AWS secara langsung atau dari AWS S3. Potongan kode di bawah ini menunjukkan daftar produk yang dikembalikan di AWS Lambda dalam C#:

```
public class Function
{

   public string FunctionHandler(ILambdaContext context)
   {

      List<Product> products = new List<Product>
      {
         new Product { Id = 1, Name = "Lithim L2" },
         new Product { Id = 2, Name = "SNU 61" }
      };

      var data = JsonConvert.SerializeObject(products);
      return data;
   }
}

public class Product
{
   public int Id { get; set; }
   public string Name { get; set; }
   public DateTime? ModifiedDate { get; set; }
}
```

`ILambdaContext` yang diinjeksikan akan memberikan akses ke parameter eksekusi.

Pengembangan dan penyebaran AWS Lambda berada di luar cakupan buku ini. Anda dapat mengembangkan dan menerbitkan AWS Lambda dari Visual Studio itu sendiri menggunakan alat pengembangan AWS untuk Visual Studio. Setelah penerbitan, kita dapat memanggil Lambda dari Visual Studio, karena kita masih belum menghubungkan peristiwa HTTP ke Lambda. Ini dilakukan menggunakan integrasi dengan AWS API Gateway. Gambar 6.11

menunjukkan pemanggilan AWS Lambda dari Visual Studio. Anda dapat melihat respons di sisi kiri. Perhatikan juga keluaran log, yang menunjukkan memori yang digunakan, waktu eksekusi, dan waktu yang dapat ditagih dari Lambda.

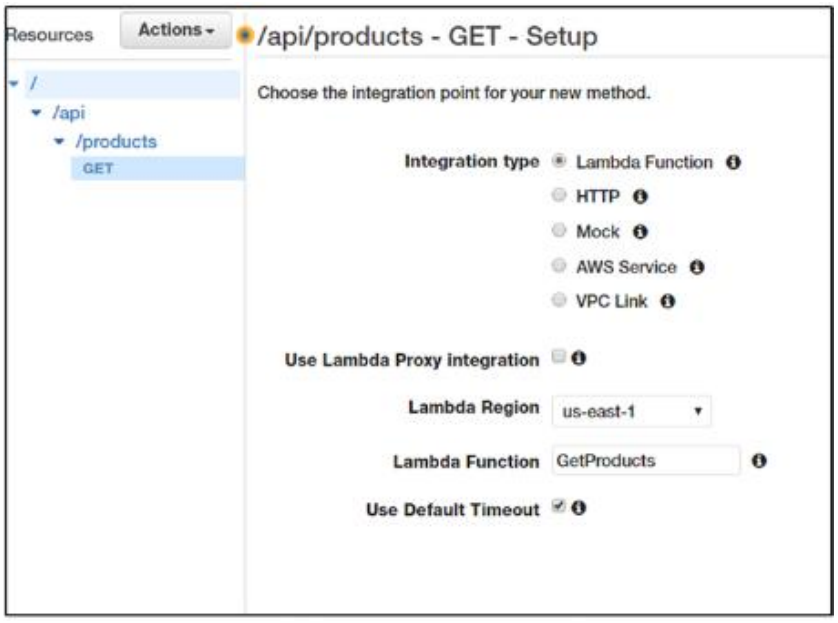


Gambar 6.11 Panggil AWS Lambda dari Visual Studio

6.6 MENYIAPKAN AWS LAMBDA DENGAN AWS API GATEWAY

Di bagian ini, kita akan menyiapkan integrasi antara AWS Lambda dan AWS API Gateway, yang mengaktifkan API tanpa server yang berfungsi di AWS. Buka Konsol AWS, buat API baru di AWS API Gateway, lalu buat dua sumber daya di bawah api, bernama "api" dan "products." (rujuk bab 4 untuk detail selengkapnya tentang cara melakukan tindakan ini). Pilih sumber daya api/products dan buat metode GET, yang ditunjukkan pada Gambar 6.12.

Di metode tersebut, pilih jenis integrasi Fungsi Lambda. Pilih wilayah, lalu pilih Fungsi Lambda (saat Anda mengetik nama Lambda, nama tersebut akan muncul di menu tarik-turun). Saat Anda mengeklik simpan, AWS akan menampilkan dialog yang menunjukkan bahwa Anda memberikan izin kepada API Gateway untuk memanggil fungsi Lambda, yang menunjukkan ARN fungsi Lambda.



Gambar 6.12 Integrasi AWS API Gateway Lambda

Secara teknis, integrasi selesai setelah langkah ini, dan Anda dapat mengujinya menggunakan tautan pengujian di API Gateway. Kita kemudian dapat menerapkan API dengan membuat tahapan (lihat bab 4 untuk detail selengkapnya). API yang diterapkan akan memiliki URL seperti ini:

```
https://vtv6xchkkk.execute-api.us-east-1.amazonaws.com/prod/api/products
```

Ringkasan

Serverless merupakan evolusi dari komputasi awan yang semakin populer karena pengalaman pengembangan yang ditawarkannya dan komputasi berbasis utilitas yang lebih terperinci. Karena itu, serverless bukan hanya cara baru untuk mengembangkan aplikasi, tetapi juga menyediakan model ekonomi baru di dunia komputasi awan. API serverless menjadi pusat perhatian karena manfaat yang disebutkan sebelumnya, serta manfaat ekonominya. Namun, platform serverless bisa mahal dalam beberapa kasus. Sifat platform serverless yang berbasis peristiwa dan dapat di-host sendiri merupakan alasan utama lainnya mengapa pengembang dapat dengan mudah mengintegrasikan model ini dengan aplikasi lama apa pun.

BAB 7
DESAIN DAN PENGEMBANGAN PRAKTIS

Lanskap desain dan pengembangan aplikasi modern sangat dipengaruhi oleh teknologi yang sedang berkembang, perubahan pola pikir pengembang, dan desakan untuk bertahan hidup serta tekanan dari rekan sejawat untuk berinovasi. Pada dasarnya, ada permintaan data yang terus meningkat, dan pengembang mengendalikan aliran data dalam aplikasi. API membantu pengembang untuk mengendalikan dan mengatur aliran data di antara layanan dan solusi dengan lebih efektif.

Dalam desain dan pengembangan praktis, ada beberapa model umum yang digunakan pengembang untuk mendapatkan manfaat maksimal dari strategi pengembangan API. Bab ini menjelaskan beberapa skenario umum dan praktik desain yang digunakan dalam pengembangan di dunia nyata.

7.1 DESAIN YANG MENGUTAMAKAN KONTRAK

Sesuai namanya, desain yang mengutamakan kontrak difokuskan pada kontrak layanan; pendekatan desain yang mengutamakan kontrak adalah tentang memulai pengembangan dengan mendefinisikan kontrak layanan dan titik akhir layanan terkait. Selain skema kontrak layanan dan titik akhir, aspek lain seperti pembuatan versi, skema URI, dan penamaan, disertakan dalam pendekatan desain.

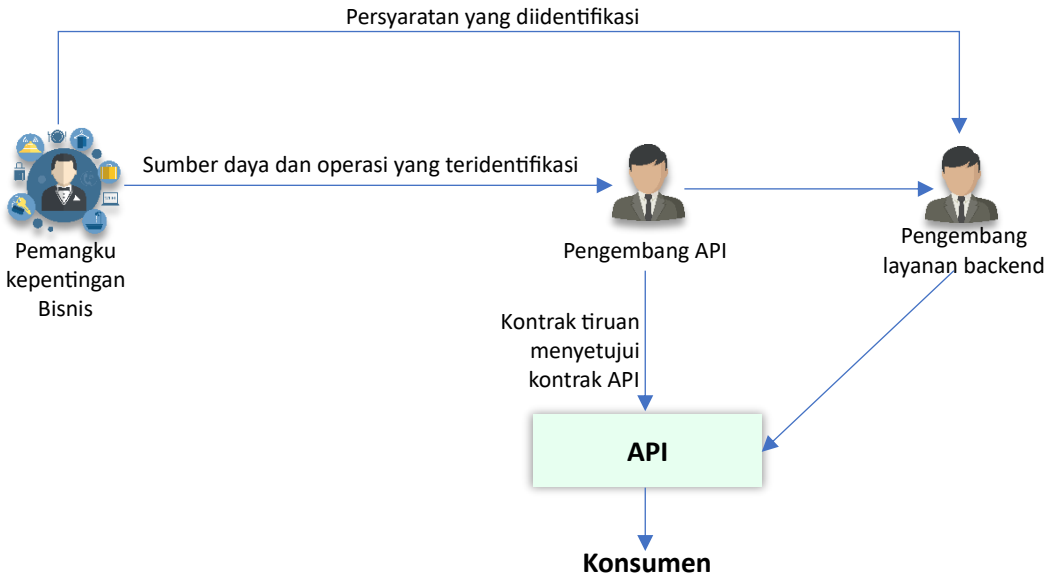
Menentukan kontrak layanan yang jelas membantu memahami model domain bisnis. Ada berbagai teknik untuk mengidentifikasi ini, tetapi event storming adalah salah satu praktik yang digunakan secara luas untuk mengidentifikasi model dan operasi domain aliran bisnis. Model dan operasi domain yang teridentifikasi adalah benih untuk desain yang mengutamakan kontrak.

Dalam dunia DDD (pengembangan berbasis domain) dan event storming, istilah model domain digunakan, tetapi dalam pendekatan desain yang mengutamakan kontrak dan desain layanan RESTful, sangat masuk akal untuk menggunakan istilah sumber daya dan operasi. Saat memulai pengembangan API dengan pendekatan yang mengutamakan kontrak, pertama-tama kita harus menyerap masalah bisnis dan mulai menentukan kontrak layanan (objek JSON yang umum) di tingkat API. Objek JSON ini menentukan skema kontrak layanan; ini menentukan kumpulan hasil dari API atau isi permintaan yang masuk.

Salah satu pendekatan implementasi yang umum dan mudah untuk desain yang mengutamakan kontrak adalah dengan menggunakan tiruan API. API tiruan sangat ampuh dalam desain yang mengutamakan kontrak, ini membantu pengembang dan pemangku kepentingan bisnis untuk memvalidasi model domain sambil menetapkan standar teknis. Ini menghilangkan banyak penulisan ulang dan diskusi yang sering terjadi pada tahap awal pengembangan.

Baik Azure API Management maupun AWS API Gateway memiliki fasilitas tiruan, yang

memungkinkan Anda menentukan model dan operasi langsung dari portal. Idenya di sini pada dasarnya adalah mempercepat implementasi dan mengidentifikasi kesenjangan alur bisnis dan penggunaan sedini mungkin, serta mengulanginya dengan cepat tanpa mengorbankan standar teknis yang diperlukan untuk desain API yang berkelas. Gambar 7.1 menggambarkan pendekatan ini dalam gambaran tingkat tinggi dan bagaimana pemangku kepentingan yang berbeda terlibat dalam pendekatan desain kontrak terlebih dahulu.



Gambar 7.1 Desain yang Mengutamakan Kontrak

Pendekatan ini memerlukan keterampilan dan praktik strategis yang cukup agar berhasil. Sebab, pengembang API dan pengembang layanan backend.

Persiapan

Pemilik bisnis atau pemilik produk atau koordinator event storming harus mengetahui model desain ini sehingga ia dapat memfasilitasi alur persyaratan untuk mengidentifikasi sumber daya dan operasi serta mengaktifkan pendekatan desain yang mengutamakan kontrak. Pengembang API harus terampil dalam memetakan domain yang sedang ditemukan ke sumber daya dan titik akhir API. Ia juga harus berpengalaman dengan platform pengembangan/tiruan API yang dipilih untuk mempercepat dan mengatur konsumen depan dan pengembang layanan backend.

Standar API tingkat dasar berbeda di antara pengembang dan tim. Selama latihan di atas, operasi dan sumber daya yang teridentifikasi dapat berubah dalam frekuensi tinggi tetapi tidak dengan standar API yang disepakati. Bab 3 memberikan panduan awal yang baik tentang standar API, panduan ini telah teruji dan dapat diperluas ke persyaratan khusus.

Tantangan Utama

Tantangan utama dari desain yang mengutamakan kontrak adalah mengidentifikasi sumber daya dan operasi yang tepat untuk diekspos sebagai API. Akan ada perubahan pada frekuensi tinggi pada tahap awal penemuan domain tetapi ini secara bertahap akan mencapai keadaan yang lebih stabil; sumber daya yang teridentifikasi tidak akan berubah (kecuali

propertinya). Misalnya, dalam sistem klinis sederhana, seorang praktisi medis adalah sumber daya yang teridentifikasi. Atribut sumber daya ini dan operasi yang sesuai akan berubah, tetapi praktisi sumber daya tidak boleh berubah ke sumber daya yang berbeda dalam frekuensi tinggi.

Orang yang memfasilitasi latihan ini harus memastikan domain bisnis ditemukan secara progresif; jika orang tersebut tidak menyusun strategi penemuan domain secara progresif, hal itu akan menjadi hambatan besar dan tantangan bagi seluruh proses. Tantangan utama kedua adalah tingkat keterampilan teknis pengembang API, karena satu-satunya tujuan pendekatan desain ini adalah untuk mempercepat pengembangan dan mengaktifkan konsumen API secepat mungkin. Jika ada persyaratan untuk aplikasi seluler, maka pengembang seluler harus diaktifkan sejak hari pertama, segera setelah persyaratan mulai mengalir, titik akhir API dengan operasi dan standar yang ditentukan harus tersedia sejak hari itu untuk mencapainya, sehingga pengembang API yang tidak memiliki keterampilan teknis dalam platform yang dipilih akan menyebabkan hambatan dalam implementasi.

Terakhir, tim pengembang API, pengembang layanan backend, dan konsumen harus menyadari dan disiplin dengan standar API yang disetujui. Perhatikan, seperti pada Gambar 7.1 pendekatan ini memiliki banyak peran, tetapi dalam proyek yang lebih kecil ini tidak memetakan ke individu yang berbeda yang memainkan peran tersebut. Mungkin ada situasi di mana satu pengembang tunggal/satu tim tunggal melakukan semua hal.

Kapan Tidak Mencobanya

Pendekatan desain ini tidak akan cocok dalam skenario alur penggunaan tingkat tinggi. Dalam model event storming, diskusi berulang dengan peningkatan ketelitian dan detail; dalam kasus tertentu, hal-hal dibahas pada tingkat yang sangat tinggi, yang membuatnya sulit untuk mengekstrak sumber daya dan operasi yang diperlukan untuk disertakan dalam API. Jika diskusi domain pada tingkat yang sangat tinggi, maka pendekatan desain kontrak terlebih dahulu tidak akan cocok.

Jika standar atau disiplin API yang disetujui tidak tersedia dan tidak diformalkan dengan baik, dalam situasi seperti itu tidak disarankan untuk menggunakan pendekatan desain kontrak terlebih dahulu, karena akan menciptakan konflik dan komunikasi yang tidak produktif di antara tim. Pertama, pastikan semua orang memikirkan standar dengan cara yang sama dan tentukan standar ke tingkat tertentu.

7.2 API DALAM LAYANAN MIKRO

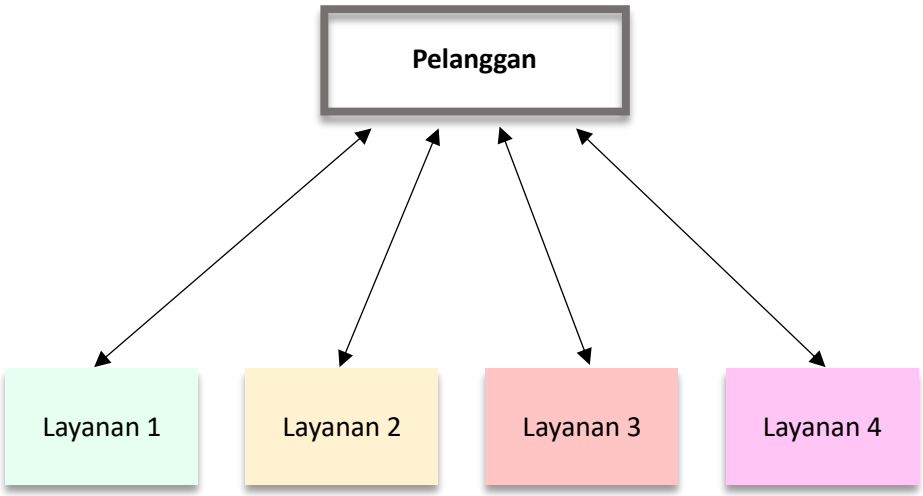
Layanan mikro adalah arsitektur desain aplikasi yang sedang berkembang. Layanan mikro terdiri dari banyak layanan yang berbeda dan layanan ini berkomunikasi satu sama lain menggunakan kontrak layanan yang ditentukan (tidak harus dalam HTTP) dan mereka memaparkan sumber daya dan operasi kepada konsumen (aplikasi klien) melalui API. Untuk mengatur komunikasi layanan antara klien dan berbagai layanan dalam layanan mikro, gateway API banyak digunakan.

Dalam model ini, ada dua pendekatan berbeda: 1) Desain yang dikoordinasikan oleh klien - desain ini memungkinkan konsumen mengelola orkestrasi layanan, tidak ada

pendekatan terpadu dalam standar API. Setiap layanan dapat memiliki standar API sendiri dan konsumen harus memahaminya secara individual untuk memisahkan layanan. 2) Pola Gateway API - Gateway API digunakan untuk melakukan orkestrasi layanan dan standar API terpadu diterapkan di gateway API, yang memungkinkan masing-masing tim pengembangan layanan memiliki standar sendiri, ini juga memungkinkan untuk membawa layanan lama ke standar API terpadu.

Desain yang dikoordinasikan oleh klien

Desain yang dikoordinasikan oleh klien didasarkan pada layanan individual yang diakses langsung oleh klien untuk menyelesaikan alur bisnis. Dalam konteks ini, setiap layanan mikro memaparkan API-nya sendiri kepada konsumen. Gambar 7.2 menggambarkan hal ini.



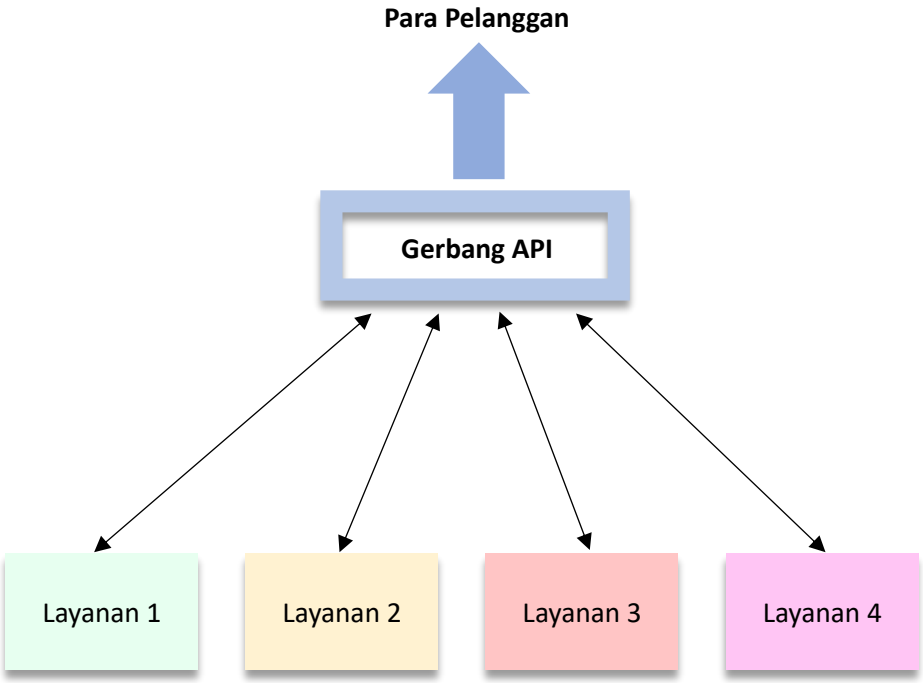
Gambar 7.2 Desain Layanan Mikro Yang Dikoordinasikan Klien

Keuntungan utama dari pendekatan ini adalah memungkinkan konsumen untuk menentukan alur bisnis, bukan alur yang telah ditetapkan sebelumnya dari satu pihak, sehingga memberikan fleksibilitas kepada konsumen. Layanan mikro skala kecil dapat menggunakan desain yang dikoordinasikan klien, terutama ketika tim berkumpul dan memiliki standar yang disetujui.

Standar API harus diikuti oleh setiap tim untuk memastikan pengembangan yang seragam, tetapi ini dapat dengan mudah dilanggar karena tidak ada tata kelola terpusat untuk API. Terdapat ketidaksepakatan umum mengenai pendekatan desain ini dibandingkan dengan desain gerbang API, tetapi pendekatan ini lebih unggul daripada pola pendekatan gerbang API dalam argumen tentang tata kelola yang lebih baik dan titik kegagalan tunggal yang diperkenalkan oleh gerbang API.

Pola Gerbang API

Pola gerbang API memperkenalkan gerbang API sebagai lapisan koordinasi terpusat antara layanan individual dan konsumen. Selain itu, praktik ini membantu mencapai manfaat tambahan seperti logika pra-autentikasi tingkat gerbang, penyimpanan sementara, kontrol aliran, dan masalah lintas sektor lainnya. Gambar 7.3 menggambarkan desain ini.



Gambar 7.3 Implementasi Gerbang API

Pola gerbang API mengeluarkan titik pusat komunikasi dan standar, yang memungkinkan setiap layanan menggunakan standar API mereka sendiri. Memiliki standar API yang berbeda untuk setiap layanan memungkinkan fleksibilitas dalam pengembangan dan membuka kemungkinan penerapan protokol yang berbeda, terutama dalam memodernisasi sistem lama. Dari sudut pandang penerapan, gerbang API menambahkan keamanan berlapis untuk layanan backend, yang memungkinkan layanan backend berada di luar Internet publik. Ada kekhawatiran umum, tentang memiliki lebih banyak logika khusus domain di gerbang API, intinya adalah, apakah baik atau buruk untuk memiliki logika bisnis di gerbang API?

Dengan semakin populernya gerbang API, vendor gerbang API membebani mereka dengan fitur yang dapat digunakan di luar model gerbang. Karena istilah gerbang bukan persyaratan fungsional dan melayani tujuan proxy terbalik; cukup jelas bahwa memasukkan logika bisnis dalam gerbang API BUKAN desain yang baik. Tetapi sekali lagi dalam kasus tertentu memanfaatkan alat/kerangka kerja yang dipilih membuat penerapan menjadi mudah dan cepat. Memiliki konteks bisnis dalam API dikenal sebagai 'Thick API gateway'.

7.3 API UNTUK INTEGRASI PERUSAHAAN

API umumnya digunakan dalam integrasi perusahaan; penggunaannya telah diikuti dari layanan web dengan semua standar WS*. Sekarang, umumnya layanan RESTful digunakan dalam konteks ini untuk memberikan fungsionalitas yang sama. Layanan dan platform RESTful ringan, dan tren modern yang muncul seperti runtime yang dapat dihosting sendiri telah membuatnya lebih fleksibel sehingga dapat ditempatkan di antara banyak sistem besar yang menjadikannya pilihan pertama dalam integrasi perusahaan. API integrasi perusahaan sering kali menangani protokol lama dan standar pengiriman pesan. Penerjemahan data dari protokol

lama ke protokol baru, dan sebaliknya. Integrasi API perusahaan juga memiliki cita rasa modern, misalnya: Perusahaan pembelajaran mendalam mengembangkan mesin AI yang dapat membuat prediksi cerdas menggunakan data keuangan.

Agar ini berfungsi, mesin AI harus memiliki integrasi dengan berbagai sistem akuntansi. Membangun titik integrasi untuk setiap platform akuntansi yang berbeda akan menjadi rumit dan memakan waktu. Jadi, perusahaan integrasi akan membantu mereka membangun integrasi API standar untuk berbagai platform akuntansi. Di dunia perusahaan, integrasi adalah bisnis besar. Faktanya, semua pengembangan aplikasi tingkat perusahaan memiliki semacam integrasi. API membantu membuka aliran data antar sistem. Platform pengiriman pesan berbasis persistensi seperti bus layanan dianggap usang, dan layanan RESTful berbasis HTTP menggantikannya.

Ringkasan

Desain API dan alat terkait dapat membantu mencapai fleksibilitas pengembangan dan model baru orientasi tim. Tidak ada aturan keras dan cepat yang menentukan prinsip desain mana yang harus dipraktikkan dalam situasi tertentu. Setiap desain memiliki pro dan kontranya sendiri dalam skenario apa pun. Membuat layanan RESTful dalam bahasa modern apa pun itu mudah dan didukung dengan baik oleh berbagai alat dan kerangka kerja. Namun, mengembangkan praktik API yang terstandarisasi dengan implementasi yang terstruktur dengan baik melampaui platform teknologi. Buku ini membahasnya secara seimbang, dengan informasi yang tepat untuk memulai pengembangan dan juga dengan menyediakan informasi tentang implementasi dan arsitektur API. Upaya yang terorganisasi di seputar standar dan implementasi ini akan menghasilkan desain API yang baik.

DAFTAR PUSTAKA

Adams, P. (2021). *Cloud API security: Implementing OAuth 2.0 and OpenID Connect with AWS and Azure*. Packt Publishing.

Amazon Web Services, Inc. (2023). *Architecting for the Cloud: AWS Best Practices*. AWS Whitepapers. https://aws.amazon.com/whitepapers/

Amazon Web Services. (2021). *AWS Lambda and serverless computing for API management*. https://aws.amazon.com/lambda/

Azure Documentation Team. (2023). *Overview of API management in Azure*. Microsoft. https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts

Barnes, P., & Li, Q. (2020). *API monitoring and performance optimization in cloud platforms*. Wiley.

Behr, T., & Ghosh, S. (2022). *Building scalable web applications using Azure and AWS*. O'Reilly Media.

Bennett, J. (2021). *Cloud-native APIs: Designing APIs for the cloud-first era*. Apress.

Bhat, R. (2020). *Designing scalable and secure APIs in cloud environments*. Pearson.

Brown, C., & Miller, R. (2020). *Designing RESTful APIs with Azure*. Packt Publishing.

Chen, F. (2021). *Best practices for building APIs with AWS and Azure cloud platforms*. Springer.

Clark, T., & Stevens, M. (2022). *Practical AWS development: Creating secure and scalable APIs*. O'Reilly Media.

Clarke, G. (2021). *Building secure APIs in cloud environments*. Pearson.

Clarke, H., & Lawson, T. (2022). *Implementing fault tolerance in API design with Azure and AWS*. Springer.

Davies, P. (2021). *Introduction to cloud computing and API management*. Springer.

Green, L., & Webb, K. (2020). *Implementing CI/CD pipelines for API development with Azure DevOps and AWS CodePipeline*. Apress.

Grewal, M., & Singh, R. (2021). *Cloud API security best practices with AWS and Azure*. Wiley.

Harris, B., & Wang, J. (2020). *API design patterns and best practices*. Pearson Education.

Jenkins, J. (2021). *Developing RESTful APIs for enterprise applications with AWS and Azure*. Springer.

Kumar, A., & Soni, P. (2023). *Cloud Architecture with AWS and Azure: A guide for developers*. Wiley.

Kumar, S. (2022). *Architecting modern APIs with microservices in AWS and Azure*. Springer.

Lee, K. (2021). *API security in cloud environments: A practical approach using AWS and Azure*. Springer.

Lewis, J., & Langford, J. (2022). *Designing scalable APIs with AWS API Gateway and Azure API Management*. Wiley.

Liu, Z., & Turner, S. (2021). *Microservices architecture with AWS and Azure: A practical guide*. Packt Publishing.

Marshall, D. (2021). *Building APIs for distributed systems with Azure and AWS*. O'Reilly Media.

Martin, R. C. (2020). *Clean architecture: A craftsman's guide to software structure and design*. Prentice Hall.

McKee, S., & Wilson, B. (2022). *API development with Microsoft Azure and AWS*. Microsoft Press.

Microsoft. (2022). *Deploying APIs on Azure*. https://docs.microsoft.com/en-us/azure/api-management/

Nakamura, H. (2020). *Architectural patterns for cloud-based systems*. Addison-Wesley.

Nielsen, T. (2023). *Securing cloud-based APIs with AWS and Azure*. Packt Publishing.

O'Neil, T. (2021). *Building and deploying APIs with AWS Lambda and Azure Functions*. O'Reilly Media.

Patel, M. (2021). *Cloud APIs in practice: A developer's guide to Azure and AWS*. Wiley.

Patel, V., & Gupta, A. (2020). *Microservices and API Gateway patterns in AWS and Azure*. Springer.

Peters, D. (2020). *Hands-on RESTful API design with AWS and Azure*. Packt Publishing.

Ramesh, V., & Paul, M. (2021). *Optimizing APIs in cloud environments with Azure and AWS*. Springer.

Roberts, J. (2020). *Building and deploying cloud-based APIs with Azure and AWS*. Apress.

Robinson, D., & Smith, T. (2023). *Cloud-based API development: Tools, services, and strategies*. Wiley.

Ross, A., & Jacobs, R. (2020). *API design essentials with AWS and Azure*. Apress.

Ross, J. (2021). *RESTful APIs with Microsoft Azure: From theory to practice*. Microsoft Press.

Santos, J. (2022). *Cloud-native API design patterns*. Wiley.

Smith, R. (2022). *Designing efficient and scalable APIs with AWS and Azure*. O'Reilly Media.

Stone, R., & Maxwell, P. (2020). *Integrating third-party APIs with AWS and Azure*. Apress.

Tan, M. (2020). *API Gateway architectures with AWS and Azure*. O'Reilly Media.

Thompson, E. (2021). *Modern API strategies using Azure and AWS*. O'Reilly Media.

Thorne, A. (2022). *API lifecycle management in the cloud: A guide to Azure and AWS*. O'Reilly Media.

White, L., & Clark, S. (2023). *Advanced API design with AWS and Azure cloud platforms*. Apress.

Wilson, C. (2020). *Scalable API architectures in cloud environments: A comprehensive guide*. Addison-Wesley.

Wilson, G. (2021). *Serverless architecture for API development with AWS Lambda and Azure Functions*. Packt Publishing.

Xu, L., & Zhang, T. (2023). *Designing APIs for performance and scalability with Azure and AWS*. Packt Publishing.

Younis, M., & Iqbal, Z. (2021). *API development and cloud infrastructure on AWS and Azure*. Springer.

Zhang, L. (2022). *Mastering API development with Azure and AWS*. Wiley.

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

Desain dan Implementasi

API untuk Cloud Computing dengan Azure dan AWS

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