



MODUL PRAKTIKUM

Teknologi Cloud Computing

Program Studi Informatika
Jurusan Informatika
Fakultas Teknik Industri
UPN "Veteran" Yogyakarta



Modul Praktikum

Teknologi *Cloud Computing*

Tim Penyusun:

Sylvert Prian Tahalea, S.Si., M.Cs.

Dewi Zunuvi Setiawati

Rico Aminanda

Rifka Canalisa Rahayu

Shazi Awaludin

Vincentius Willy Ardiyanto

MODUL 1

CLOUD COMPUTING DAN SERVICE PROVIDER

A. Tujuan Praktikum

- Mahasiswa mampu memahami konsep cloud computing
- Mahasiswa mampu memahami penggunaan cloud service provider
- Memahami konsep manajemen resource akses di Google Cloud Platform.
- Mampu menerapkan penggunaan Identity & Access Management

B. Alokasi Waktu

2 x 60 menit

C. Dasar Teori

1. Cloud Computing

Menurut NIST, cloud computing adalah model untuk memungkinkan untuk diakses dari manapun, nyaman, sesuai permintaan akses jaringan bersama dengan sumber daya komputasi yang telah dikonfigurasi (misalnya, jaringan, server, penyimpanan, aplikasi, dan layanan) yang dapat dengan cepat ditetapkan dan dirilis dengan usaha manajemen minimal dari penyedia layanan. Secara singkatnya, Cloud computing (komputasi awan) merupakan teknologi komputasi dihubungkan kepada jaringan yang berbasis internet sehingga memungkinkan untuk menjalankan beberapa aplikasi dan program pada beberapa komputer dalam waktu bersamaan. Cloud computing juga termasuk dalam teknologi abstraksi infrastruktur yang disembunyikan sehingga pengguna dapat memanfaatkannya tanpa mengetahui proses, infrastruktur, dan teknologi yang ada di dalamnya.

2. Karakteristik Cloud Computing

Sebagaimana yang telah dijelaskan diatas bahwa pengertian cloud computing merupakan model yang memanfaatkan jaringan internet untuk kemudahannya, namun tidak semua model yang memungkinkan untuk diakses dimanapun adalah cloud computing. Hal tersebut harus memenuhi karakteristik sebagai berikut.

a. *On-demand self service*

On-demand self service (layanan mandiri sesuai permintaan) merupakan layanan yang dimiliki oleh sistem dimana konsumen dapat memutuskan pilihannya sesuai kebutuhan antara lain dapat secara sepihak menyediakan kemampuan komputasi, seperti waktu server dan penyimpanan jaringan, sesuai kebutuhan secara otomatis tanpa memerlukan interaksi manusia dengan setiap penyedia layanan.

b. *Broad network access*

Broad network access merupakan kemampuan sistem yang dapat tersedia pada jaringan dan diakses melalui mekanisme standar pada platform kecil (misalnya, ponsel, tablet, laptop, dan workstation).

c. *Resource pooling*

Sumber daya komputasi penyedia layanan dikumpulkan untuk melayani beberapa konsumen menggunakan model multi-penyewa, dengan sumber daya fisik dan virtual yang berbeda secara dinamis ditugaskan dan akan ditugaskan kembali sesuai dengan permintaan konsumen. Hal ini memberikan keringanan untuk pelanggan dimana umumnya pelanggan tidak memiliki kontrol atau pengetahuan atas tepat lokasi sumber daya yang disediakan tetapi mungkin dapat menentukan lokasi pada tingkat yang lebih tinggi abstraksi (misalnya, negara, provinsi, atau Data Center). Contoh sumber daya termasuk penyimpanan, pemrosesan, memori, dan bandwidth jaringan.

d. *Rapid elasticity*

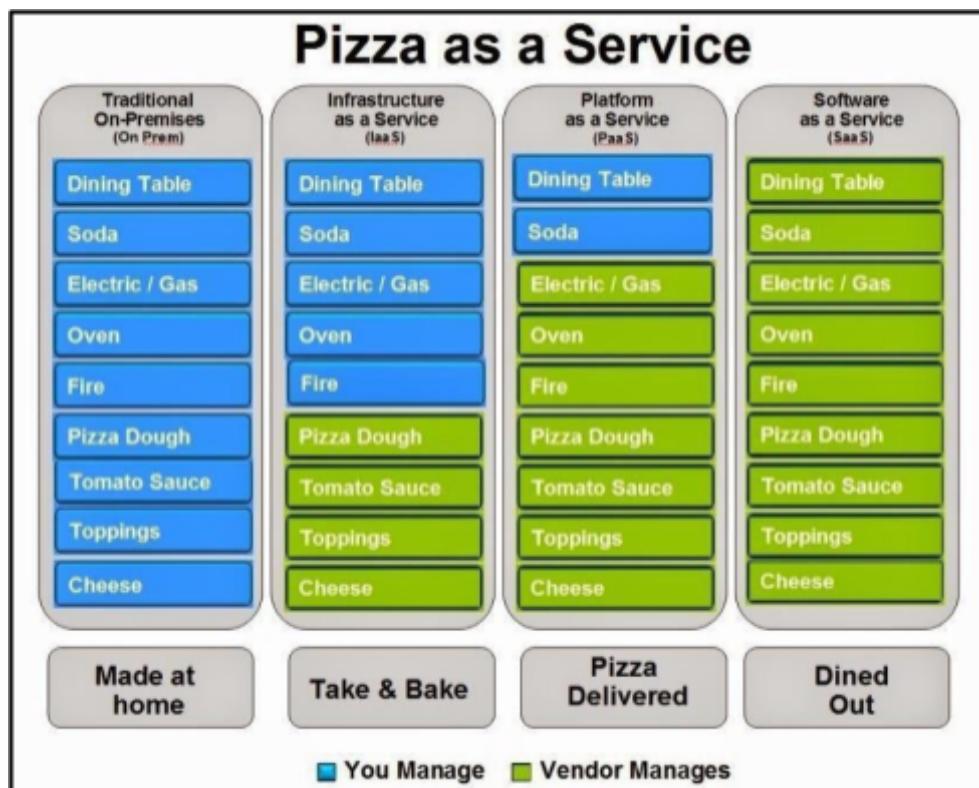
Sistem bersifat elastis atau otomatis pada saat ditetapkan dan dirilis, sehingga dengan cepat dapat memproses permintaan masuk maupun keluar. Untuk konsumen, kemampuan yang disediakan sering tidak terbatas dan dapat disesuaikan jumlahnya setiap saat.

e. *Measured service*

Sistem cloud secara otomatis mengontrol dan mengoptimalkan penggunaan sumber daya dengan memanfaatkan kemampuan metering pada beberapa tingkat abstraksi yang sesuai dengan jenis layanan (misalnya, Penyimpanan, pemrosesan, bandwidth, dan akun pengguna aktif). Penggunaan sumber daya dapat dipantau, dikontrol, dan dilaporkan, memberikan transparansi bagi penyedia dan konsumen dari layanan yang dimanfaatkan.

3. Model Layanan

Adapun jenis layanan yang tersedia pada cloud computing menurut NIST adalah sebagai berikut.



Sumber: Dunelm Technology

- **Infrastructure as a Service (IaaS)**

Layanan yang diberikan kepada konsumen adalah untuk penyediaan, pengelolaan, penyimpanan, jaringan, dan sumber daya komputasi lainnya. Konsumen dapat menyebarkan dan menjalankan perangkat lunak secara bebas termasuk sistem operasi dan aplikasi. Konsumen tidak mengelola atau mengontrol infrastruktur cloud yang mendasar tetapi memiliki kontrol atas sistem operasi, penyimpanan, dan aplikasi yang dikerahkan.

Keuntungan dari IaaS adalah konsumen tidak perlu membeli komputer fisik, dan konfigurasi komputer virtual dapat diubah dengan mudah. Jika komputer sudah kelebihan beban, bisa dilakukan upgrade CPU, RAM, maupun storage. Target pengguna dari IaaS adalah *network architect* dan *IT administrator*. Contoh penyedia layanan IaaS: Amazon EC2 dan Windows Azure.

- **Platform as a Service (PaaS)**

Layanan yang diberikan kepada konsumen adalah pemberian akses ke infrastruktur cloud yang dibuat sehingga aplikasi yang diperoleh atau yang dibuat konsumen bisa berjalan. Konsumen tidak mengelola atau mengendalikan infrastruktur cloud seperti jaringan, server, sistem operasi, atau penyimpanan, tetapi memiliki kontrol atas aplikasi yang dijalankan dan mungkin pengaturan konfigurasi untuk hosting.

Keuntungan dari PaaS adalah konsumen tidak perlu memikirkan tentang kebutuhan maupun pemeliharaan lingkungan (OS, jaringan, database engine, framework, dll.). Hal tersebut merupakan tanggung jawab dari penyedia layanan. Target pengguna PaaS adalah *software developer*. Contoh PaaS: Amazon Web Service (AWS), Windows Azure, dan Google App Engine.

- **Software as a Service (SaaS)**

Layanan yang diberikan kepada konsumen adalah layanan yang bisa diakses menggunakan aplikasi penyedia layanan pada infrastruktur cloud. Aplikasi ini dapat diakses dari berbagai perangkat klien baik melalui antarmuka klien tipis, seperti browser web (misalnya, email berbasis web), atau antarmuka program. Konsumen tidak mengelola atau mengontrol infrastruktur cloud seperti jaringan, server, sistem operasi, penyimpanan, atau bahkan kemampuan aplikasi secara spesifik.

Salah satu keuntungan dari SaaS adalah pengguna tidak perlu membeli lisensi baru setiap ada peningkatan software ke versi terbaru. Pengguna dimudahkan untuk bisa berlangganan dan membayar sesuai pemakaian kepada penyedia layanan SaaS. Pengguna SaaS adalah *End User*. Contoh produk SaaS: Office 365, Gmail, Adobe CC, dan Skype.

4. Model Penyebaran

Selain memiliki 5 karakteristik utama dan 3 model layanan, cloud computing juga memiliki 4 model penyebaran, yaitu *private cloud*, *community cloud*, *public cloud*, dan *hybrid cloud*.

1) **Private Cloud**

Infrastruktur cloud ditetapkan untuk penggunaan eksklusif oleh satu organisasi yang terdiri dari beberapa konsumen (misalnya unit bisnis). Ini mungkin dimiliki, dikelola, dan dioperasikan oleh organisasi, dan/atau pihak ketiga.

2) **Community Cloud**

Infrastruktur cloud disediakan untuk penggunaan eksklusif oleh komunitas konsumen tertentu dari organisasi yang memiliki kepentingan bersama (misalnya, misi, persyaratan keamanan, kebijakan, dan pertimbangan). Ini mungkin dimiliki, dikelola, dan dioperasikan oleh satu atau lebih organisasi dalam komunitas, dan/atau pihak ketiga.

3) **Public Cloud**

Infrastruktur cloud disediakan untuk penggunaan terbuka oleh publik. Ini mungkin dimiliki, dikelola, dan dioperasikan oleh bisnis, akademik, dan/atau organisasi pemerintah.

4) **Hybrid Cloud**

Infrastruktur cloud yang terdiri dari dua atau lebih infrastruktur cloud yang berbeda (pribadi, komunitas, atau publik) yang tetap menjadi entitas unik, namun terikat bersama oleh teknologi standar atau proprietary yang memungkinkan portabilitas data dan aplikasi.

5. **Resiko Cloud Computing**

Selain memiliki keuntungan dengan layanan dan model yang diberikan, komputasi cloud juga memiliki resiko seperti waktu respons, proteksi data, maupun kepemilikan data. Adapun resiko cloud computing dapat dibagi menjadi beberapa, antara lain.

1) **Service Level**

Provider mungkin tidak konsisten dengan performa layanan yang mengakibatkan jeleknya transaksi. Hal ini mengharuskan pelanggan untuk memahami secara baik tentang layanan apa saja yang diberikan oleh provider.

2) **Privacy**

Hosting yang digunakan bersifat sementara dan dimiliki secara fisik oleh penyedia layanan sehingga terdapat kemungkinan adanya kebocoran data atau adanya aliran data masuk maupun keluar tanpa sepengetahuan pelanggan.

3) **Compliance**

Pelanggan sebaiknya mengetahui level layanan yang disediakan oleh provider. Hal ini berguna untuk pemenuhan layanan yang seharusnya diterima oleh pelanggan dan untuk menghindari kerugian yang disebabkan oleh kecurangan yang dilakukan penyedia layanan.

4) Kepemilikan Data (*Data Ownership*)

Pelanggan harus paham dengan aturan main atau legal kontrak terkait kepemilikan data ketika menggunakan layanan cloud. Ketika data di upload di cloud, terdapat resiko data hilang oleh karenanya diperlukan adanya backup. Selain itu, pelanggan juga harus memastikan memiliki hak kepemilikan data sehingga tidak terjadi hal yang tidak diinginkan nantinya.

5) Mobilitas Data

Mobilitas data pelanggan merupakan hal yang cukup krusial. Misalnya kemungkinan pengambilan maupun pemindahan data ketika terjadi pemutusan kontrak atau selesainya masa kontrak.

6. Cloud Service Provider

a. Definisi

Cloud service provider merupakan pihak ketiga penyedia layanan cloud. Provider akan menawarkan akses sumber daya IT yang mudah kepada klien tanpa memerlukan infrastruktur maupun perangkat keras internal. Selain dari pertimbangan aspek biaya yang murah, layanan Cloud Computing dapat menunjang efektivitas kinerja dari perusahaan. Hal inilah yang mendorong semakin beragamnya cloud service provider.

b. Jenis Cloud Service Provider

Terdapat banyak perusahaan penyedia layanan cloud. Pada modul ini, akan dibahas beberapa dari produk layanan cloud sebagai berikut.

1) Google Cloud Platform

Google Cloud Platform merupakan *platform* berisi kumpulan layanan cloud yang dirilis pada awal tahun 2020 oleh Google. Disadur dari *website* resminya, Google Cloud Platform menawarkan banyak layanan cloud beberapa diantaranya yaitu:

1. Komputasi

- Compute Engine

Layanan compute engine pada Google Cloud Platform merupakan jenis layanan IaaS. Layanan ini memungkinkan pengguna untuk membuat dan menjalankan *virtual machine* di platform Google, dengan opsi untuk menggunakan CPU, GPU, atau Cloud TPU

tertentu. Compute engine juga membuat pengguna bisa melakukan migrasi antar host tanpa melakukan boot ulang.

- App Engine

Layanan ini memungkinkan pengguna untuk membuat dan menjadi host aplikasi pada sistem yang sama yang menggerakkan aplikasi Google.

- Google Cloud VMware Engine (GCVE)

Suatu VMware berbentuk layanan yang secara khusus dirancang untuk menjalankan beban kerja VMware pada Google Cloud Platform.

GCVE memungkinkan pelanggan untuk menjalankan mesin virtual VMware sebagaimana aslinya dalam pusat data berbasis perangkat lunak yang didedikasikan, tersendiri.

2. Penyimpanan

- Cloud Storage

Suatu layanan RESTful untuk penyimpanan dan pengaksesan data Anda pada infrastruktur Google. Layanan menggabungkan kinerja dan keterukuran Cloud dari Google dengan keamanan tingkat tinggi dan kemampuan berbagi.

3. Identitas dan Akses

- Identity & Access Management (IAM)

Layanan ini memberi administrator kemampuan untuk mengelola sumber daya cloud secara terpusat dengan mengendalikan siapa saja yang dapat mengambil tindakan apa pada sumber daya tertentu.

4. Komputasi tanpa Server

- Cloud Run

Cloud Run (yang dikelola sepenuhnya) memungkinkan pengguna untuk menjalankan stateless container di perlengkapan yang dikelola sepenuhnya.

5. Developer Tools

- Cloud Build

Layanan yang menjalankan build pengguna di infrastruktur Google Cloud Platform. Cloud Build dapat mengimpor kode sumber dari Cloud Storage, Cloud Source Repositories, GitHub, atau Bitbucket; menjalankan build sesuai spesifikasi Anda; dan menghasilkan artefak seperti container Docker atau arsip Java.

2) Amazon Web Services (AWS)

Penyedia layanan cloud yang didirikan oleh perusahaan Amazon. AWS merupakan perusahaan yang mengenalkan metode layanan *pay-as-you-go*. Beberapa layanan Amazon Web Services sebagai berikut.

1. Cloud Computing

- EC2 (Elastic Compute Cloud)

Mesin virtual dimana pengguna mendapatkan OS level control yang memungkinkan mereka untuk menjalankannya sesuai keinginan.

- Elastic Beanstalk

Layanan ini menawarkan penyebaran otomatis dan penyediaan sumber daya seperti situs web produksi yang sangat *scalable*.

2. Migration

- DMS (Database Migration Service)

Layanan untuk memindahkan on-site database ke AWS. Jenis layanan ini memungkinkan pengguna untuk bermigrasi dari satu jenis database ke yang lainnya.

- Snowball

Aplikasi yang memungkinkan pengguna mentransfer beberapa tera-byte data di dalam dan di luar lingkungan Amazon Web Services.

3) Azure

Platform yang menawarkan berbagai produk dan layanan cloud. Azure didirikan oleh perusahaan Microsoft. Beberapa layanannya adalah sebagai berikut.

1. Komputasi

- Virtual Machine

Layanan yang menawarkan mesin virtual berbasis windows dan linux.

2. Kontainer

- Azure Kubernetes Service (AKS)

Layanan AKS membuat pengguna dapat mengembangkan dan menyebarkan aplikasi cloud-native di Azure, pusat data, atau di tepi dengan pagar pembatas dan alur kode-ke-cloud bawaan.

3. Infrastruktur desktop virtual

- Azure Virtual Desktop

Azure Virtual Desktop menawarkan kemudahan untuk pengguna untuk merasakan pengalaman desktop jarak jauh dengan keamanan yang terjamin.

7. Resources dan Management Access

Google Cloud menyediakan berbagai jenis resources yang bisa bebas dipilih. Misalnya seperti organizations, folder, dan juga project. Kamu bisa mengelompokkan dan mengatur keseluruhan resources tersebut berdasarkan hierarki Google Cloud lain. Organisasi hierarki semacam ini bisa membuat kamu menjadi lebih mudah dalam mengelola seluruh resources yang kamu miliki.

Kemudian, dalam setiap resources, kamu bisa menambahkan beberapa peran terhadap member dari resource tersebut. Dalam hal ini, kamu akan menggunakan yang namanya Cloud IAM atau Cloud Identity and Access Management. Dengan menggunakan IAM ini, kamu bisa membeda-bedakan hak akses dari satu pengguna dengan pengguna yang lain. Sehingga resources yang kamu miliki bisa menjadi lebih aman.

a. IAM

Identity and Access Management (IAM) memungkinkan administrator mengotorisasi siapa yang dapat mengambil tindakan pada resource tertentu, memberi Anda kontrol dan visibilitas penuh untuk mengelola resource Google Cloud secara terpusat. Untuk perusahaan dengan struktur organisasi yang kompleks, ratusan grup kerja, dan banyak proyek, IAM memberikan pandangan terpadu tentang kebijakan keamanan di seluruh organisasi Anda, dengan audit bawaan untuk memudahkan proses kepatuhan.

b. Roles

Roles merupakan peran yang akan diberikan oleh Owner kepada pengguna yang akan ditambahkan. Ada bermacam-macam jenis roles yang bisa diberikan. Bahkan, Owner bisa memberikan roles secara custom untuk mengatur hak akses suatu pengguna yang akan ditambahkan terhadap resource yang ada di dalam project.

Namun, secara sederhana, ada 4 basic roles yang harus diketahui. Yakni Owner, Editor, Viewer, dan juga Browser. Penjelasan lebih lanjut mengenai masing-masing roles, akan dibahas di langkah-langkah praktikum.

c. Resources

Resources merupakan sumber daya yang tersimpan di dalam sebuah project. Di GCP sendiri, terdapat banyak jenis sumber daya yang bisa ditambahkan. Contohnya yaitu bucket yang bisa ditambahkan untuk menjadi cloud storage, dan ada juga Virtual Machine Instance atau VM Instance yang merupakan bagian dari Compute Engine.

Setiap resource memiliki biaya masing-masing tiap kali penggunaanya. Besar kecilnya biaya ini berbeda-beda tergantung bagaimana pemakaian dan juga konfigurasi saat pembuatan resource baru.

D. Langkah Praktikum

Pembuatan Account Google Cloud Platform

Langkah-langkah pendaftaran akun Google Cloud Platform adalah sebagai berikut.

1. Pilih negara tempat anda berasal dan kebutuhan yang diinginkan.

Step 1 of 2 Account Information

DEWI ZUNUVI SETIAWATI [SWITCH ACCOUNT](#)

Good news! You're eligible for an additional \$100.00 in Free Trial credits for a total of \$400.00. You'll receive these credits within 24 hours of completing signup.

Country: Indonesia

What best describes your organization or needs?
Please select: Other

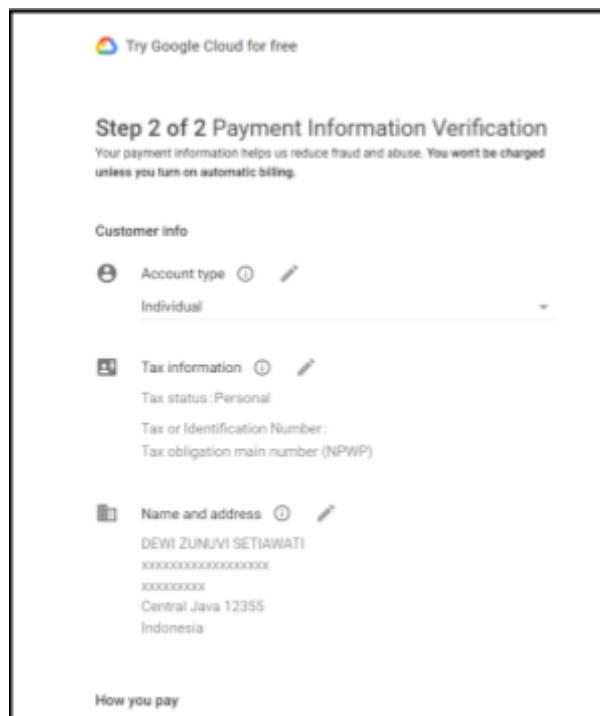
Terms of Service
 I have read and agree to the [Google Cloud Platform Terms of Service](#), [Supplemental Free Trial Terms of Service](#), and the terms of service of [any applicable services and APIs](#).

Required to continue

Email updates
 I would like to receive periodic emails on news, product updates and special offers from Google Cloud and Google Cloud Partners.

[CONTINUE](#)

2. Isi semua formulir termasuk bagian pembayaran.



Try Google Cloud for free

Step 2 of 2 Payment Information Verification

Your payment information helps us reduce fraud and abuse. You won't be charged unless you turn on automatic billing.

Customer info

Account type: Individual

Tax information

Tax status: Personal

Tax or Identification Number: DEWI ZUNUVI SETIAWATI

Name and address

DEWI ZUNUVI SETIAWATI

XXXXXXXXXXXXXX

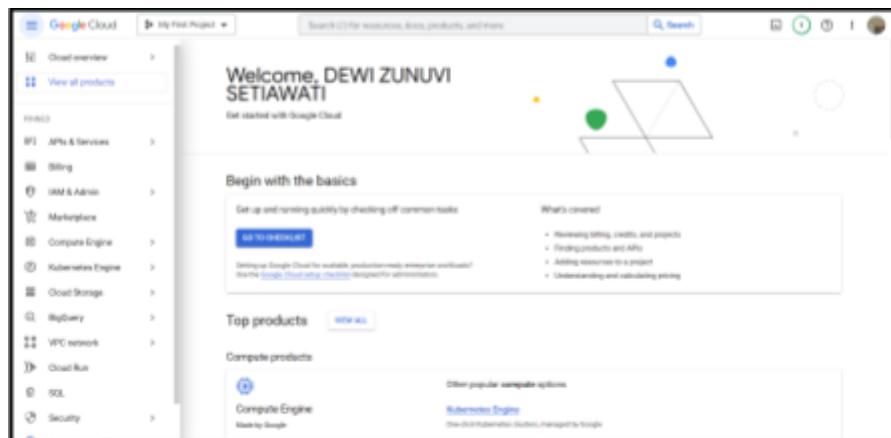
XXXXXXXXXX

Central Java 12355

Indonesia

How you pay

3. Pendaftaran akun selesai ketika muncul tampilan sebagai berikut



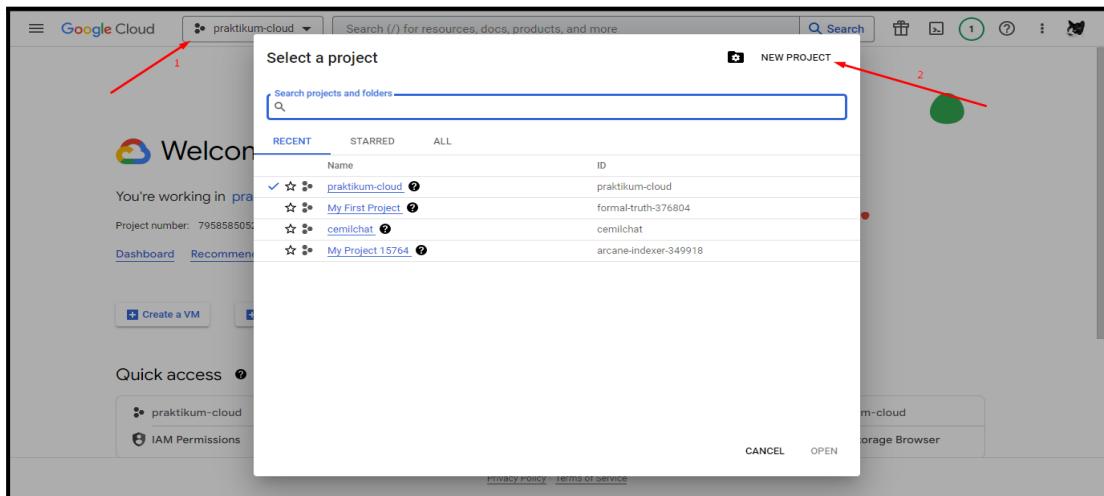
Resource dan Management Access

Setelah memahami dasar teori dari praktikum kali ini, selanjutnya praktikan harus melakukan beberapa langkah praktikum untuk memperdalam pemahaman terkait IAM di Google Cloud Platform ini.

Secara garis besar, terdapat dua hal yang harus dilakukan. Yang pertama adalah membuat project baru dan menambahkan akun lain ke dalam project tersebut. Kemudian yang kedua adalah menambahkan resource baru dan memastikan pengguna lain bisa mengakses resource tersebut berdasarkan role yang diberikan.

Berikut hal pertama yang harus praktikan lakukan:

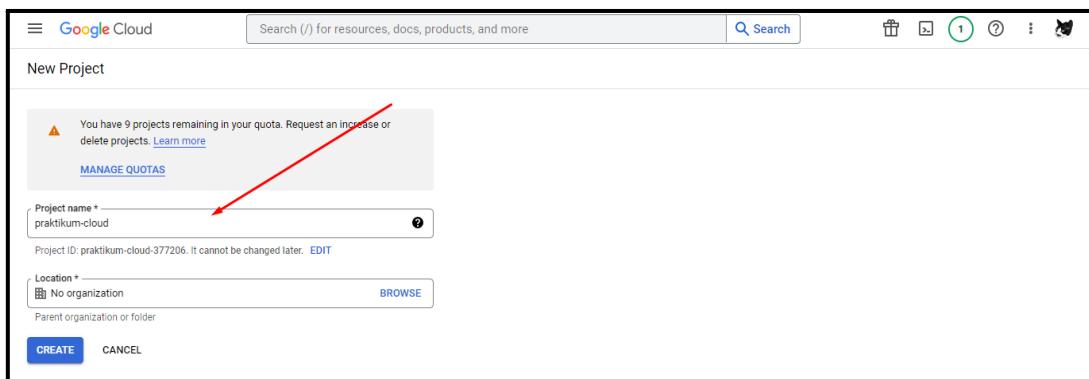
1. Membuat Project Baru dan Menambahkan Akun Lain ke Project



Gambar 1.1

Hal pertama yang harus praktikan lakukan adalah membuat project terlebih dahulu. Untuk itu, silahkan masuk ke <http://cloud.google.com/welcome> kemudian ikuti seperti pada gambar di atas.

Lakukan secara berurut mulai dari langkah satu dan juga langkah dua.

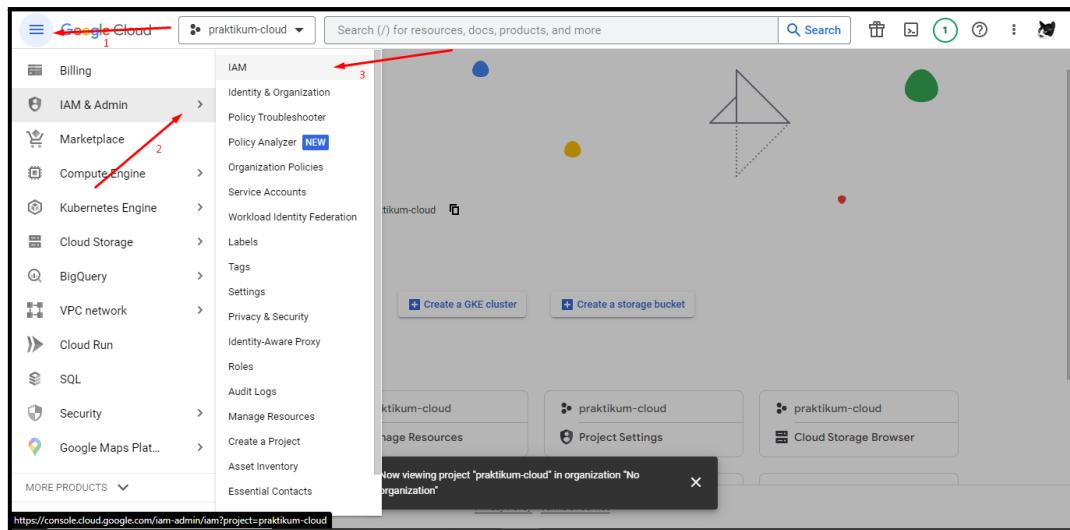


Gambar 1.2

Selanjutnya, praktikan harus memasukkan nama project seperti yang ditunjuk oleh anak panah pada gambar di atas. Praktikan bisa memasukkan nama project dengan **nim** atau bisa juga dengan nama **praktikum-cloud** seperti pada contoh di gambar.

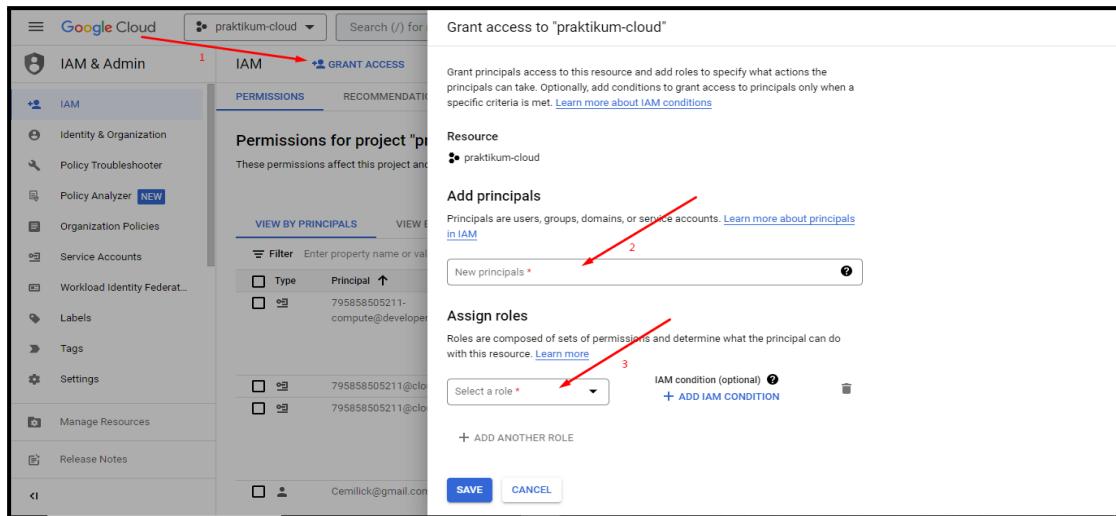
Selanjutnya, praktikan akan langsung diarahkan ke halaman welcome untuk project yang baru saja dibuat. Nah, sampai disini, praktikan sudah selesai dan berhasil membuat project baru.

Lalu, apa yang harus dilakukan praktikan selanjutnya? Selanjutnya praktikan harus menambahkan pengguna baru yang bisa mengakses project tersebut. Nah, disinilah IAM akan diterapkan.



Gambar 1.3

Untuk masuk ke menu IAM sendiri, praktikan bisa mengikuti langkah seperti pada **Gambar 1.3.**



Gambar 1.4

Setelah berhasil masuk ke menu IAM, praktikan akan melihat tampilan permissions untuk project yang sudah dibuat sebelumnya. Nah, disini, praktikan bisa melihat tombol Grant Access seperti halnya yang terdapat pada gambar di atas.

Tombol **GRANT ACCESS** berfungsi untuk memberikan akses kepada pengguna lain agar bisa masuk ke dalam project tersebut. Ada 4 role utama (basic) yang bisa kamu berikan kepada pengguna. Berikut diantaranya beserta penjelasannya:

- **Owner** : Memerlukan verifikasi melalui email yang diberikan akses terlebih dahulu. Bisa mengakses semua fitur di dalam project. Termasuk menambahkan akses ke pengguna baru.

- **Editor** : Bisa langsung ditambahkan tanpa verifikasi email. Memiliki akses untuk membuat, melihat, mengedit, dan menghapus resource. Hanya saja tidak bisa melakukan perubahan pada fitur IAM.
- **Viewer** : Hanya bisa melihat resource yang ada tanpa bisa melakukan perubahan terhadap resource yang tersedia.
- **Browser** : Hanya bisa melihat hierarki sebuah project saja. Jadi hanya bisa melihat di sebuah project terdapat resource apa saja. Namun tidak bisa mengakses masuk ke dalam resource yang ada di dalam sebuah project.

Itulah role yang bisa praktikan berikan kepada pengguna baru. Kemudian berdasarkan **Gambar 1.4**, praktikan harus memasukkan data di dalam dua kolom. Kolom pertama untuk **email** dari pengguna baru yang akan ditambahkan. Sedangkan data satunya yaitu untuk **role** dari pengguna baru tersebut.

Untuk 4 role basic, praktikan bisa menemukannya dengan menekan option pada **kolom form role** kemudian pilih **basic** lalu pilih **satu dari 4 role** yang tersedia.

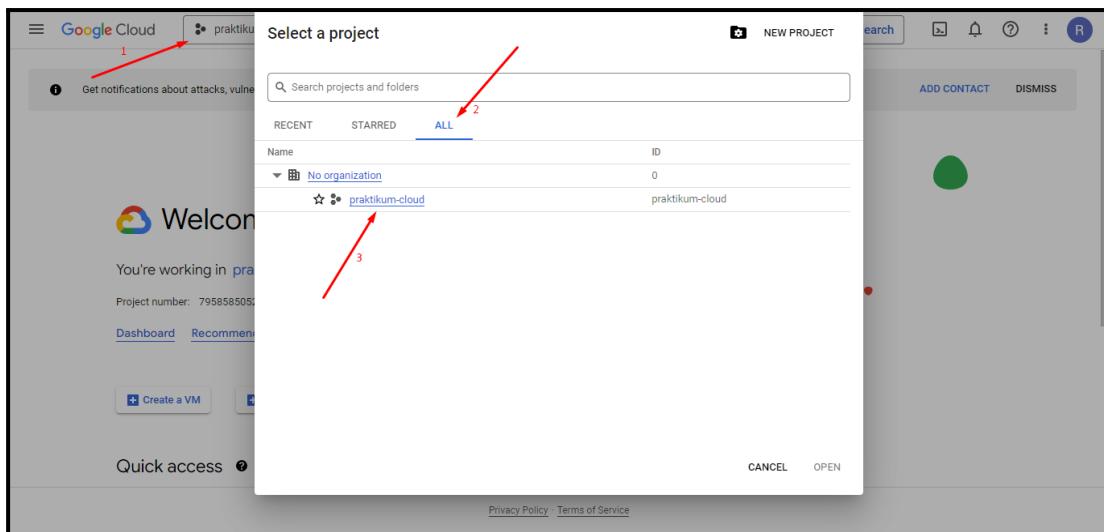
Selanjutnya, praktikan bisa langsung menekan tombol **save** untuk menyimpan pengaturan dan pemberian akses kepada pengguna baru yang sudah dituliskan.

Service Account	Role	Actions
Cloud Build Service Account	Editor	
Google APIs Service Agent	Editor	
Rico Aminanda	Owner	
App Engine default service account	Editor	
ricoaminanda@gmail.com	Editor	
ricoaminanda@gmail.com	Owner	
ricoaminanda@gmail.com	Owner	
ricoaminanda@gmail.com	Owner	

Gambar 1.5

Setelah menekan tombol **save**, pastikan e-mail dari pengguna yang sudah ditambahkan, sudah berada di dalam daftar permissions pada IAM. Seperti contoh email yang tertera pada gambar di atas yang berada di dalam kotak.

Sampai sini, praktikan sudah berhasil menambahkan pengguna baru yang bisa mengakses project tersebut. Langkah selanjutnya adalah memastikan apakah pengguna baru tersebut sudah bisa mengakses project melalui akun sesuai dengan email yang ditambahkan atau belum.



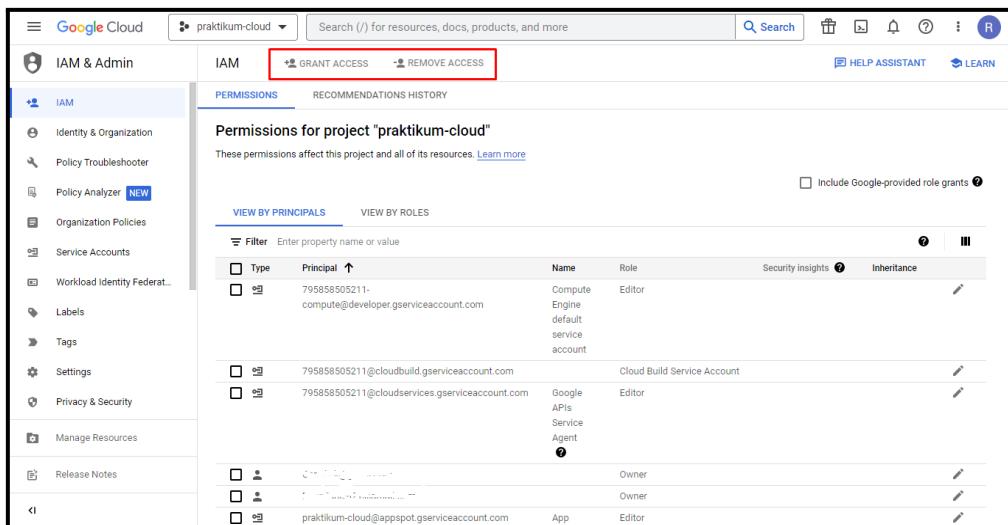
Gambar 1.6

Caranya adalah dengan mengakses <http://cloud.google.com/welcome> dengan menggunakan akun google lain sesuai dengan yang sudah ditambahkan pada IAM sebelumnya. Jika sudah masuk ke akun sesuai dengan yang ditambahkan, praktikan bisa langsung mengikuti arahan sesuai dengan gambar di atas.

Pastikan sudah tertera nama project sesuai dengan yang dibuat sebelumnya. Apabila di tab recent tidak terlihat project nya, praktikan bisa berpindah ke tab **All** untuk melihat semua project yang bisa diakses.

Dalam hal ini, biasanya memerlukan waktu untuk project tersebut bisa muncul di akun yang baru saja ditambahkan. Untuk itu, tidak perlu panik apabila project tidak muncul. Tunggu saja sekitar 1 - 3 menit kemudian refresh halaman tersebut dan project akan muncul dengan sendirinya.

Kemudian pengguna baru tinggal memilih project tersebut untuk mencoba menjelajahi keseluruhan resource yang ada di project.



Gambar 1.7

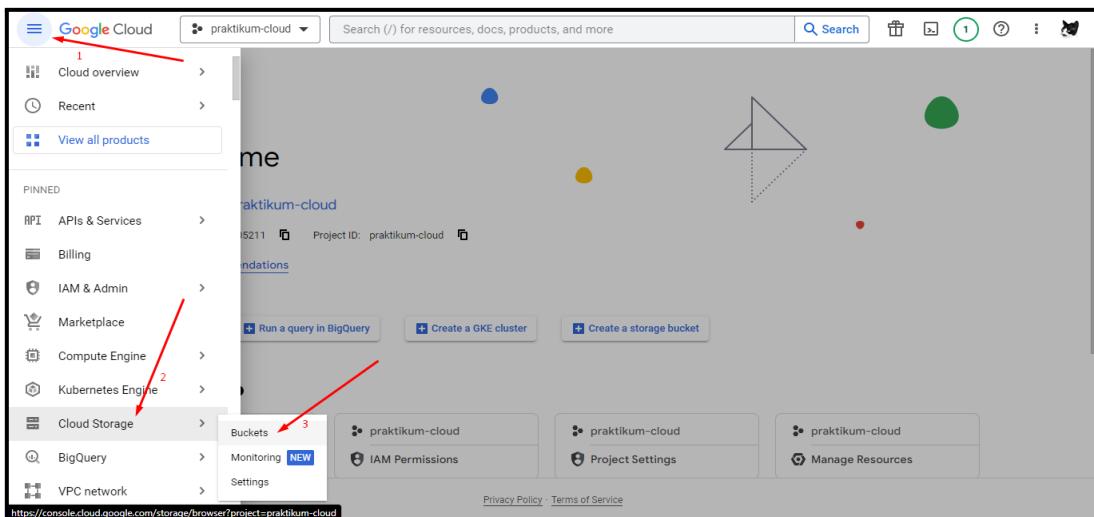
Nah, apabila pengguna yang ditambahkan memiliki role selain Owner, maka pengguna tersebut akan melihat tombol **GRANT ACCESS** dan **REMOVE ACCESS** di menu IAM berwarna gelap. Yang artinya, fitur untuk mengelola permissions sendiri tidak diijinkan untuk pengguna dengan role **selain Owner**.

Sampai sini, langkah pertama sudah selesai dikerjakan. Selanjutnya, praktikan akan belajar bagaimana cara menambahkan resource dan juga bagaimana cara menghapus permissions untuk akun yang sudah ditambahkan sebelumnya.

2. Membuat Resource Baru dan Menghapus Permissions untuk User

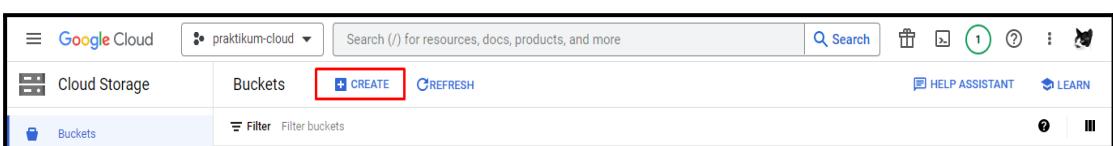
Selanjutnya, praktikan bisa mencoba membuat resource baru dan memastikan bahwa pengguna yang sudah ditambahkan sebelumnya, bisa mengakses resource tersebut.

Dalam praktikum kali ini, praktikan akan diminta untuk membuat sebuah resource berupa bucket. Perlu diperhatikan, untuk pembuatan resource ini, praktikan harus sudah mendaftarkan billing account terlebih dahulu. Setelah itu, praktikan baru bisa membuat sebuah resource dengan menggunakan billing account yang ada.



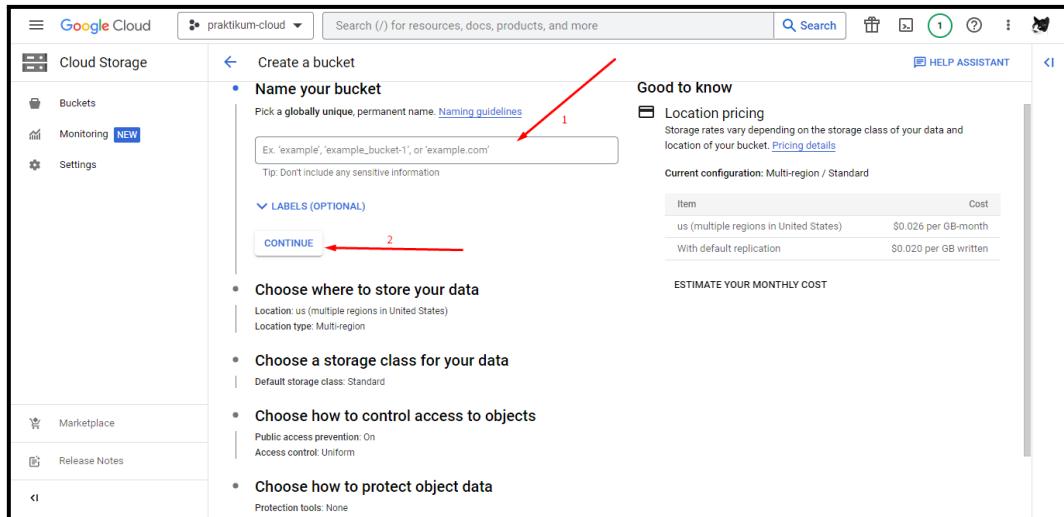
Gambar 2.1

Untuk mengakses menu bucket, praktikan bisa mengikuti langkah seperti pada gambar di atas.



Gambar 2.2

Selanjutnya praktikan bisa melihat terdapat tombol CREATE di bagian seperti pada gambar di atas. tombol tersebut bisa digunakan untuk membuat sebuah bucket baru. Untuk itu, silahkan klik tombol CREATE.

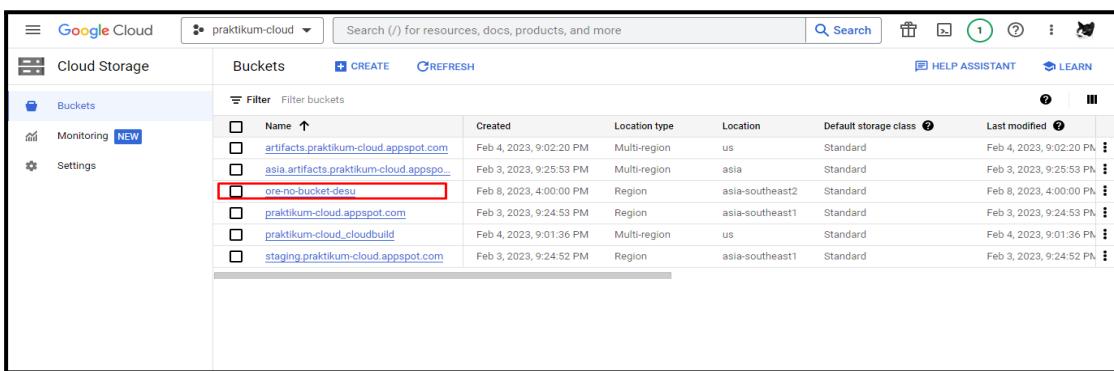


Gambar 2.3

Kemudian praktikan akan diminta untuk mengisi beberapa data. Diantaranya adalah sebagai berikut.

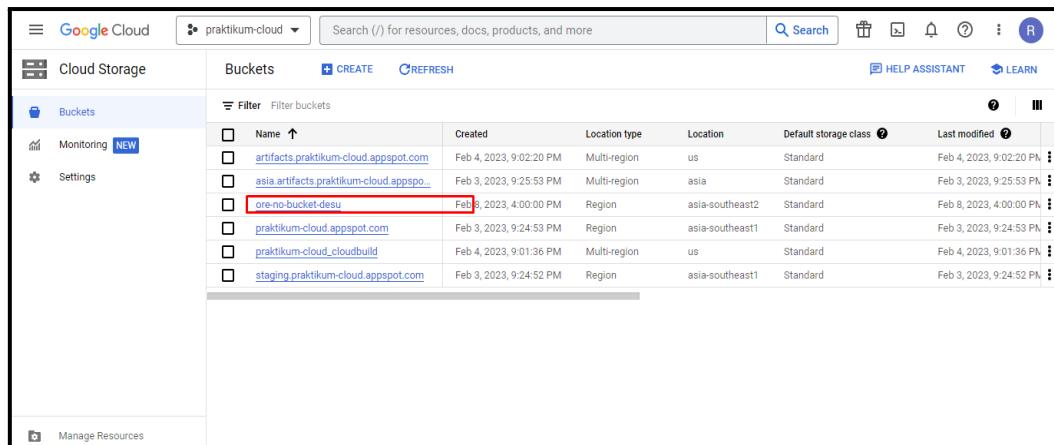
- **Nama Bucket** : Silahkan diisi bebas sesuai dengan yang diinginkan (ex: nim-bucket)
- **Region to Store Data** : Silahkan pilih **region** → **asia-southeast2 (Jakarta)**
- **Storage Class** : Silahkan pilih **Standard**

Kemudian untuk form lainnya, biarkan saja sesuai default nya. Setelah itu, praktikan bisa langsung menekan tombol Create di bagian bawah untuk membuat bucket baru.



Gambar 2.4

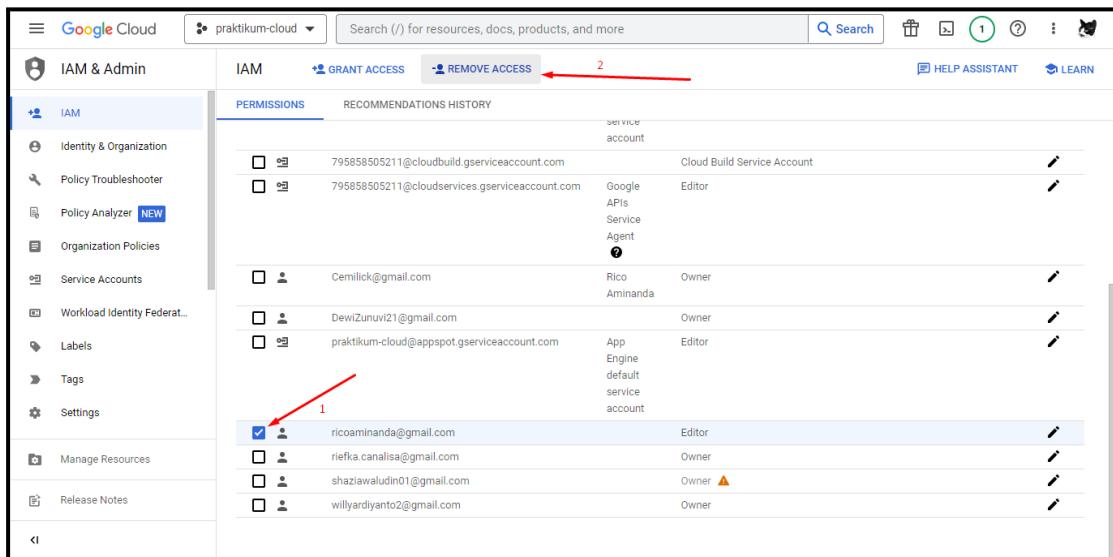
Setelah proses pembuatan selesai, selanjutnya praktikan bisa mengecek di bagian bucket. Pastikan sudah ada bucket yang ditambahkan sebelumnya di dalam daftar bucket yang ada di project milik praktikan.



Gambar 2.5

Selanjutnya, praktikan bisa mencoba membuka project dengan menggunakan akun lain yang sudah ditambahkan sebelumnya. Pastikan bucket yang sudah dibuat, juga sudah muncul di dalam daftar bucket di akun pengguna lain.

Sampai sini, praktikan sudah berhasil membuat resource baru yaitu sebuah bucket dan sudah berhasil mengakses resource tersebut dengan akun lain yang sudah ditambahkan ke dalam project. Selanjutnya, saatnya untuk mencoba bagaimana caranya menghapus akses dari pengguna lain ke dalam project sehingga akun yang sudah ditambahkan sebelumnya, kini sudah tidak bisa lagi untuk mengakses project beserta resource yang terdapat di dalamnya.

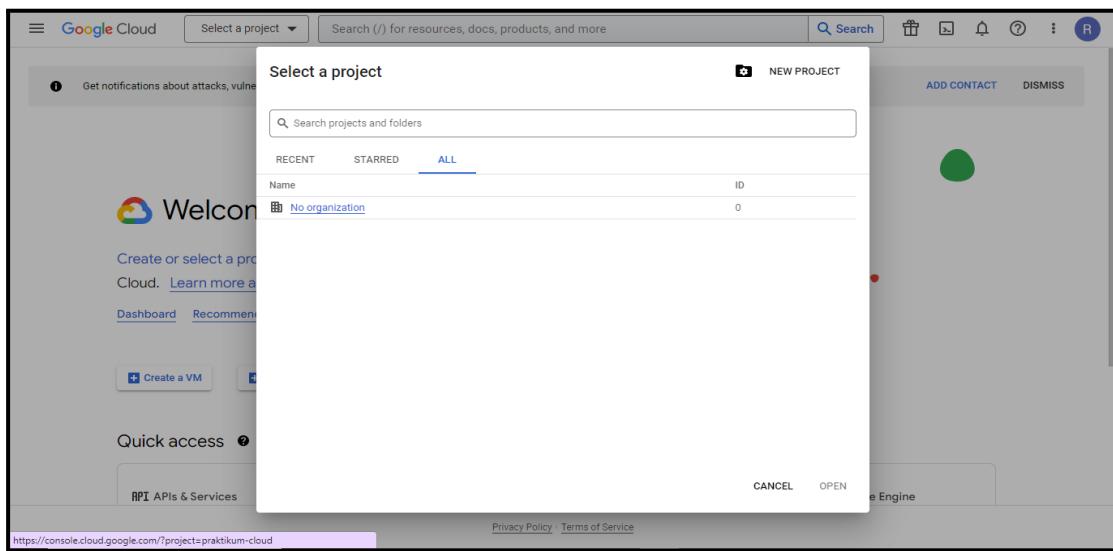


Gambar 2.6

Dalam hal ini, silahkan untuk mengakses project dengan menggunakan akun utama milik praktikan kembali (bukan menggunakan akun yang sudah ditambahkan). Kemudian, silahkan masuk ke menu IAM untuk melihat daftar pengguna yang bisa mengakses project.

Setelah itu, praktikan bisa mengikuti langkah seperti pada gambar di atas. Silahkan klik centang pada pengguna yang akan dihapus aksesnya. Kemudian tekan tombol **REMOVE ACCESS** di bagian atas.

Ketika praktikan sudah melakukan langkah ini, maka, akun pengguna yang sudah dihapus, tidak akan bisa lagi mengakses project tersebut termasuk resource di dalamnya.



Gambar 2.7

Sehingga, ketika pengguna lain mengakses <http://cloud.google.com/welcome> kembali kemudian pergi ke daftar project, maka pengguna tersebut tidak akan menemukan daftar project yang sudah dihapus aksesnya.

Sampai sini, praktikan sudah berhasil menghapus akses suatu pengguna ke sebuah project. Perlu diperhatikan kembali bahwa pengelolaan akses IAM ini hanya bisa dilakukan oleh pengguna dengan role **Owner** saja. Jadi, pastikan praktikan memberikan role yang tepat saat memberikan akses ke pengguna lain.

MODUL 2

MEMAHAMI *INFRASTRUCTURE AS A SERVICE* DENGAN GOOGLE COMPUTE ENGINE

Infrastructure as a Service (IaaS) adalah pengiriman perangkat keras (server, penyimpanan dan jaringan), dan perangkat lunak terkait (teknologi virtualisasi sistem operasi, sistem file), sebagai layanan atau dengan kata lain, ini adalah model layanan cloud. Ini adalah evolusi dari hosting tradisional yang tidak memerlukan komitmen jangka panjang dan memungkinkan pengguna untuk menyediakan sumber daya sesuai permintaan yang berarti organisasi tidak perlu membeli peralatan dan berinvestasi di ruang fisik, biasanya kamar khusus dengan daya dan pendinginan. Salah satu contohnya adalah Google Compute Engine. Dalam modul ini, Anda akan mempelajari cara kerja Google Compute Engine.

A. Tujuan Praktikum

- Menjelaskan tujuan dan kasus penggunaan untuk Google Compute Engine
- Membuat *Virtual Machine* di Google Compute Engine
- Akses *Virtual Machine* menggunakan IP Eksternal

B. Alokasi Waktu

2 x 60 menit

C. Dasar Teori

Google Compute Engine adalah salah satu layanan Google Cloud Platform yang memungkinkan Anda membuat dan menjalankan mesin virtual di infrastruktur Google. Anda dapat menyesuaikan *Virtual Machine* (VM) Anda tergantung pada kebutuhan Anda. Anda perlu tahu jenis mesin virtual mana yang Anda butuhkan. Ada beberapa contoh jenis *Virtual Machine* yang disediakan Google:

1. Scale out workloads (T2A, T2D)

VM Tau (T2A, T2D) menawarkan kombinasi performa yang hebat untuk beban kerja perluasan skala termasuk server web, layanan mikro dalam kontainer, transcoding media, dan aplikasi Java skala besar. Pilih dari VM berbasis x86 atau Arm untuk memenuhi beban kerja dan persyaratan bisnis.

2. General purpose workloads (E2, N2, N2D, N1)

E2, N2, N2D, dan N1 adalah mesin serbaguna yang menawarkan keseimbangan kinerja yang baik, dan cocok untuk berbagai beban kerja umum termasuk database, lingkungan pengembangan dan pengujian, aplikasi web, dan game seluler. Mereka mendukung hingga 224 vCPU dan memori 896 GB.

3. Ultra-high memory (M2, M1)

Mesin yang dioptimalkan memori menawarkan konfigurasi memori tertinggi hingga 12 TB untuk satu instans. Mereka sangat cocok untuk beban kerja intensif memori seperti database dalam memori besar seperti SAP Hana, dan beban kerja analitik data dalam memori.

4. Compute-intensive workloads (C3, C2, C2D)

Mesin yang dioptimalkan komputasi memberikan performa per inti tertinggi pada Compute Engine dan dioptimalkan untuk beban kerja seperti komputasi performa tinggi (HPC), server game, dan penyajian API yang sensitif terhadap latensi.

5. Most demanding applications and workloads (A2)

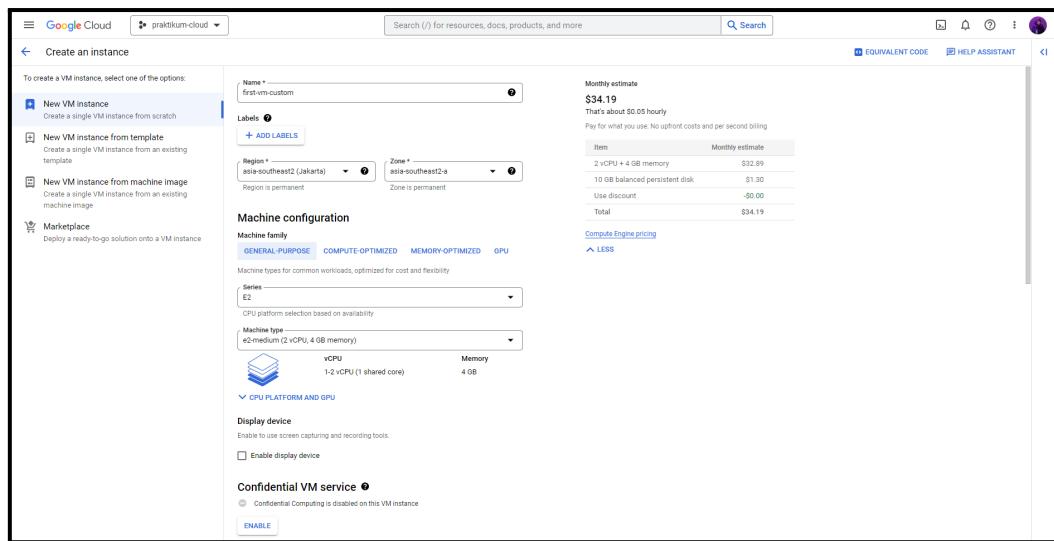
Mesin yang dioptimalkan akselerator didasarkan pada GPU NVIDIA Ampere A100 Tensor Core. Setiap GPU A100 menawarkan performa komputasi hingga 20x lipat dibandingkan dengan GPU generasi sebelumnya. VM ini dirancang untuk beban kerja Anda yang paling menuntut seperti pembelajaran mesin dan komputasi performa tinggi.

Anda dapat menyesuaikan VM Anda sendiri dengan memilih sistem Operasi yang Anda butuhkan, jenis mesin yang memenuhi kebutuhan Anda, bagaimana Anda mengatur pekerjaan batch atau beban kerja.

D. Langkah Praktikum

1. **Login** ke Akun GCP (Google Cloud Platform) Anda
2. **Aktifkan API Compute Engine**
3. **Pilih proyek** (Pilih dari tugas pertama)
4. **Buka Compute Engine lalu buka Instans VM**
5. **Pilih buat Instans VM**

Kemudian halaman instance vm baru akan terbuka seperti gambar berikut



- Anda dapat mengisi **name** sebagai **first-vm-custom**.
- Untuk praktikum ini, kita akan memilih **Region APAC** dan **Zone** antara **asia-southeast2-a, asia-southeast2-b, asia-southeast2-c**.

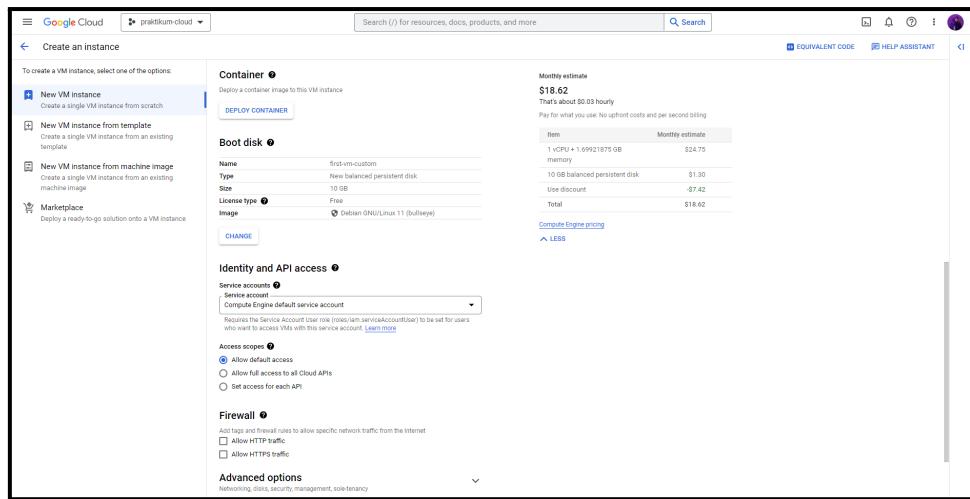
Ketika Anda ingin memilih di Wilayah dan Zona, Anda perlu mempertimbangkan tempat tersebut karena terkait dengan lokasi fisik server tempat vm akan disebarluaskan secara permanen.

Umumnya, lebih baik memilih yang lebih dekat ke kantor. Wilayah APAC adalah untuk wilayah Asia Pasifik yang meliputi Asia Timur, Asia Selatan, Asia Tenggara, dan Australia. Indonesia atau khususnya Jakarta termasuk dan memiliki kode zona sebagai **asia-tenggara2-a, asia-tenggara2-b, asia-tenggara2-c**.

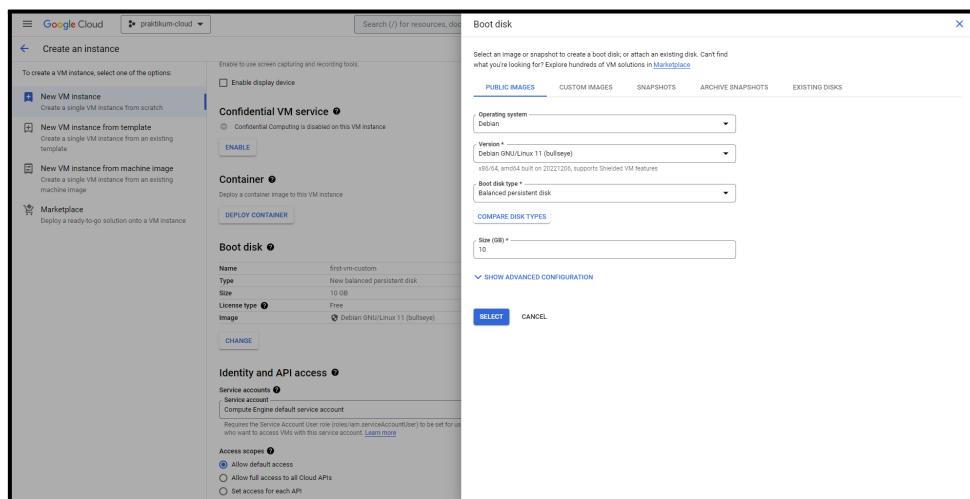
- Untuk keperluan praktikum ini, Anda dapat memilih **Machine Family** sebagai **General purpose** dan memilih **Series N1**. Sedangkan untuk **Machine type**, pilih **g1-small**.

Setiap zona juga memiliki jenis mesin masing-masing yang disediakan. Indonesia kebanyakan hanya menyediakan jenis mesin **E2, N2, N1, dan M1**.

- Berikutnya jika Anda menggulir ke bawah, gambar berikut akan ditampilkan



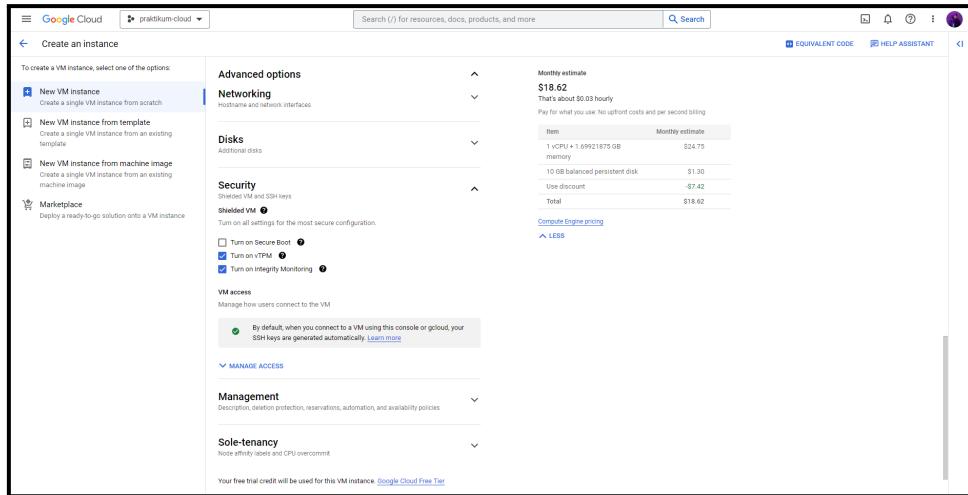
- e. Selanjutnya akan dipilih **disk boot** atau **sistem operasi**
 Gambar-gambar berikut menunjukkan.



- f. Karena ada banyak pilihan, untuk praktikum ini, kita akan menggunakan **Debian OS dengan Versi Linux 11 (mendukung Shielded VM)**. Adapun hal-hal lain, tidak perlu mengubahnya, cukup gunakan default.
- g. Selanjutnya, atur **Access scopes** ke **Allow full access to all Cloud APIs**
 Cakupan akses berarti Anda dapat menentukan akses ke VM ke individu tertentu atau terbuka untuk seluruh organisasi. Kami akan berasumsi karena kami masih mengembangkan aplikasi kami yang berarti kami membutuhkan lebih banyak akses API, itulah sebabnya kami memilih Izinkan akses penuh ke semua API Cloud.

- h. Selanjutnya, kami mengatur pengaturan **Firewall** ke **Allow HTTPS Traffic**. Pengaturan firewall akan menentukan apakah VM Anda akan dapat diakses melalui protokol internet HTTP dan HTTPS.

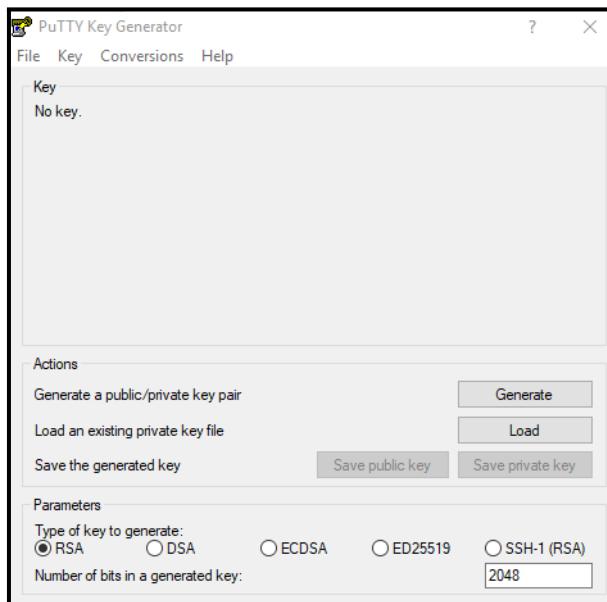
- i. Klik **Management, security, disks, networking, sole tenancy**
j. Buka bagian **Security** dan gambar berikut akan ditampilkan



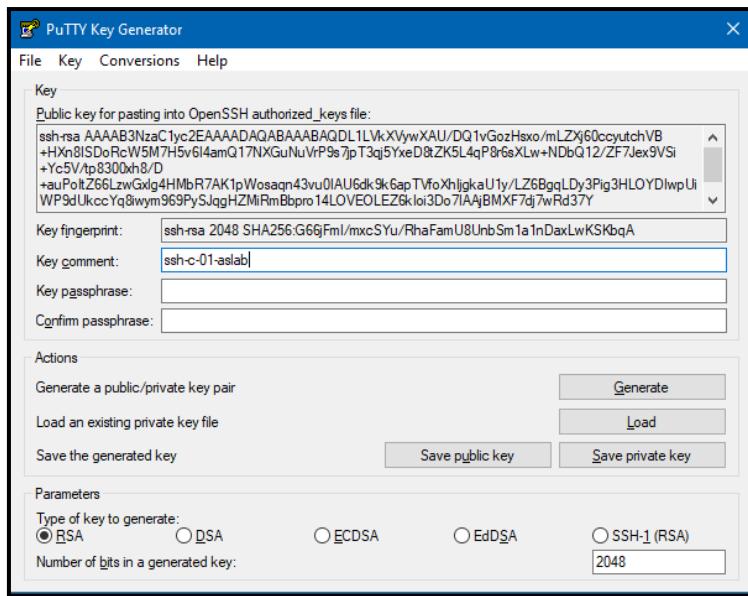
- k. Buka **Manage Access** dan kami akan **manually add an SSH key** untuk VM.

Bagi Pengguna **Windows** untuk menghasilkan kunci SSH:

- Anda perlu **mengunduh Putty** dari <https://www.putty.org>
- Setelah menginstal Putty, kita perlu membuka **PuttyGen**. Gambar berikut akan ditampilkan



- Kami memilih **type of key RSA** untuk dihasilkan dan selanjutnya klik **Generate**.
- Setelah mengikuti instruksi, kunci akan dihasilkan seperti gambar berikut



- Kita dapat mengubah **Key comment** dan **Key passphrase** untuk memastikan kunci aman untuk digunakan.

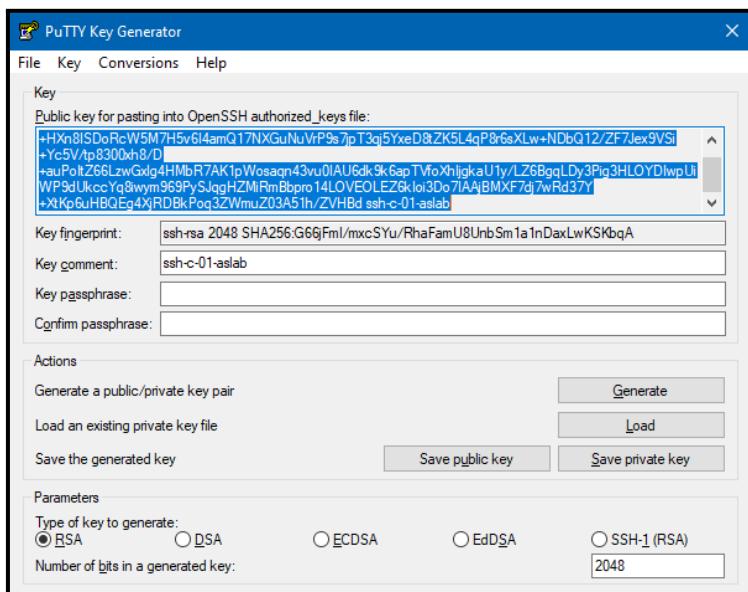
Key comment mirip dengan **nama pengguna** sementara **key passphrase** mirip dengan **kata sandi**.

- **Mengubah Key comment**

Komentar Kunci : SSH-Plug-NomorKelompok

Contoh : SSH-C-01

- Selanjutnya, klik **Save private key**
- **Salin** semua nilai pada **Public key box**, seperti gambar berikut ini



- Kemudian, **tempelkan** pada **SSH Keys box** di GCP VM seperti gambar berikut

Security

Shielded VM and SSH keys

Shielded VM [?](#)

Turn on all settings for the most secure configuration.

Turn on Secure Boot [?](#)

Turn on vTPM [?](#)

Turn on Integrity Monitoring [?](#)

VM access

Manage how users connect to the VM

By default, when you connect to a VM using this console or gcloud, your SSH keys are generated automatically. [Learn more](#)

Control VM access through IAM permissions [?](#)
Link VM access to the user's IAM role. Enables OS Login. [Learn more](#)

Require 2-step verification
Require a second form of user authentication. [Learn more](#)

Block project-wide SSH keys
When checked, project-wide SSH keys cannot access this instance. [Learn more](#)

Add manually generated SSH keys

Add your own keys for VM access through a 3rd-party tool. You cannot use these keys when IAM-based access (using OS Login) is enabled. [Learn more](#)

SSH key 1 * –
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDL1LVkXVywXAU/DQ1vGo;

Enter public SSH key

+ ADD ITEM

MANAGE ACCESS

Untuk Pengguna **Mac / Linux** untuk menghasilkan kunci SSH:

- **Buka Terminal**
- Masukkan perintah berikut di jendela Terminal.

```
ssh-keygen -t rsa
```

Ini memulai proses pembuatan kunci. Saat Anda menjalankan perintah ini, utilitas `ssh-keygen` meminta Anda untuk menunjukkan dimana menyimpan kunci.

- **Tekan** tombol **ENTER** untuk menerima lokasi default. Utilitas `ssh-keygen` meminta Anda untuk frasa **passphrase**
- **Ketik** kata **passphrase**. Anda juga dapat **menekan** tombol **ENTER** untuk **menerima default (tanpa frasa sandi)**. Namun, ini tidak disarankan. Anda harus **memasukkan** kata **sandi** untuk **kedua kalinya** untuk **melanjutkan**.
- Setelah Anda **mengkonfirmasi passphrase**, sistem **generates key pair**.

```
Your identification has been saved in /Users/myname/.ssh/id_rsa.  
Your public key has been saved in /Users/myname/.ssh/id_rsa.pub.
```

```

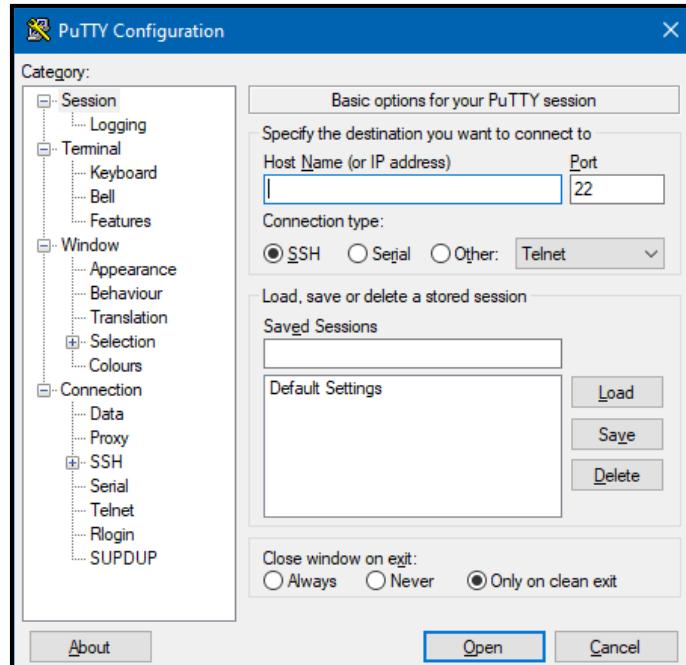
The key fingerprint is:
ae:89:72:0b:85:da:5a:f4:7c:1f:c2:43:fd:c6:44:38 myname@mymac.local
The key's randomart image is:
+--[ RSA 2048]--+
|          .       |
|          . . o   |
|          o . . S |
|          + + o . +|
|          . + o = o +|
|          o...o * o |
|          . oo.o .  |
+-----+

```

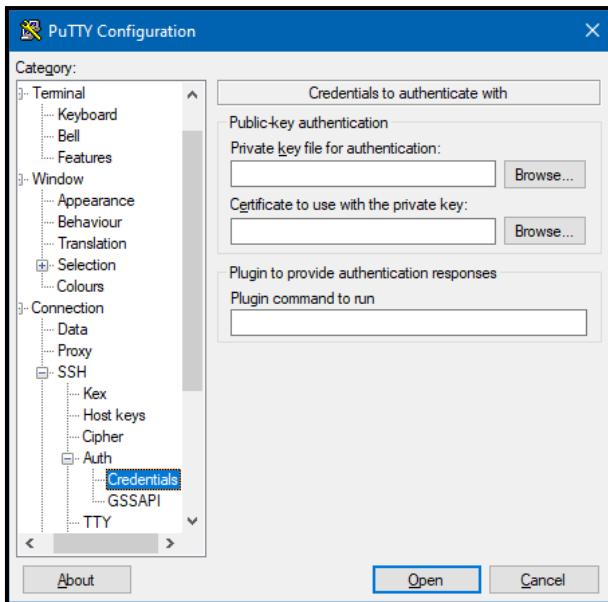
- Kemudian, Anda dapat **menyalin public key** yang disimpan ke file **id_rsa.pub** dan **menempatkannya** ke **SSH Keys box** di VM GCP.
- Terakhir, Anda dapat mengklik tombol untuk **Membuat** VM kustom Anda. Anda akan membutuhkan beberapa menit sebelum Anda dapat mengaksesnya (tergantung pada koneksi internet Anda)
 - Selanjutnya, kita akan mencoba mengakses vm dari pc kita sendiri.

Untuk **Pengguna Windows**:

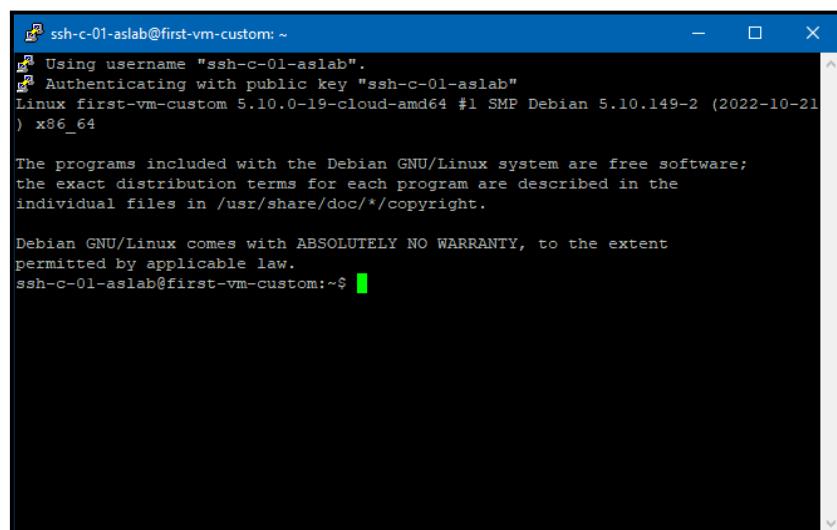
- Buka aplikasi Putty dan gambar berikut menunjukkan



- Salin **IP Eksternal** dari halaman instans VM dan tempelkan ke **Host Name**. Isi **Port** sebagai **22**.
- Selanjutnya buka bagian **SSH** di bilah samping dan buka bagian **Credentials**. gambar berikut memperlihatkan



- d. Klik **browse** pada bagian **Private key file** dan pilih private key sebelumnya yang kita simpan ketika kita generate SSH key.
- e. Terakhir, klik **buka**. Jika Putty **tidak dapat mengaksesnya**, coba dari **langkah a** tetapi **ubah Host Name** sebagai **SSHKeyComment@ExternalIP**
Misalnya : ssh-c-01-aslab@34.128.75.10
- f. Jika gambar berikut ditampilkan, itu berarti Anda sudah dapat mengakses vm Anda dari pc Anda sendiri.



Untuk Pengguna **Mac/Linux** :

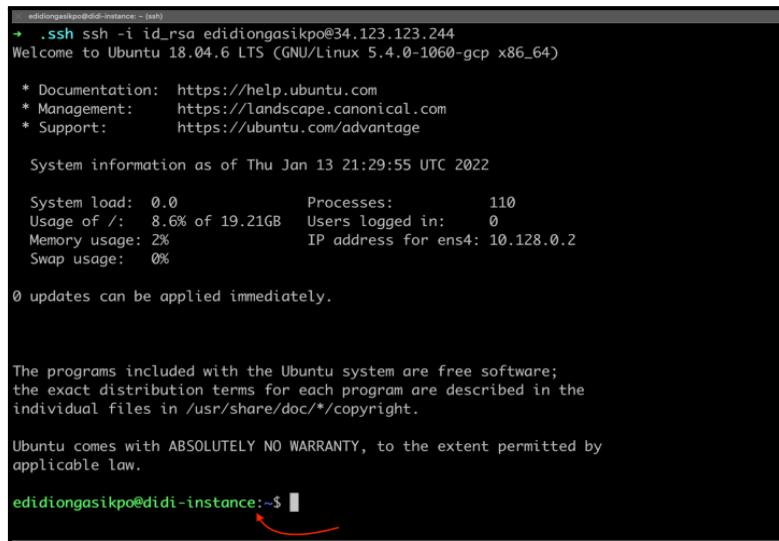
- a. Buka Terminal
- b. Tulis kode berikut

```
ssh -i PATH_TO_PRIVATE_KEY USERNAME@EXTERNAL_IP
```

Ubah ke lokasi pada kunci pribadi dan nama pengguna atau komentar kunci dari kunci SSH yang Anda buat dan tempel ip eksternal vm. Misalnya

```
ssh -i id_rsa edidiongasikpo@34.123.123.244
```

- c. Jika gambar berikut ditampilkan, itu berarti Anda sudah berhasil menghubungkan



```
edidiongasikpo@didid-instance:~$ .ssh ssh -i id_rsa edidiongasikpo@34.123.123.244
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1060-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Jan 13 21:29:55 UTC 2022

System load: 0.0      Processes:           110
Usage of /: 8.6% of 19.21GB  Users logged in: 0
Memory usage: 2%      IP address for ens4: 10.128.0.2
Swap usage: 0%

0 updates can be applied immediately.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

edidiongasikpo@didid-instance:~$
```

7. Untuk memastikan bahwa itu benar-benar terhubung, **tulis** perintah berikut

```
df -H
```

Dan ini adalah **contoh output**.

Filesystem	Size	Used	Avail	Use%	Mounted on
udev	870M	0	870M	0%	/dev
tmpfs	176M	365k	176M	1%	/run
/dev/sda1	11G	1.9G	8.0G	19%	/
tmpfs	880M	0	880M	0%	/dev/shm
tmpfs	5.3M	0	5.3M	0%	/run/lock
/dev/sda15	130M	6.2M	124M	5%	/boot/efi
tmpfs	176M	0	176M	0%	/run/user/1000

8. Kami dapat mencoba mengakses vm dari browser dengan IP eksternal

- a. Kita perlu **memperbarui package** menggunakan perintah berikut

```
sudo apt update
```

- b. Kita perlu **menginstal apache2** package menggunakan perintah berikut

```
sudo apt install apache2
```

- c. Kita perlu **memindahkan direktori kita** dengan menggunakan perintah berikut

```
cd /var/www/html/
```

- d. Setelah itu, kita perlu **menghapus index.html** dengan menggunakan perintah berikut

```
sudo rm index.html
```

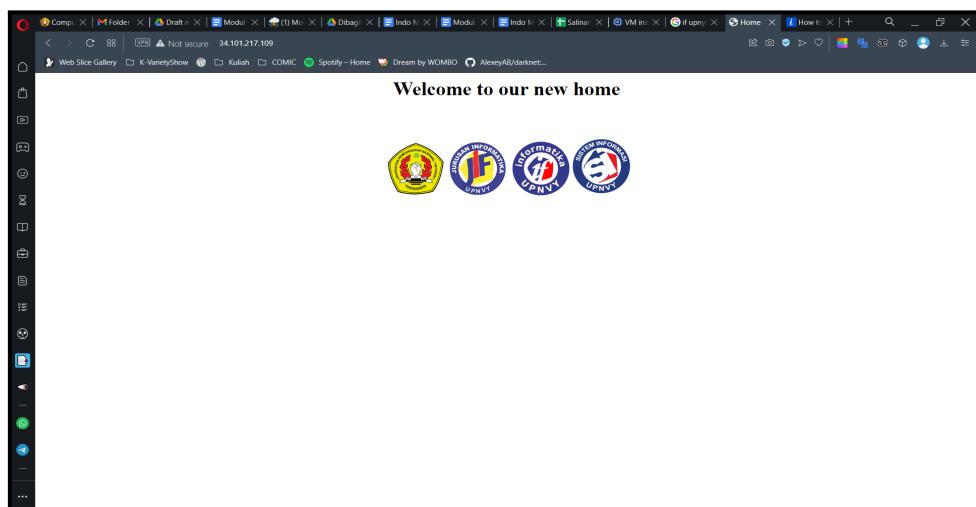
- e. Kita perlu membuat html file baru, namun sebelum itu, kita perlu menyalin isi dari index.html dari link berikut <https://bit.ly/modul-iaas-index>

- f. Buat index.html file baru dengan menggunakan perintah berikut

```
sudo nano index.html
```

- g. Kita dapat **menempelkan kode html** dari step sebelumnya dengan menggunakan **ctrl+shift+v** dan **menyimpannya** dengan menggunakan **ctrl+s**

- h. Kita sudah dapat mengakses vm dengan menggunakan IP eksternal, seperti gambar berikut



MODUL 3 - 1

MENGENAL PLATFORM AS A SERVICE

DENGAN APP ENGINE

App Engine adalah salah satu penawaran Platform as a Service (PaaS) di Google Cloud Platform. App Engine adalah platform tanpa server dan terkelola sepenuhnya untuk mengembangkan dan menghosting aplikasi web dalam skala besar. Anda dapat mengembangkan aplikasi Anda dari beberapa bahasa populer, *framework*, dan *library*, dan App Engine akan menangani penyediaan server dan menskalakan aplikasi Anda berdasarkan permintaan. Dalam modul ini, Anda akan mempelajari cara kerja App Engine.

A. Tujuan Praktikum

- Membandingkan lingkungan App Engine Standard dengan lingkungan App Engine Flexible
- Pratinjau aplikasi App Engine menggunakan Cloud Shell.
- Luncurkan aplikasi App Engine lalu nonaktifkan.

B. Alokasi Waktu

1 x 60 menit

C. Dasar Teori

Anda dapat menjalankan aplikasi Anda di App Engine dengan menggunakan lingkungan fleksibel App Engine atau lingkungan standar App Engine. Anda juga dapat memilih untuk menggunakan kedua lingkungan secara bersamaan untuk aplikasi Anda dan memungkinkan layanan Anda memanfaatkan manfaat setiap lingkungan.

Kapan harus memilih lingkungan standar

- Instance aplikasi dijalankan di sandbox, menggunakan lingkungan runtime dari bahasa yang didukung yang tercantum di bawah ini.
- Aplikasi yang perlu berurusan dengan penskalaan cepat.
- Lingkungan standar optimal untuk aplikasi dengan karakteristik berikut:

- *Source code* ditulis dalam versi tertentu dari bahasa pemrograman yang didukung:
 - Python 2.7, Python 3.7, Python 3.8, Python 3.9, Python 3.10, dan Python 3.11 (preview)
 - Java 8, Java 11, dan Java 17
 - Node.js 10, Node.js 12, Node.js 14, Node.js 16, dan Node.js 18
 - PHP 5.5, PHP 7.2, PHP 7.3, PHP 7.4, dan PHP 8.1
 - Ruby 2.5, Ruby 2.6, Ruby 2.7, dan Ruby 3.0
 - Go 1.11, Go 1.12, Go 1.13, Go 1.14, Go 1.15, Go 1.16, Go 1.18, dan Go 1.19
- Dimaksudkan untuk dijalankan secara **gratis atau dengan biaya yang sangat rendah**, di mana Anda hanya membayar apa yang Anda butuhkan dan saat Anda membutuhkannya. Misalnya, aplikasi Anda dapat menskalakan ke 0 instans saat tidak ada lalu lintas.
- Mengalami lonjakan **lalu lintas yang tiba-tiba dan ekstrem** yang memerlukan penskalaan segera.

Kapan harus memilih lingkungan yang fleksibel

- Instance aplikasi berjalan dalam container Docker di mesin virtual (VM) Compute Engine.
- Aplikasi yang menerima lalu lintas yang konsisten, mengalami fluktuasi lalu lintas reguler, atau memenuhi parameter untuk menaikkan dan menurunkan skala secara bertahap.
- Lingkungan yang fleksibel optimal untuk aplikasi dengan karakteristik berikut:
 - *Source code* yang ditulis dalam versi salah satu bahasa pemrograman yang didukung:
 - Python, Java, Node.js, Go, Ruby, PHP, atau .NET
 - Berjalan di Docker container yang menyertakan runtime khusus atau *source code* yang ditulis dalam bahasa pemrograman lain.
 - Menggunakan atau bergantung pada framework yang menyertakan kode asli.
 - Mengakses resource atau layanan project Google Cloud Anda yang berada di jaringan Compute Engine.

D. Langkah Praktikum

Aktifkan Cloud Shell

Cloud Shell adalah mesin virtual yang dimuat dengan alat pengembangan. Ini menawarkan direktori home 5GB yang persisten dan berjalan di Google Cloud. Cloud Shell memberikan akses command line ke resource Google Cloud Anda.

1. Klik Aktifkan Cloud Shell  di bagian atas Google Cloud Console.

Saat Anda terhubung, Anda sudah diautentikasi, dan proyek disetel ke PROJECT_ID Anda. Outputnya berisi baris yang mendeklarasikan PROJECT_ID untuk sesi ini:

```
Your Cloud Platform project in this session is set to YOUR_PROJECT_ID
```

gcloud adalah alat baris perintah untuk Google Cloud. Sudah diinstal sebelumnya di Cloud Shell dan mendukung penyelesaian tab.

2. (Opsional) Anda dapat mencantumkan nama akun aktif dengan perintah ini:

```
gcloud auth list
```

3. Klik **Authorize**.

4. Output Anda sekarang akan terlihat seperti ini:

Output:

```
ACTIVE: *
ACCOUNT: praktikan@gmail.com
To set the active account, run:
$ gcloud config set account `ACCOUNT`
```

5. (Opsional) Anda dapat mencantumkan project ID dengan perintah ini:

```
gcloud config list project
```

Output:

```
[core]
project = <project_ID>
```

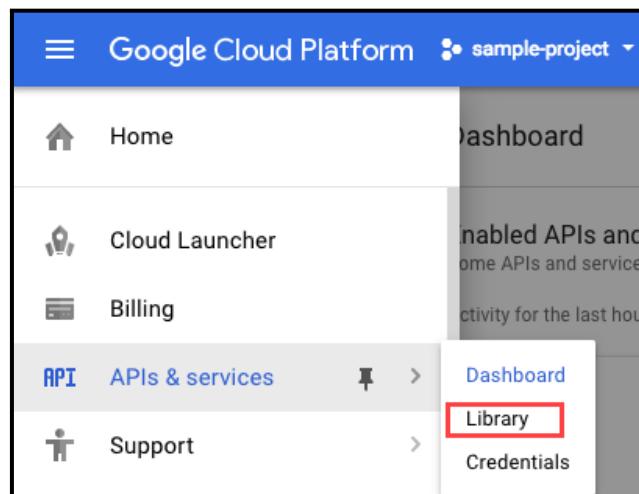
Contoh output:

```
[core]
project = praktikum-cloud
```

Aktifkan API Admin Google App Engine

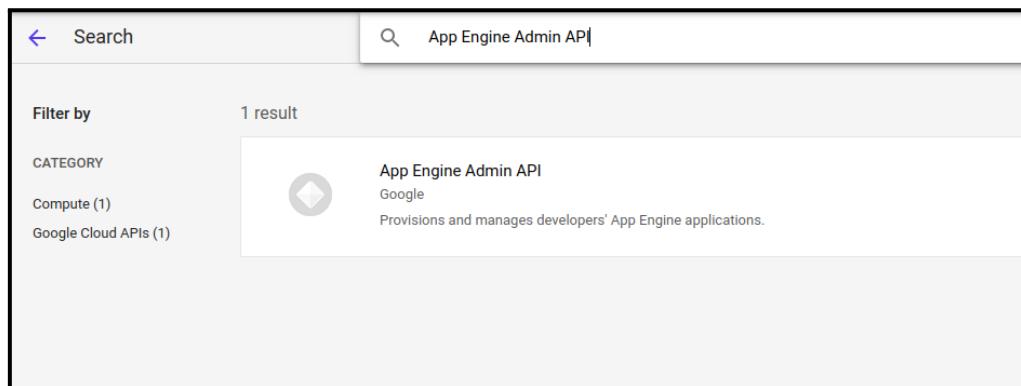
App Engine Admin API memungkinkan developer menyediakan dan mengelola Aplikasi App Engine mereka

1. Di menu sebelah kiri, klik **APIs & Services > Library**.

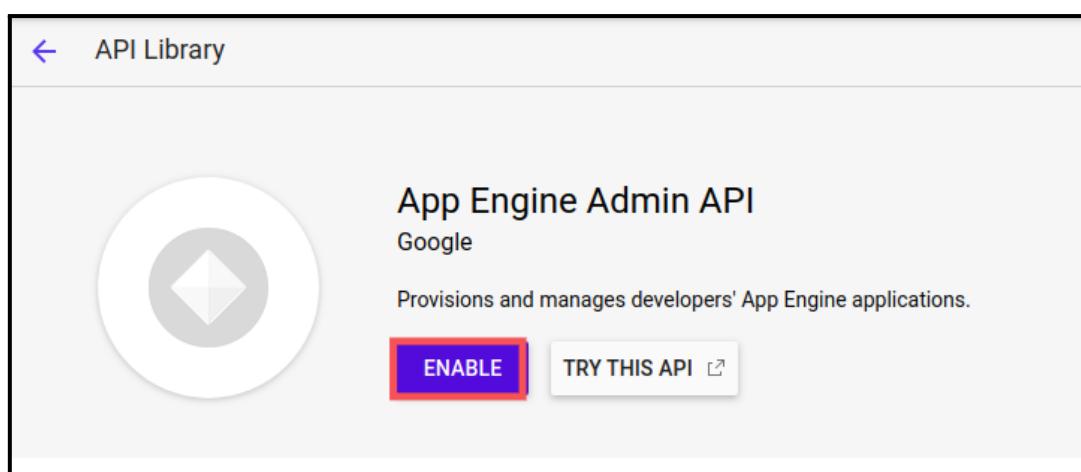


2. Ketik "App Engine Admin API" dalam search box.

3. Klik **App Engine Admin API**



4. Klik **Enable**.



Buat Server untuk Mendengarkan Permintaan HTTP

Inti dari layanan web Anda adalah server HTTP. Contoh kode dalam panduan ini menggunakan framework Express.js untuk menangani permintaan HTTP, tetapi Anda bebas menggunakan framework web pilihan Anda.

1. Buat folder baru bernama my-nodejs-service untuk layanan Node.js Anda

```
mkdir my-nodejs-service
```

2. Arahkan ke folder di terminal Anda

```
cd my-nodejs-service
```

3. Buat file package.json dengan menjalankan

```
npm init
```

4. Tambahkan express sebagai dependensi dengan menjalankan

```
npm install express
```

Dan konfirmasikan bahwa Express muncul di bidang dependensi file package.json Anda. Ini contohnya:

```
{
  ...
  "dependencies": {
    "express": "^4.16.3"
  }
  ...
}
```

5. Tambahkan "start" awal ke file package.json Anda:

```
"scripts": {
  "start": "node server.js"
}
```

6. Buat file bernama server.js di folder yang sama dan tambahkan kode berikut

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello from App Engine!');
});

// Listen to the App Engine-specified port, or 8080 otherwise
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
```

```
  console.log(`Server listening on port ${PORT}...`);  
});
```

7. Buat file bernama app.yaml di folder yang sama dan tambahkan kode berikut

```
runtime: nodejs16
```

File app.yaml menentukan setelan untuk lingkungan runtime layanan App Engine Anda. Layanan Anda tidak akan diterapkan tanpa file ini.

Pada titik ini, Anda harus memiliki struktur file seperti berikut:

```
my-nodejs-service/  
node_modules  
app.yaml  
package.json  
package-lock.json  
server.js
```

Terapkan Aplikasi Anda

1. Untuk menerapkan aplikasi Anda ke App Engine, jalankan perintah berikut dari dalam direktori root aplikasi tempat file app.yaml berada:

```
gcloud app deploy
```

Output:

```
[1] asia-east2 (supports standard and flexible)  
[2] asia-northeast1 (supports standard and flexible)  
[3] asia-northeast2 (supports standard and flexible)  
[4] asia-northeast3 (supports standard and flexible)  
[5] asia-south1 (supports standard and flexible)  
[6] australia-southeast1 (supports standard and flexible)  
[7] europe-west (supports standard and flexible)  
[8] europe-west2 (supports standard and flexible)  
[9] europe-west3 (supports standard and flexible)  
[10] europe-west6 (supports standard and flexible)  
[11] northamerica-northeast1 (supports standard and flexible)
```

2. Masukkan wilayah tempat Anda ingin aplikasi Anda berada.

Output:

```
Services to deploy:  
  
descriptor: [/home/praktikan/my-nodejs-service/app.yaml]  
source: [/home/praktikan/my-nodejs-service]  
target project: [project-id]
```

```
target service: [default]
target version: [20230101t141111]
target url: [https://project-id.as.r.appspot.com]
target service account: [App Engine default service account]
```

3. Masukkan Y saat diminta untuk mengonfirmasi penyebaran layanan.

Sample output:

```
Beginning deployment of service [default]...
Some files were skipped. Pass `--verbosity=info` to see which ones.
You may also view the gcloud log file, found at
[/tmp/tmp.YZRoP4bCoj/logs/2017.11.17/09.08.37.201396.log].
=====
[ Uploading 5 files to Google Cloud Storage
=====
File upload done.
Updating service [default]...done.
Updating service [default]...Waiting for operation
[apps/qwiklabs-gcp-e6160e374e92ffbf/operations/bf540c31-338f-4532-bcdc
-e47768040d0c] to complete...done.
Updating service [default]...done.
Deployed service [default] to
[https://qwiklabs-gcp-e6160e374e92ffbf.appspot.com]
You can stream logs from the command line by running:
  $ gcloud app logs tail -s default
To view your application in the web browser run:
  $ gcloud app browse
```

Lihat Aplikasi Anda

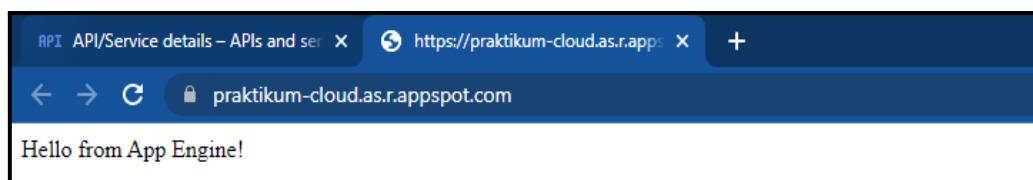
1. Untuk meluncurkan browser Anda, masukkan perintah berikut:

```
gcloud app browse
```

Sample output, link Anda akan berbeda:

```
Did not detect your browser. Go to this link to view your app:
https://project-id.appspot.com
```

2. Klik link untuk melihat aplikasi Anda.



Aplikasi Anda disebarluaskan dan Anda dapat membaca pesan singkat di browser Anda.

Membuat Perubahan

Sekarang buat perubahan pada aplikasi sampel Anda.

1. Buka file server.js dengan editor nano:

```
nano server.js
```

2. Ubah "hello world!" ke "goodbye world!".
3. Tekan CTRL + X > Y > Enter untuk keluar dan menyimpan file.
4. Pada Cloud Shell, jalankan perintah berikut untuk menerapkan ulang aplikasi Anda:

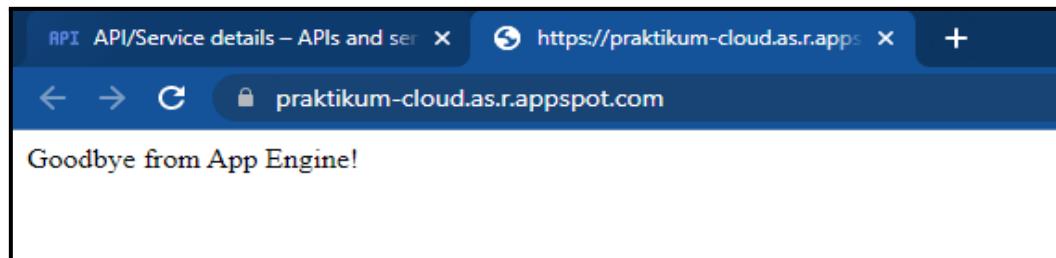
```
gcloud app deploy
```

5. Masukkan Y saat diminta untuk mengonfirmasi penyebaran layanan.

Anda akan menerima output berikut:

```
To view your application in the web browser run:  
$ gcloud app browse
```

6. Segarkan tab browser dengan penerapan App Engine Anda. Anda harus melihat yang berikut ini:



MODUL 3 - 2

MENGENAL PLATFORM AS A SERVICE

DENGAN CLOUD RUN

Cloud Run adalah salah satu penawaran Platform as a Service (PaaS) di Google Cloud Platform. Cloud Run adalah implementasi Google Cloud dari Knative, *deployment* tanpa server ke container tempat Anda dapat menerapkan dan mengelola web tanpa overhead yang diperlukan untuk penerapan berbasis VM atau Kubernetes. Cloud Run juga memberi Anda kemampuan untuk menurunkan skala ke nol saat tidak ada permintaan yang masuk ke web Anda. Dalam modul ini, Anda akan mempelajari cara bekerja dengan Cloud Run.

A. Tujuan Praktikum

- Buat image Docker dengan Cloud Build dan unggah ke gcr.io
- Deploy image Docker ke Cloud Run
- Manage deployment Cloud Run
- Siapkan endpoint untuk aplikasi di Cloud Run

B. Alokasi Waktu

1 x 60 menit

C. Dasar Teori

Fitur Utama Cloud Run:

- Bahasa pemrograman apa pun, library apa pun, biner apa pun
 - Di Cloud Run, Anda dapat menggunakan bahasa pemrograman pilihan Anda, library sistem operasi atau bahasa apa pun, atau biner Anda sendiri.
- Manfaatkan standar dan alur kerja kontainer
 - Kontainer telah menjadi standar untuk mengemas dan menyebarkan kode dan dependensinya. Cloud Run cocok dipadukan dengan ekosistem kontainer seperti cloud build, cloud code, registry artefak, dan docker.

- Bayar per penggunaan
 - Di Cloud Run, Anda hanya membayar saat kode berjalan, ditagih hingga 100 milidetik.
- dll.

D. Langkah Praktikum

Aktifkan Cloud Shell

Cloud Shell adalah mesin virtual yang dimuat dengan alat pengembangan. Ini menawarkan direktori home 5GB yang persisten dan berjalan di Google Cloud. Cloud Shell memberikan akses command line ke resource Google Cloud Anda.

1. Klik Aktifkan Cloud Shell  di bagian atas Google Cloud Console.

Saat Anda terhubung, Anda sudah diautentikasi, dan proyek disetel ke PROJECT_ID Anda. Outputnya berisi baris yang mendeklarasikan PROJECT_ID untuk sesi ini:

```
Your Cloud Platform project in this session is set to YOUR_PROJECT_ID
```

gcloud adalah alat baris perintah untuk Google Cloud. Sudah diinstal sebelumnya di Cloud Shell dan mendukung penyelesaian tab.

2. (Opsiional) Anda dapat mencantumkan nama akun aktif dengan perintah ini:

```
gcloud auth list
```

3. Klik **Authorize**.

4. Output Anda sekarang akan terlihat seperti ini:

Output:

```
ACTIVE: *
ACCOUNT: praktikan@gmail.com
To set the active account, run:
$ gcloud config set account `ACCOUNT`
```

5. (Opsiional) Anda dapat mencantumkan project ID dengan perintah ini:

```
gcloud config list project
```

Output:

```
[core]
project = <project_ID>
```

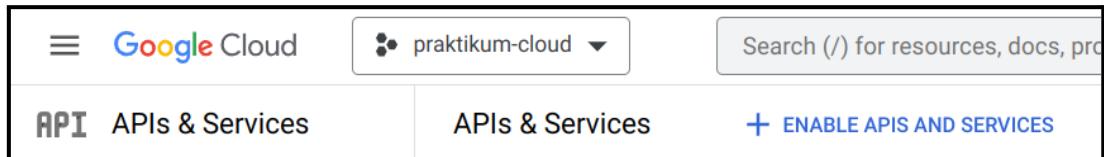
Contoh output:

```
[core]
project = praktikum-cloud
```

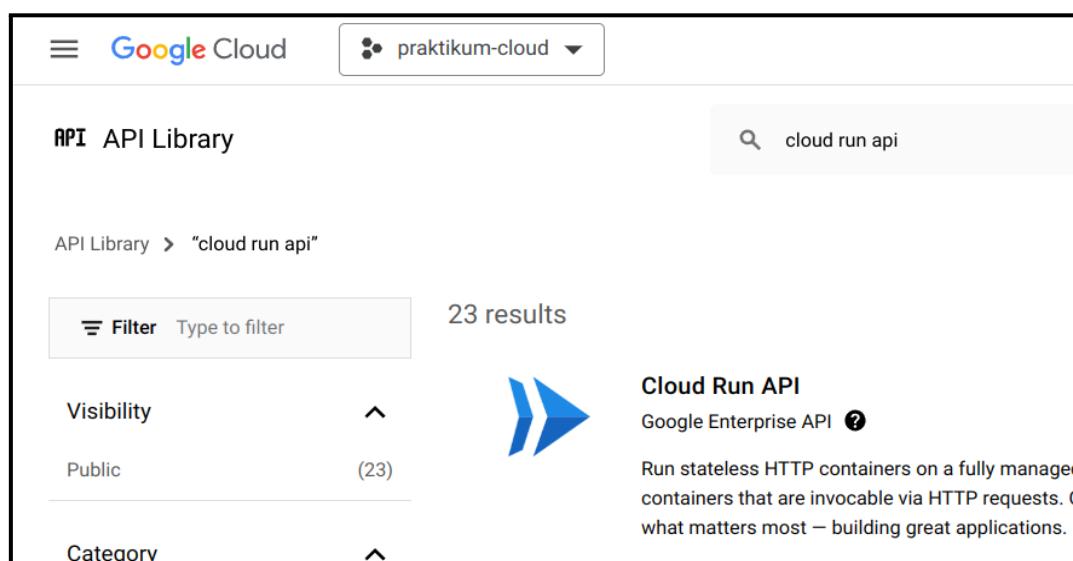
Aktifkan Google Cloud Run API

App Engine Admin API memungkinkan pengembang menyediakan dan mengelola Aplikasi App Engine mereka

1. Di menu kiri, klik **APIs & Services > Dashboard > Enable APIs And Services**.



2. Ketikan "Cloud Run API" di dalam search box.
3. Klik **Cloud Run API** lalu klik **ENABLE**



Buat Server untuk Mendengarkan Permintaan HTTP

Inti dari layanan web Anda adalah server HTTP. Contoh kode dalam panduan ini menggunakan framework Express.js untuk menangani permintaan HTTP, tetapi Anda bebas menggunakan framework web pilihan Anda.

1. Buat folder baru bernama my-nodejs-service-cr untuk layanan Node.js Anda

```
mkdir my-nodejs-service-cr
```

2. Arahkan ke folder di terminal Anda

```
cd my-nodejs-service-cr
```

3. Buat file package.json dengan menjalankan

```
npm init
```

4. Tambahkan express sebagai dependensi dengan menjalankan:

```
npm install express
```

Dan konfirmasikan bahwa Express muncul di file's dependency field package.json Anda. Ini contohnya:

```
{
  ...
  "dependencies": {
    "express": "^4.16.3"
  }
  ...
}
```

5. Tambahkan skrip "start" ke file package.json Anda:

```
"scripts": {
  "start": "node server.js"
}
```

6. Buat file bernama server.js di folder yang sama dan tambahkan kode berikut

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello from App Engine!');
});

// Listen to the App Engine-specified port, or 8080 otherwise
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server listening on port ${PORT}...`);
});
```

7. Buat Dockerfile di folder yang sama dan tambahkan kode berikut

```
FROM node:16
WORKDIR /app
COPY package.json /app
RUN npm install
COPY . /app
CMD ["npm", "start"]
```

Di tahap ini, Anda harus memiliki struktur file seperti berikut:

```
my-nodejs-service-cr/
package.json
server.js
Dockerfile
```

Buat Docker Container dengan Cloud Build

1. Untuk membuat Docker container dengan Cloud Build, Anda harus memastikan untuk mengaktifkan Cloud Build API. Jalankan perintah berikut untuk mengaktifkannya:

```
gcloud services enable cloudbuild.googleapis.com
```

2. Setelah mengaktifkan API, jalankan perintah berikut untuk memulai proses pembuatan:

```
gcloud builds submit --tag
gcr.io/${GOOGLE_CLOUD_PROJECT}/my-nodejs-service-cr:1.0.0 .
```

3. Untuk melihat histori build Anda atau melihat prosesnya secara real-time, Anda dapat membuka Cloud Console, lalu klik menu Navigasi  > Cloud Build > History. Di sana, Anda dapat melihat daftar semua bangunan Anda sebelumnya, tetapi seharusnya hanya ada yang Anda buat.

Sample output:



Build	Source	Git commit	Trigger	Started	Image target	Image tag
985ce3e5-aeba...	gs://gcb-docs-project_cloudbuild/source/1508159187.8-b0d8841d51674a30aebe1e55bb99486f.tgz	–	–	9:06 AM	gcr.io/gcb-docs-project/quickstart-image	latest
d2e89f1b-cbf9...	gs://gcb-docs-project_cloudbuild/source/1508158566.55-725755714baa4b7e9e99984c422ec4e2.tgz	–	–	8:56 AM	gcr.io/gcb-docs-project/quickstart-image	latest

Men-deploy Container ke Cloud Run

1. Untuk *deploy* aplikasi Anda, masukkan perintah berikut:

```
gcloud run deploy
--image=gcr.io/${GOOGLE_CLOUD_PROJECT}/my-nodejs-service-cr:1.0.0
--platform managed
```

Anda akan diminta untuk menentukan wilayah mana yang ingin Anda jalankan. Pilih wilayah yang paling dekat dengan Anda, lalu terima nama layanan default yang disarankan (mynodejsservicecr).

Sample output:

```
Please specify a region:
[1] asia-east1
[2] asia-east2
[3] asia-northeast1
[4] asia-northeast2
[5] asia-northeast3
```

```
[6] asia-south1
. .
[34] us-west3
[35] us-west4
[36] cancel
Please enter numeric choice or text value (must exactly match list item):
```

Untuk tujuan pengujian, izinkan permintaan yang tidak diautentikasi ke aplikasi. Masukkan y pada prompt.

```
Allow unauthenticated invocations to [mynodejsservicecr] (y/N)?
```

- Untuk memverifikasi bahwa penerapan berhasil dibuat, jalankan perintah berikut:

```
gcloud run services list
```

Sample output:

```
✓
SERVICE: mynodejsservicecr
REGION: asia-southeast1
URL: https://mynodejsservicecr-q24ea4nura-as.a.run.app
LAST DEPLOYED BY: praktikan@gmail.com
LAST DEPLOYED AT: 2023-02-04T14:51:21.686138Z
```

Output menunjukkan beberapa hal. Anda dapat melihat *deployment* Anda, serta pengguna yang men-*deploy* (alamat email Anda) dan URL yang dapat Anda gunakan untuk mengakses aplikasi. Sepertinya semuanya berhasil dibuat!

Buka URL yang disediakan dalam daftar layanan di browser web Anda dan Anda akan melihat situs web yang sama dengan yang Anda pratinjau secara lokal.

Buat Revisi Baru dengan Konkurensi Lebih Rendah

Sekarang buat perubahan pada aplikasi sampel Anda.

- Deploy ulang container image yang sama dengan nilai konkurensi 1 (hanya untuk tujuan pengujian) dan lihat apa yang terjadi:

```
gcloud run deploy
--image=gcr.io/${GOOGLE_CLOUD_PROJECT}/mynodejsservicecr:1.0.0 --platform
managed --concurrency 1
```

Jawab pertanyaan selanjutnya seperti yang Anda lakukan pertama kali. Setelah perintah berhasil, periksa Cloud Console untuk melihat hasilnya.

2. Dari dasbor Cloud Run, klik layanan mynodejsservicecr untuk melihat detailnya.

3. Sekarang, pulihkan konkurensi asli tanpa deploying ulang. Anda dapat menyetel nilai konkurensi ke default 80 atau 0, yang akan menghilangkan batasan konkurensi apa pun, dan menyetelnya ke maks default (yang kebetulan 80 pada saat penulisan ini). Jalankan perintah berikut di Cloud Shell untuk mengupdate revisi saat ini:

```
gcloud run deploy
--image=gcr.io/${GOOGLE_CLOUD_PROJECT}/mynodejsservicecr:1.0.0 --platform
managed --concurrency 80
```

Perhatikan bahwa revisi lain telah dibuat, lalu lintas telah dialihkan, dan konkurensi kembali ke 80.

Buat Perubahan pada Website

1. Buka file server.js dengan editor nano:

```
nano server.js
```

2. Sekarang ubah "hello world!" ke "goodbye world!".
3. Tekan **CTRL + X** > **Y** > **Enter** untuk keluar dan menyimpan file.
4. Sekarang setelah kode Anda diperbarui, Anda perlu mendeploy kembali Docker Container Anda dan menerbitkannya ke Container Registry. Anda dapat menggunakan perintah yang sama seperti sebelumnya, kecuali kali ini Anda akan memperbarui label versi!

Jalankan perintah berikut untuk meng-trigger Cloud Build baru dengan versi image terbaru 2.0.0:

```
gcloud builds submit --tag
gcr.io/${GOOGLE_CLOUD_PROJECT}/mynodejsservicecr:2.0.0 .
```

Perbarui Situs Web dengan Waktu Henti Nol

1. Dari baris perintah, Anda dapat menerapkan ulang layanan untuk memperbarui image ke versi baru dengan perintah berikut:

```
gcloud run deploy  
--image=gcr.io/${GOOGLE_CLOUD_PROJECT}/mynodejsservicecr:2.0.0 --platform  
managed
```

2. Anda akan melihat bahwa layanan Anda sekarang menggunakan versi terbaru gambar yang diterapkan dalam revisi baru.

Untuk memverifikasi perubahan Anda, navigasikan kembali ke URL eksternal layanan Cloud Run Anda dan perhatikan bahwa judul aplikasi Anda telah diperbarui.

Jalankan perintah berikut untuk mencantumkan layanan dan melihat alamat IP jika Anda lupa:

```
gcloud run services list
```

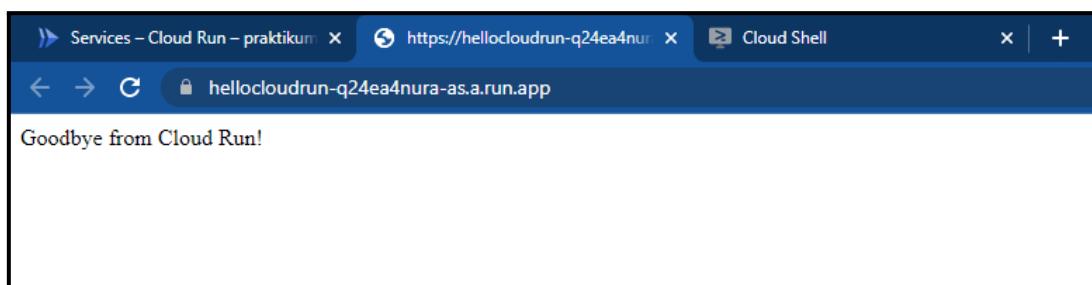
Sample output:

```
✓ Service mynodejsservicecr in region asia-southeast1

URL:      https://mynodejsservicecr-q24ea4nura-as.a.run.app
Ingress:  all
Traffic:
  100% LATEST (currently mynodejsservicecr-00008-vuz)

Last updated on 2023-02-04T14:51:21.686138Z by praktikan@gmail.com:
  Revision hellocloudrun-00008-vuz
  Image:      gcr.io/project-id/mynodejsservicecr:2.0.0
  Port:       8080
  Memory:    512Mi
  CPU:        1000m
  Service account: 795858505211-compute@developer.gserviceaccount.com
  Concurrency: 80
  Max Instances: 100
  Timeout:    300s
```

Situs web Anda sekarang harus menampilkan teks yang Anda ubah!



Selamat! Anda telah menyelesaikan modul ini.

MODUL 4

PENYIMPANAN DI CLOUD DENGAN CLOUD STORAGE

Cloud storage merupakan layanan untuk menyimpan objek-obyek Anda di Google Cloud. Sebuah objek merupakan data tetap yang terdiri dari sebuah file dengan berbagai format. Anda menyimpan objek-objek di sebuah kontainer bernama *buckets*. Semua *buckets* berhubungan dengan sebuah *project*, Anda dapat mengelompokkan *project* Anda dalam sebuah *Organization*. Setiap *project*, *bucket*, dan *object* di Google Cloud merupakan sebuah *resource* di Google Cloud, seperti Compute Engine Instance. Pada modul ini kita akan bersama-sama belajar mengenai cloud storage.

A. Tujuan Praktikum

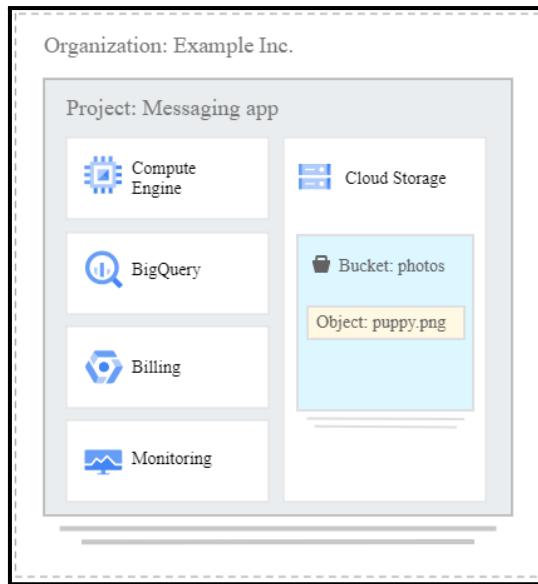
- Mengetahui apa itu cloud storage, struktur, storage class, cara interaksi, dan pengamanan data Cloud Storage
- Membuat *bucket*, *upload object* ke dalam *bucket*, membagikan *bucket*, membuat & menghapus folder
- Hosting web statis di cloud storage

B. Alokasi Waktu

2 x 60 menit

C. Dasar Teori

Cloud Storage merupakan penyimpanan objek yang global, secure, dan scalable untuk data tetap seperti gambar, text, video, maupun format lainnya. Cloud storage memiliki struktur seperti gambar di bawah:



Sumber: Google Cloud Storage Documentation

Berikut merupakan bagaimana struktur cloud storage dapat diterapkan di kasus dunia nyata:

- **Organization:** Sesuai dengan namanya, organization merupakan perusahaan atau organisasi Anda, contohnya Example Inc., membuat Google Cloud Organization bernama exampleinc.org.
- **Project:** Example inc. sedang mengembangkan beberapa aplikasi dan tiap-tiap aplikasi tersebut terhubung dengan sebuah project. Tiap project memiliki set Cloud Storage APIs tersendiri, begitu juga dengan resource lainnya.
- **Bucket:** Tiap project dapat mengandung lebih dari satu buckets, yang merupakan kontainer untuk menyimpan objects Anda. Contohnya, Anda memiliki bucket `photos` untuk semua file gambar yang dihasilkan oleh aplikasi Anda dan memisahkan bucket `videos`.
- **Object:** Merupakan file secara individu, seperti sebuah gambar dengan nama `puppy.png`

Ketika menciptakan sebuah bucket Anda memiliki pilihan bergantung pada **budget, ketersediaan, dan frekuensi akses**. Pilihan storage yang tersedia di Cloud Storage terdapat empat yang dapat dipelajari di tabel berikut:

Jenis Storage	Deskripsi	Kegunaan	Biaya (GiB per month)
Standard Storage	Penyimpanan untuk data yang diakses secara sering ("hot" data) dan/atau	"Hot" data, termasuk website, streaming video, dan aplikasi mobile	\$.02

	disimpan hanya untuk waktu yang singkat		
Nearline Storage	Layanan penyimpanan biaya rendah, andal untuk menyimpan data yang diakses secara tidak sering	Data yang dapat disimpan minimal selama 30 hari, backup data & pemulihan bencana	\$.01
Coldline Storage	Layanan penyimpanan biaya sangat rendah dan andal untuk menyimpan data yang diakses secara tidak sering	Data yang dapat disimpan selama minimal 90 hari, backup data & pemulihan bencana	\$.004
Archival Storage	Layanan penyimpanan biaya terendah dan andal untuk arsip data, online backup, dan pemulihan bencana	Data yang dapat disimpan minimal selama 365 hari, arsip jangka panjang	\$.0012

Terdapat beberapa cara untuk dapat berinteraksi dengan Google Cloud Storage, beberapa diantaranya yaitu:

- **Console:** Google Cloud Console menyediakan interface visual untuk mengatur data di browser.
- **Google cloud CLI:** Google Cloud CLI mengizinkan Anda untuk berinteraksi dengan Cloud Storage melalui terminal menggunakan perintah `gcloud storage` .
- **Client libraries:** Client library Cloud Storage mengizinkan Anda untuk dapat mengatur data menggunakan bahasa pilihan Anda, termasuk C++, C#, Go, Java, Node.js, PHP, Python, dan Ruby.
- **REST APIs:** Mengatur data dengan JSON atau XML.
- **Terraform:** Terraform merupakan tools Infrastructure-as-code (IaC) yang dapat dimanfaatkan untuk penyediaan infrastruktur untuk Cloud Storage

Google Cloud Storage memberikan Anda control untuk bagaimana Anda mengamankan dan membagikan data Anda. Berikut merupakan beberapa cara untuk mengamankan data yang sudah di upload di Google Cloud Storage:

- **Identity and Access Management (IAM):** Gunakan IAM untuk mengatur siapa saja yang dapat mengakses ke resources di Google Cloud Project. Resources tersebut termasuk buckets dan object Google Cloud Storage, dan entitas Google Cloud lainnya seperti Instance VM GCE. Anda dapat memberikan *principles* dengan akses tertentu untuk buckets dan objects, seperti `update`, `create`, atau `delete`.

- **Data Encryption:** Cloud Storage menggunakan server-side encryption untuk enkripsi data Anda secara default. Anda juga dapat menggunakan pilihan enkripsi data tambahan seperti customer-managed encryption keys dan customer-supplied encryption keys.
- **Authentication:** Memastikan bahwa semua yang mengakses data anda memiliki kredensial yang benar.
- **Bucket Lock:** Mengatur berapa lama objects di buckets harus dipertahankan dengan menentukan *retention policy*/kebijakan penyimpanan.
- **Object Versioning:** Ketika sebuah versi live dari suatu object digantikan atau dihapus, object dapat dipertahankan sebagai sebuah *non current version* jika Anda mengaktifkan Object Versioning.

Informasi lebih lanjut dapat kunjungi halaman dokumentasi Cloud Storage Berikut:

<https://cloud.google.com/storage/docs/storage-classes#standard>

<https://cloud.google.com/storage/>

<https://cloud.google.com/storage/docs/discover-object-storage-console>

D. Langkah Praktikum

Aktifkan Cloud Shell

Langkah pengaktifan cloud shell dapat dilihat di modul-modul sebelumnya.

Membuat Sebuah Bucket

Bucket merupakan kontainer dasar yang menampung data Anda di Cloud Storage.

1. Buka cloud shell.
2. Ketikkan perintah `gcloud storage buckets create` dan nama unik untuk membuat sebuah bucket, contohnya dapat dilihat di bawah ini:

```
gcloud storage buckets create gs://my-buckets/
--uniform-bucket-level-access
```

Perintah di atas akan membuat bucket dengan nama “my-bucket”. Anda harus memilih nama bucket yang globally-unique

Aturan penamaan bucket:

- a. Nama bucket hanya boleh berisi huruf kecil, karakter numerik, tanda hubung (-), garis bawah (_), dan titik (.). Spasi tidak diperbolehkan. Nama yang mengandung titik memerlukan verifikasi.
- b. Nama bucket harus diawali dan diakhiri dengan angka atau huruf.
- c. Nama bucket harus berisi 3-63 karakter. Nama yang berisi titik dapat berisi hingga 222 karakter, tetapi setiap komponen yang dipisahkan titik tidak boleh lebih dari 63 karakter.
- d. Nama bucket tidak dapat direpresentasikan sebagai alamat IP dalam notasi desimal bertitik (misalnya, 192.168.5.4).
- e. Nama bucket tidak boleh diawali dengan awalan "goog".
- f. Nama bucket tidak boleh berisi "google" atau salah eja, seperti "g00gle".

Jika berhasil maka akan muncul output seperti di bawah ini:

```
Creating gs://my-bucket/...
```

Jika nama bucket tidak globally-unique maka akan muncul output seperti di bawah ini:

```
Creating gs://my-bucket/...
ServiceException: 409 Bucket my-bucket already exists.
```

Mengunggah Object ke Dalam Bucket

1. Download gambar dari url berikut:

<https://if.upnyk.ac.id/public/uploads/logo.png>

Beri nama logo.png dan letakkan file gambar tersebut di dalam komputer Anda, contohnya di Desktop.

2. Ketikkan perintah `gcloud storage cp` untuk menyalin gambar dari lokasi file disimpan menuju ke bucket yang sudah dibuat sebelumnya:

```
gcloud storage cp Desktop/logo.png gs://my-buckets
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
Copying file://Desktop/logo.png [Content-Type=image/png]...
Uploading gs://my-bucket/logo.png: 0 B/164.3 KiB
```

3. Buat file dengan nama index.html dengan isi sebagai berikut:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
```

```

<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
<title>If Site</title>
</head>
<body>
<table cellspacing="20">
<tr>
<td>

</td>
<td>
<h1>Jurusan Informatika Universitas Pembangunan Veteran
Yogyakarta</h1> <br>
<p>
<a
    href="https://www.google.com/maps/place/Universitas+Pembangunan+Nasional+
%22Veteran%22+Yogyakarta+-+Kampus+2+Babarsari/@-7.7673607,110.3881035,13z
/data=!4m9!1m2!2m1!1skampus+2!3m5!1s0x2e7a599155555555:0x536eb168b1dca148
!8m2!3d-7.7821544!4d110.4146253!15sCghrYW1wdXMgMpIBEXB1YmxpY191bml2ZXJzaX
R54AEA"
        >Yogyakarta, Indonesia</a>
    >
</i>
</p>
</td>
</table>
</body>
</html>

```

4. Buat file dengan nama error.html dengan isi sebagai berikut:

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
<title>If Site</title>
</head>
<body>
<h1>404 Page Not Found</h1>
</body>
</html>

```

5. Tambahkan index.html dan error.html ke dalam bucket yang sudah dibuat.

Mengunduh Object dari Bucket

1. Gunakan perintah `gcloud storage cp` untuk mengunduh gambar yang sudah Anda simpan di bucket ke suatu tempat di komputer, seperti Desktop:

```
gcloud storage cp gs://my-buckets/logo.png Desktop/logo2.png
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
Copying gs://my-bucket/logo.png...
Downloading file://Desktop/logo2.png: 0 B/164.3 KiB
```

Menyalin Object ke Sebuah Folder di Bucket

1. Gunakan perintah `gcloud storage cp` untuk membuat sebuah folder dan menyalin gambar ke dalam nya:

```
gcloud storage cp gs://my-buckets/logo.png
gs://my-buckets/new-folder/logo3.png
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
Copying gs://my-bucket/logo.png [Content-Type=image/png]...
Copying     ...my-bucket/new-folder/logo3.png: 164.3 KiB/164.3 KiB
```

Listing Isi dari Bucket atau Folder

1. Gunakan perintah `gcloud storage ls` untuk menampilkan semua konten di level teratas bucket Anda

```
gcloud storage ls gs://my-buckets
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
gs://my-bucket/logo.png
gs://my-bucket/new-folder/
```

Membuat Object Dapat Diakses Secara Publik

1. Gunakan perintah `gcloud storage buckets add-iam-policy-binding` untuk memberikan izin `all users` untuk melihat object yang disimpan dalam bucket.

```
gcloud storage buckets add-iam-policy-binding gs://my-buckets
--member=allUser --role=roles/storage.objectViewer
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
bindings:
- members:
  - allUsers
  role: roles/storage.objectViewer
```

Jika ingin menghapus akses gunakan perintah di bawah ini

```
gcloud storage buckets remove-iam-policy-binding gs://my-buckets  
--member=allUser --role=roles/storage.objectViewer
```

Menghapus Sebuah Object

1. Gunakan perintah `gcloud storage rm` untuk untuk menghapus object yang disimpan dalam bucket.

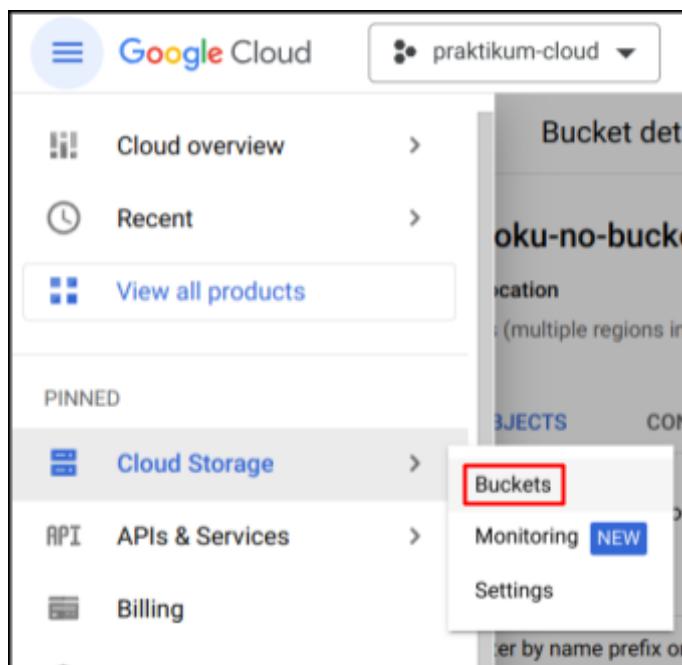
```
gcloud storage rm gs://my-bucket/new-folder/logo3.png
```

Jika berhasil maka akan menampilkan output seperti berikut.

```
Removing gs://my-bucket/new-folder/awesome-logo3.png...
```

Modifikasi Konfigurasi Website

1. Di menu sebelah kiri, klik **Cloud Storage > Buckets**

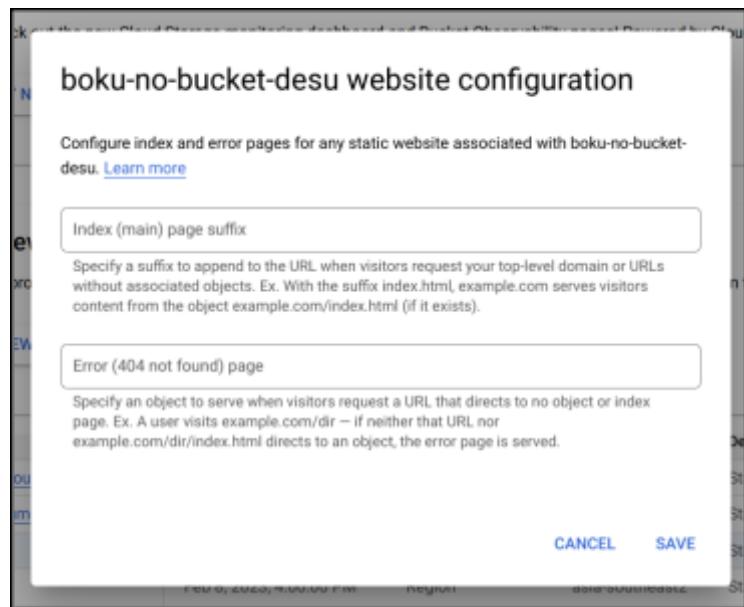


Anda akan dibawa ke halaman berikut

2. Pada daftar bucket klik tiga titik yang ada di pojok kanan bagian bucket yang ingin dikonfigurasi

3. Setelah meng-klik tiga titik tersebut maka akan muncul pilihan sebagai berikut, pilih edit website configuration

4. Setelah meng-klik edit website configuration maka akan muncul pop-up seperti berikut, masukkan index.html di bagian index page suffix, dan error.html di bagian error page lalu klik save.



5. Salin URL dari kolom public access dan buka di tab baru

Selamat Anda sudah berhasil menerapkan Web Statis di Cloud Storage! Supaya lebih baik lagi anda dapat daftarkan DNSnya.

MODUL 5

PRAKTIKUM TEKNOLOGI CLOUD

COMPUTING WEB SERVICE & API

A. Tujuan Praktikum

- Memahami konsep pembuatan Web Service dan API.
- Mampu membuat web service dengan memanfaatkan API menggunakan bahasa pemrograman javascript dengan nodejs.
- Mampu membuat API yang dapat menerima request dan mengirimkan response.

B. Alokasi Waktu

2 x 60 menit

C. Dasar Teori

1. Web Service

Web service merupakan aplikasi yang berisi sekumpulan basis data (database) dan perangkat lunak (software) atau bagian dari program perangkat lunak yang diakses secara remote oleh piranti dengan perantara tertentu. Melalui web service, memungkinkan pengguna untuk mengatasi permasalahan berupa interoperability dan mengintegrasikan sistem berbeda.

2. RESTful API

API RESTful adalah antarmuka yang digunakan oleh dua sistem komputer untuk bertukar informasi secara aman melalui internet. Sebagian besar aplikasi bisnis harus berkomunikasi dengan aplikasi internal dan pihak ketiga lainnya untuk melakukan berbagai tugas.

Misalnya, untuk menghasilkan slip gaji bulanan, sistem akun internal Anda harus berbagi data dengan sistem perbankan pelanggan Anda untuk mengotomatiskan tagihan dan berkomunikasi dengan aplikasi absensi internal. API RESTful mendukung pertukaran informasi ini karena mengikuti standar komunikasi perangkat lunak yang aman, andal, dan efisien.

3. NodeJS

Node.js adalah runtime environment untuk JavaScript yang bersifat open-source dan cross-platform. Dengan Node.js kita dapat menjalankan kode JavaScript di mana pun, tidak hanya terbatas pada lingkungan browser.

Node.js menjalankan V8 JavaScript engine (yang juga merupakan inti dari Google Chrome) di luar browser. Ini memungkinkan Node.js memiliki performa yang tinggi.

4. ExpressJS

Express.js adalah framework web app untuk Node.js yang ditulis dengan bahasa pemrograman JavaScript. Framework open source ini dibuat oleh TJ Holowaychuk pada tahun 2010 lalu.

Express.js adalah framework back end. Artinya, ia bertanggung jawab untuk mengatur fungsionalitas website, seperti pengelolaan routing dan session, permintaan HTTP, penanganan error, serta pertukaran data di server.

5. MySQL

MySQL adalah sistem manajemen database relasional (RDBMS) open-source berbasis SQL yang bekerja dengan model client-server. Kalau DBMS adalah sistem manajemen database secara umum, RDBMS merupakan software pengelolaan database berdasarkan model relasional.

Pengembang pertama MySQL adalah MySQL AB, sebuah perusahaan asal Swedia, yang memulai perjalanannya di tahun 1994. Hak kepemilikan MySQL kemudian diambil secara menyeluruh oleh perusahaan teknologi Amerika Serikat, Sun Microsystems, ketika mereka membeli MySQL AB pada tahun 2008.

Di tahun 2010, Oracle yang adalah salah satu perusahaan teknologi terbesar di Amerika Serikat mengakuisisi Sun Microsystems. Sejak saat itu, MySQL sepenuhnya dimiliki oleh Oracle.

6. CORS untuk CSRF Protection

CORS (Cross-Origin Resource Sharing) adalah protokol yang dibangun di atas HTTP untuk memungkinkan Javascript pada halaman yang berasal dari satu situs untuk mengakses metode di situs lain.

CORS merupakan protokol penghubung antara browser dan web-service yang memberitahu browser bahwa itu adalah "OK" untuk mengeksekusi kode Javascript dari panggilan lintas domain.

D. Langkah Praktikum

1. Mempersiapkan environment

Dalam langkah pertama ini, hal pertama yang harus praktikan lakukan adalah menginstall node JS dan express JS terlebih dahulu. Untuk node js sendiri, praktikan bisa mengunduhnya melalui <https://nodejs.org/en/download/>. Pilihlah sesuai dengan OS yang praktikan gunakan.

Setelah proses instalasi node JS selesai, praktikan bisa membuat folder project terlebih dahulu. Dalam praktikum kali ini, studi kasus yang akan digunakan adalah aplikasi **to do list**. Untuk itu, silahkan buatlah folder dengan nama **ToDoList**.

Kemudian, silahkan masuk ke terminal yang sudah berada di direktori project to do list. Jalankan perintah berikut ini untuk menginisialisasi project node js.

```
npm init
```

Saat proses inisialisasi, biarkan konfigurasi secara default. Jadi hanya perlu menekan tombol enter saja.

Selanjutnya, saatnya untuk menginstall express JS, mysql, body-parser dan juga CORS sebagai dependencies yang akan digunakan nantinya. Praktikan bisa menggunakan perintah berikut ini di terminal.

```
npm install express cors mysql body-parser
```

Sampai sini, proses persiapan environment sudah selesai. Selanjutnya praktikan bisa langsung mengerjakan project express untuk pembuatan API.

2. Mempersiapkan konfigurasi database dan menghubungkan ke project

Langkah berikutnya adalah mempersiapkan database terlebih dahulu. Praktikan bisa membuka control panel XAMPP untuk menjalankan apache server dan mysql database. Kemudian silahkan buat database dengan nama **todolist**. Kemudian buatlah tabel dengan nama **task** yang memiliki kolom **id** (primary key, integer, auto increment), **name** (varchar 255), **time** (datetime), dan **status** (boolean).

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	 Change  Drop 
2	name	varchar(255)	utf8mb4_general_ci		No	None			 Change  Drop 
3	time	datetime			No	None			 Change  Drop 
4	status	tinyint(1)			No	None			 Change  Drop 

Gambar 2.1

Kurang lebih nanti bentuk strukturnya seperti pada gambar di atas. Nah, setelah itu, silahkan membuka folder project express yang sudah dibuat sebelumnya dengan menggunakan code editor. Contohnya seperti Visual Studio Code.

```

js dbjs > ...
1  const mysql = require('mysql');
2  //local mysql db connection
3  const dbConn = mysql.createConnection({
4    host      : 'localhost',
5    user      : 'root',
6    password  : '',
7    database  : 'todolist'
8  });
9  dbConn.connect(function(err) {
10    if (err) throw err;
11    console.log("Database Connected!");
12  });
13 module.exports = dbConn;

```

Gambar 2.2

Buatlah file bernama **db.js** kemudian tuliskan kode seperti pada gambar di atas untuk menghubungkan database ke dalam project API. Sampai sini, database sudah berhasil dikonfigurasi. Selanjutnya adalah membuat model dan juga controller untuk project todolist ini.

3. Membuat Model untuk Todolist API

Langkah selanjutnya yang harus praktikan lakukan adalah membuat file bernama **model.js**. Di dalam file **model.js**, praktikan bisa menambahkan kode program berikut ini sebagai konfigurasi awal.

```

1  // Import DB Connection from db.js
2  var dbConn = require("./db");
3
4  // Create new Object, Task
5  var Task = function (todo) {
6    this.name = todo.name;
7    this.time = todo.time;
8    this.status = todo.status ? todo.status : 0;
9  };
10

```

Gambar 3.1

Kode program di atas berfungsi untuk memanggil fungsi koneksi ke database yang dibuat sebelumnya. Kemudian dibuat juga sebuah object untuk menjadi model yang akan berguna untuk menjalankan query ke database nantinya.

- **name** : merupakan nama tugas yang berada di todolist
- **time** : merupakan waktu kapan tugas itu harus dikerjakan
- **status** : merupakan status apakah tugas sudah dikerjakan atau belum (1 atau TRUE berarti sudah dikerjakan, 0 atau FALSE berarti belum dikerjakan)

Secara default, status akan bernilai 0 atau FALSE apabila tidak diberikan inputan pada saat menambahkan tugas baru.

Sampai sini, model sudah berhasil terhubung dengan database. Selanjutnya perlu menambahkan function untuk menjalankan query untuk menambahkan, mengambil, mengedit, dan menghapus tugas.

```
11 // Create new Task
12 Task.create = function (newTask, result) {
13   dbConn.query("INSERT INTO task set ?", newTask, function (err, res) {
14     if (err) {
15       console.log("error: ", err);
16       result(null, err);
17     } else {
18       console.log(res.insertId);
19       result(null, res.insertId);
20     }
21   });
22 };
23
```

Gambar 3.2

Praktikan bisa menambahkan kode program pada gambar di atas, di file **model.js** tepat berada di bawah kode program dari **Gambar 3.1**. Fungsi di atas berguna untuk menambahkan tugas baru ke database.

Cara kerjanya yaitu dengan membuat object dari **Task** bernama **create** yang nantinya akan dipanggil di controller.

```
24 // Show all Task
25 Task.findAll = function (result) {
26   dbConn.query("SELECT * FROM task", function (err, res) {
27     if (err) {
28       console.log("error: ", err);
29       result(null, err);
30     } else {
31       console.log("Task : ", res);
32       result(null, res);
33     }
34   });
35 };
36
```

Gambar 3.3

Kemudian tambahkan kode program di atas di file **model.js** tepat berada di bawah kode program dari **Gambar 3.2**. Kode program di atas berguna untuk menampilkan semua task yang sudah tersimpan di database.

Cara kerjanya yaitu dengan membuat object dari **Task** bernama **findAll** yang nantinya akan dipanggil di controller.

```
37 // Find Task by Id
38 Task.findById = function (id, result) {
39   dbConn.query("SELECT * FROM Task WHERE id = ? ", id, function (err, res) {
40     if (err) {
41       console.log("error: ", err);
42       result(null, err);
43     } else {
44       result(null, res);
45     }
46   });
47 };
48
```

Gambar 3.4

Kemudian tambahkan kode program di atas di file **model.js** tepat berada di bawah kode program dari **Gambar 3.3**. Kode program di atas berguna untuk menampilkan satu task yang sudah tersimpan di database berdasarkan **id task**.

Cara kerjanya yaitu dengan membuat object dari **Task** bernama **findById** yang nantinya akan dipanggil di controller.

```
49 // Update Task
50 Task.update = function (id, Task, result) {
51   dbConn.query(
52     "UPDATE task SET name=?,time=?,status=? WHERE id = ?",
53     [Task.name, Task.time, Task.status, id],
54     function (err, res) {
55       if (err) {
56         console.log("error: ", err);
57         result(null, err);
58       } else {
59         result(null, res);
60       }
61     });
62   );
63 };
```

Gambar 3.5

Kemudian tambahkan kode program di atas di file **model.js** tepat berada di bawah kode program dari **Gambar 3.4**. Kode program di atas berguna untuk mengupdate satu task berdasarkan **id task**.

Cara kerjanya yaitu dengan membuat object dari **Task** bernama **update** yang nantinya akan dipanggil di controller. Terdapat 3 parameter dalam function ini yaitu id, Task, dan result. Untuk id berguna untuk **id task** yang akan diubah. Sedangkan untuk **Task** berguna untuk data task yang baru. Result merupakan hasil dari query yang dijalankan.

```
65 // Delete Task
66 Task.delete = function (id, result) {
67   dbConn.query("DELETE FROM Task WHERE id = ?", [id], function (err, res) {
68     if (err) {
69       console.log("error: ", err);
70       result(null, err);
71     } else {
72       result(null, res);
73     }
74   });
75 };
```

Gambar 3.6

Kemudian tambahkan kode program di atas di file **model.js** tepat berada di bawah kode program dari **Gambar 3.5**. Kode program di atas berguna untuk menghapus satu task yang sudah tersimpan di database berdasarkan **id task**.

Cara kerjanya yaitu dengan membuat object dari **Task** bernama **delete** yang nantinya akan dipanggil di controller.

Sampai sini, function CRUD di model sudah berhasil dibuat. Selanjutnya hanya perlu melakukan export model agar bisa dipanggil di controller nantinya. Caranya adalah dengan menambahkan kode program berikut ini.

```
module.exports = Task;
```

Nah, sekarang, model sudah selesai dibuat. Selanjutnya praktikan perlu melakukan konfigurasi untuk controllernya.

4. Membuat Controller untuk Todolist API

Nah, sekarang, model sudah selesai dibuat. Selanjutnya praktikan perlu melakukan konfigurasi untuk controllernya. Praktikan perlu membuat file bernama **controller.js**. Di dalam file **controller.js**, silahkan tambahkan kode program berikut ini.

```
const Task = require("./model");
```

Kode program di atas berguna untuk memanggil model agar bisa digunakan di controller. Selanjutnya, praktikan perlu menjalankan setiap fungsi yang sudah ada di model sebelumnya.

```
3  exports.findAll = function (req, res) {
4    Task.findAll(function (err, task) {
5      console.log("controller");
6      if (err) res.send(err);
7      console.log("res", task);
8      res.send(task);
9    });
10  };
```

Gambar 4.1

Tambahkan kode program diatas di file **controller.js** tepat berada di bawah kode program yang sudah ditambahkan sebelumnya untuk memanggil model. Kode program ini berfungsi untuk menampilkan keseluruhan task yang ada di database.

Cara kerjanya sendiri yaitu dengan memberikan nama fungsi **findAll** dan tambahkan sebagai object dari **exports** agar bisa dipanggil di router nantinya. Di dalam fungsi ini, jalankanlah **Task.findAll** untuk menjalankan fungsi **findAll** yang sudah dibuat di model sebelumnya.

```
12  exports.create = function (req, res) {
13    const new_task = new Task(req.body);
14    //handles null error
15    if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
16      res
17        .status(400)
18        .send({ error: true, message: "Please provide all required field" });
19    } else {
20      Task.create(new_task, function (err, task) {
21        if (err) res.send(err);
22        res.json({
23          error: false,
24          message: "task added successfully!",
25          data: task,
26        });
27      });
28    }
29  };

```

Gambar 4.2

Tambahkan kode program diatas di file **controller.js** tepat berada di bawah kode program yang sudah ditambahkan sebelumnya di **Gambar 4.1**. Kode program ini berguna untuk menjalankan fungsi penambahan task yang sudah ada di model sebelumnya.

Cara kerjanya yaitu dengan memberikan nama fungsi **create** dan tambahkan sebagai object dari **exports** agar bisa dipanggil di router nantinya. Di dalam fungsi ini, buatlah object baru bernama **new_task**. Kemudian buatlah error handling terlebih dahulu untuk menangani ketika input pengguna tidak sesuai atau tidak lengkap. Kemudian jalankan **Task.create** untuk menjalankan fungsi **create** yang sudah dibuat di model sebelumnya.

```
31  exports.findById = function (req, res) {
32    Task.findById(req.params.id, function (err, task) {
33      if (err) res.send(err);
34      res.json(task);
35    });
36  };

```

Gambar 4.3

Selanjutnya, tambahkan kode program diatas di file **controller.js** tepat berada di bawah kode program yang sudah ditambahkan sebelumnya untuk memanggil model. Kode program ini berfungsi untuk menampilkan suatu task berdasarkan id tertentu.

Cara kerjanya sendiri yaitu dengan memberikan nama fungsi **findAll** dan tambahkan sebagai object dari **exports** agar bisa dipanggil di router nantinya. Di dalam fungsi ini, jalankanlah **Task.findAll** untuk menjalankan fungsi **findAll** yang sudah dibuat di model sebelumnya.

```
38  exports.update = function (req, res) {
39    if (req.body.constructor === Object && Object.keys(req.body).length === 0) {
40      res
41        .status(400)
42        .send({ error: true, message: "Please provide all required field" });
43    } else {
44      Task.update(req.params.id, new Task(req.body), function (err, task) {
45        if (err) res.send(err);
46        res.json({ error: false, message: "Task successfully updated" });
47      });
48    }
49  };

```

Gambar 4.4

Selanjutnya, tambahkan kode program diatas ke dalam file **controller.js** tepat berada di bawah kode program pada **Gambar 4.3**. Kode program ini berfungsi untuk menjalankan fungsi update yang sudah dibuat pada model sebelumnya.

Cara kerjanya sendiri yaitu dengan memberikan nama fungsi **update** dan tambahkan sebagai object dari **exports** agar bisa dipanggil di router nantinya. Di dalam fungsi ini, lakukanlah error handling terlebih dahulu untuk memastikan data yang dikirim sudah benar. Jika sudah benar, jalankan **Task.update** dengan membawa parameter **req.params.id** sebagai id task yang akan diupdate, **new Task(req.body)** untuk membuat object task baru dengan isi sesuai data baru yang dikirimkan, dan juga parameter berupa fungsi setelah query update dijalankan.

```
51  exports.delete = function (req, res) {
52    Task.delete(req.params.id, function (err, task) {
53      if (err) res.send(err);
54      res.json({ error: false, message: "Task successfully deleted" });
55    });
56  };
57
```

Gambar 4.5

Kemudian yang terakhir yaitu menambahkan kode program diatas ke dalam file **controller.js** tepat berada di bawah kode program untuk update task pada **Gambar 4.4**. Kode program di atas berfungsi untuk menghapus task berdasarkan id dari task tersebut.

Cara kerjanya yaitu dengan memberikan nama fungsi **delete** dan tambahkan sebagai object dari **exports** agar bisa dipanggil di router nantinya. Di dalam fungsi ini, jalankanlah **Task.delete** untuk menjalankan fungsi **delete** yang sudah dibuat pada model sebelumnya.

Sampai langkah ini, model, controller dan koneksi ke database sudah bisa bekerja dengan baik. Selanjutnya, praktikan memerlukan router yang akan berguna untuk mengarahkan client ketika mengakses url dari API atau bisa juga disebut dengan endpoint dari API.

5. Membuat Router untuk Todolist API

Dalam pembuatan router untuk todolist API ini, praktikan bisa membuat file baru bernama **router.js** terlebih dahulu. Setelah itu, di dalam **router.js** ini, tuliskan kode program berikut ini.

```
JS router.js > ...
1  const express = require("express");
2  const router = express.Router();
3  const controller = require("./controller");
4
```

Gambar 5.1

Kode program di atas berguna untuk **mengimport package express JS** kemudian menggunakan **object router** yang sudah disediakan oleh express itu sendiri. Selanjutnya, dipanggil juga controller yang sudah dibuat sebelumnya untuk disimpan pada variabel bernama **controller**.

```
5  // Get All Task
6  router.get("/", controller.findAll);
7
```

Gambar 5.2

Selanjutnya, praktikan bisa menambahkan kode program seperti pada gambar di atas. Kode program di atas berarti, ketika ada client yang mengakses <http://localhost:3000/> dengan method **GET**, maka akan dijalankan **controller.findAll** yang sebelumnya sudah dibuat.

```
8  // Create New Task
9  router.post("/", controller.create);
10
```

Gambar 5.3

Selanjutnya, praktikan bisa menambahkan kode program pada gambar di atas tepat berada di bawah kode program **Gambar 5.2**. Kode program di atas berarti, ketika ada client yang mengakses <http://localhost:3000/> dengan method **POST** , maka akan dijalankan **controller.create** yang sebelumnya sudah dibuat.

```
11 // Get Task by Id
12 router.get("/:id", controller.findById);
13
```

Gambar 5.4

Selanjutnya, praktikan bisa menambahkan kode program pada gambar di atas tepat berada di bawah kode program **Gambar 5.3**. Kode program di atas berarti,

ketika ada client yang mengakses <http://localhost:3000/1> dengan method **GET**, maka akan dijalankan **controller.findById** yang sebelumnya sudah dibuat.

Angka 1 pada URL yang diakses oleh client, akan menjadi id dari task yang akan ditampilkan nantinya.

```
14 // Update a Task based on task id
15 router.put("/:id", controller.update);
16
```

Gambar 5.5

Selanjutnya, praktikan bisa menambahkan kode program pada gambar di atas tepat berada di bawah kode program **Gambar 5.4**. Kode program di atas berarti, ketika ada client yang mengakses <http://localhost:3000/1> dengan method **PUT**, maka akan dijalankan **controller.update** yang sebelumnya sudah dibuat.

Angka 1 pada URL yang diakses oleh client, akan menjadi id dari task yang akan di update nantinya.

```
17 // Delete a Task based on task id
18 router.delete("/:id", controller.delete);
19
```

Gambar 5.6

Selanjutnya, praktikan bisa menambahkan kode program pada gambar di atas tepat berada di bawah kode program **Gambar 5.5**. Kode program di atas berarti, ketika ada client yang mengakses <http://localhost:3000/1> dengan method **DELETE**, maka akan dijalankan **controller.delete** yang sebelumnya sudah dibuat.

Angka 1 pada URL yang diakses oleh client, akan menjadi id dari task yang akan dihapus nantinya.

Terakhir, ketika semua sudah dituliskan, jangan lupa untuk menyertakan module export di bagian akhir. Hal ini karena router ini nantinya akan dipanggil kembali di index.js yang akan dijalankan sebagai server.

6. Melakukan konfigurasi file index.js

Semua komponen sudah siap mulai dari koneksi ke database, model, controller, hingga router. Hanya saja, semua komponen itu belum berjalan di dalam server sehingga bisa menghasilkan web service yang mampu menyediakan API. Untuk itu, praktikan perlu membuat file baru bernama **index.js**. Kemudian tuliskan kode program seperti pada gambar di bawah ini untuk konfigurasi awalnya.

```
js index.js > ...
1  const express = require("express");
2  const bodyParser = require("body-parser");
3  const cors = require("cors");
4  const PORT = 3000;
5
```

Gambar 6.1

Pertama, praktikan harus mengimpor **package express JS** kemudian menyimpannya di variabel **express**. Selanjutnya, **package body-parser** disimpan di variabel **bodyParser**. Lalu **package cors** disimpan di variabel **cors**. Dan terakhir buatlah variabel **PORT** dengan nilai **3000** sebagai port dari server yang akan praktikan jalankan nantinya.

Nilai **PORT** ini akan berpengaruh terhadap port yang digunakan ketika mengakses API baik melalui browser maupun Postman nantinya.

```
6  // create express app
7  const app = express();
8
```

Gambar 6.2

Selanjutnya, praktikan perlu membuat sebuah variabel bernama **app** untuk menjadi object **express()**. Caranya yaitu dengan menambahkan kode program di atas tepat di bawah dari kode program pada **Gambar 6.1**.

```
9  // parse requests of content-type - application/x-www-form-urlencoded
10 app.use(bodyParser.urlencoded({ extended: true }));
11
```

Gambar 6.3

Setelah itu, praktikan harus menambahkan kode program di atas untuk mengijinkan request dalam bentuk **application/x-www-form-urlencoded**. Request dalam bentuk ini, bisa praktikan lakukan nantinya dengan bantuan tools **Postman**.

Kode program di atas bisa diletakkan tepat berada di bawah kode program **Gambar 6.2**.

```
12 // parse requests of content-type - application/json
13 app.use(bodyParser.json());
14
```

Gambar 6.4

Kemudian praktikan harus menambahkan kode program di atas untuk mengijinkan request dalam bentuk **application/json**. Request dalam bentuk ini, bisa praktikan lakukan nantinya dengan bantuan tools **Postman**.

Kode program di atas bisa diletakkan tepat berada di bawah kode program **Gambar 6.3**.

```
15 // using cors for csrf protection
16 app.use(cors());
17
```

Gambar 6.5

Setelah itu, praktikan juga harus menambahkan kode program di atas untuk menerapkan **cors** sebagai pengamanan **csrf protection**. Untuk kode program di atas ini, bisa diletakkan tepat berada di bawah kode program **Gambar 6.4**.

```
18 // Require task routes
19 const router = require("./router");
20
21 // using as middleware
22 app.use("", router);
23
```

Gambar 6.6

Selanjutnya praktikan harus menambahkan kode program di atas untuk memanggil **router** yang sudah dibuat sebelumnya. Kemudian, router tersebut akan dijalankan dengan menggunakan perintah pada baris 22 di **Gambar 6.6**.

Perintah tersebut memiliki dua parameter, yang pertama adalah **prefix** dan yang kedua adalah router yang mau dijalankan sehingga bisa diakses oleh client. Untuk prefix sendiri bisa diisi string kosong apabila tidak ingin ada tambahan prefix pada URL API nya.

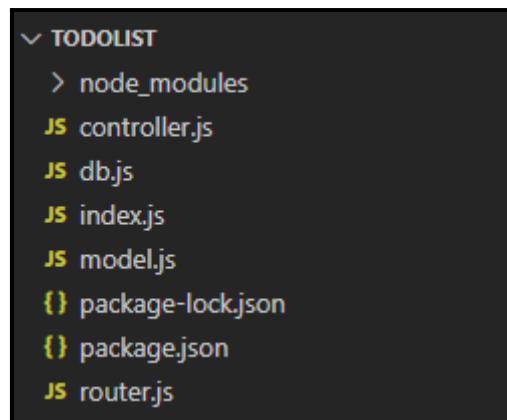
Kode program di atas bisa diletakkan tepat berada di bawah kode program **Gambar 6.5**.

```
24 // running server
25 app.listen(PORT, () => {
26   console.log(`server running on port ${PORT}`);
27 });
28
```

Gambar 6.7

Terakhir, praktikan harus menambahkan kode program di atas tepat berada di bawah kode program **Gambar 6.6**. Kode program ini berfungsi untuk menjalankan **Port** berdasarkan dengan konfigurasi port di awal untuk menjalankan **layanan web service** sehingga **API** dapat diakses oleh **client** ketika proses ini berhasil dijalankan.

Nah, ketika praktikan sudah selesai mengikuti setiap langkah praktikum dari 1 sampai 6, seharusnya struktur project express yang praktikan miliki kurang lebih akan seperti pada gambar di bawah ini.



Gambar 6.8

7. Melakukan Testing Endpoint API dengan Postman

Apabila semuanya dirasa sudah sesuai dan sudah selesai, sekarang saatnya untuk melakukan testing setiap endpoint yang sudah dibuat, dengan menggunakan **Postman**. Untuk mempermudah proses testingnya, praktikan bisa langsung mendownload template testing todolist app ini melalui link <http://bit.ly/3HKBZbb> atau bisa juga dengan melakukan scan pada QR Code di bawah ini.



Setelah download selesai, silahkan lakukan import ke dalam **Postman** lalu pastikan setiap tes berhasil dan tidak ada yang menampilkan pesan error di **Postman**.

Good Luck guys (^ v ^)

MODUL 6

DOCKER

Docker adalah platform open source yang menjalankan aplikasi dan membuat prosesnya lebih mudah untuk dikembangkan, didistribusikan. Aplikasi yang dibangun di docker dikemas dengan semua dependensi pendukung ke dalam bentuk standar yang disebut kontainer. Kontainer ini terus berjalan dengan cara yang terisolasi di atas kernel sistem operasi. Docker dapat dengan mudah berkoordinasi dengan instrumen pihak ketiga, yang membantu menyebarluaskan dan mengelola kontainer docker dengan mudah. Kontainer Docker dapat dengan mudah disebarluaskan ke lingkungan berbasis cloud

A. Tujuan Praktikum

- Menjelaskan tujuan dan kasus penggunaan untuk Docker
- Menjalankan, menghentikan, dan menghapus kontainer Docker
- Memahami penggunaan perintah di Docker

B. Alokasi Waktu

2 x 60 menit

C. Dasar Teori

Docker menyediakan fasilitas untuk mengotomatiskan aplikasi saat disebarluaskan ke dalam Kontainer. Dalam lingkungan Kontainer tempat aplikasi divirtualisasikan dan dijalankan, docker menambahkan lapisan tambahan mesin penyebarluasan di atasnya. Cara docker dirancang adalah dengan memberikan lingkungan yang cepat dan ringan di mana kode dapat dijalankan secara efisien dan terlebih lagi menyediakan fasilitas tambahan dari proses kerja yang mahir untuk mengambil kode dari komputer untuk pengujian sebelum produksi

Ada empat komponen internal utama docker, termasuk *Docker Client and Server*, *Docker Images*, *Docker Registries*, dan *Docker Containers*.

1. *Docker Client and Server*

Docker dapat dijelaskan sebagai aplikasi berbasis klien dan server. Server docker mendapatkan permintaan dari klien docker dan kemudian memprosesnya sesuai dengan itu. API RESTful (*Representational state transfer*) lengkap dan biner klien baris perintah dikirim oleh docker. Docker daemon/server dan docker client dapat

dijalankan pada mesin yang sama atau docker client lokal dapat dihubungkan dengan *remote server* atau *daemon*, yang berjalan pada mesin lain.

2. *Docker Images*

Ada dua metode untuk *build images*. Yang pertama adalah *build image* dengan menggunakan template *read-only*. Dasar dari setiap *image* adalah *image* dasar. *Image* sistem operasi pada dasarnya adalah *image* dasar, seperti Ubuntu 14.04 LTS, atau Fedora 20. *Image* sistem operasi membuat wadah dengan kemampuan untuk menjalankan OS sepenuhnya. *Image* dasar juga dapat dibuat dari awal. Aplikasi yang diperlukan dapat ditambahkan ke *image* dasar dengan memodifikasinya, tetapi perlu untuk *build image* baru. Proses *build image* baru disebut "melakukan perubahan". Metode kedua adalah membuat *file docker*. *File docker* berisi daftar instruksi ketika perintah "*Docker build*" dijalankan dari terminal bash, ia mengikuti semua instruksi yang diberikan dalam *file docker* dan *build image*. Ini adalah cara otomatis untuk *build image*.

3. *Docker Registries*

Image Docker ditempatkan di *docker registries*. Ini bekerja sesuai dengan *source code repository* dimana *image* dapat di-*push* atau di-*pull* dari satu sumber. Ada dua jenis registries, publik dan *private*. Docker Hub disebut registri publik di mana setiap orang dapat *pull image* yang tersedia dan *push image* mereka sendiri tanpa membuat *image* dari awal. *Image* dapat didistribusikan ke area tertentu (publik atau *private*) dengan menggunakan fitur hub docker.

4. *Docker Containers*

Image Docker membuat *docker container*. Kontainer menampung seluruh *kit* yang diperlukan untuk aplikasi, sehingga aplikasi dapat dijalankan dengan cara yang terisolasi. Sebagai contoh, misalkan ada *image* OS Ubuntu dengan SQL SERVER, ketika *image* ini dijalankan dengan perintah *docker run*, maka akan dibuat *container* dan SQL SERVER akan berjalan di OS Ubuntu.

D. Langkah Praktikum

Instal Docker Desktop

Anda dapat mengunduh *file* kebutuhan di <https://bit.ly/install-docker-desktop>

Untuk Pengguna **Windows** :

- **Instal WSL2 dari command prompt**

WSL2 (*Subsistem Windows* untuk Linux versi 2) adalah versi baru arsitektur yang memungkinkan Anda menggunakan Linux di atas Windows 10 secara

asli (menggunakan mesin virtual ringan) dan menggantikan WSL. Fitur ini menjalankan kernel Linux aktual di mesin virtual, yang meningkatkan kinerja dan kompatibilitas aplikasi dibandingkan versi sebelumnya sambil mempertahankan pengalaman yang sama seperti rilis pertama.

- **Buka *command prompt* sebagai *Administrator***
- **tulis dan lakukan** kode berikut

```
wsl --instal
```

- Jika hasil berikut menunjukkan, maka Anda selesai menginstal

```
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.

Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.

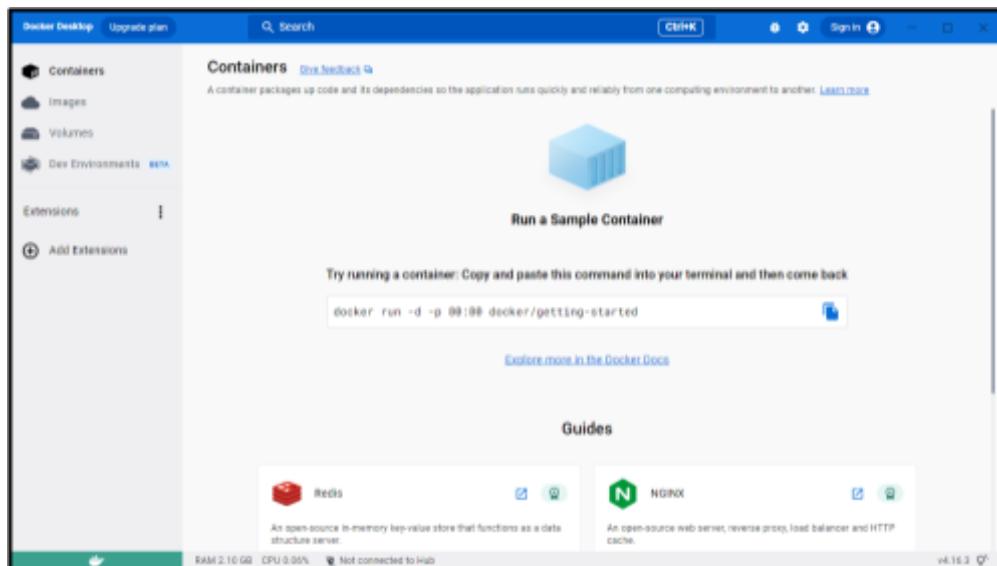
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.

Installing: Ubuntu
Ubuntu has been installed.

The requested operation is successful. Changes will not be
effective until the system is rebooted.
```

- **Cari dan Unduh Docker Desktop** dari situs web resmi di <https://www.docker.com/products/docker-desktop/> atau dari tautan yang sebelumnya diberikan
 - **Instal Docker Desktop**
 - **Restart PC Anda**
 - Jika Anda mendapatkan **error message** yang mengatakan **tidak dapat connect ke Docker**, lakukan langkah berikut:
 - **Uninstall Docker Desktop**
 - **Tulis** teks berikut ke **address manajer file** Anda
- ```
%appdata%\Docker
```
- Menghapus **folder Docker**
  - **Instal ulang** Docker Desktop dan **Buka** sebagai **administrator**

- Setelah Anda melakukan instalasi, ini akan menjadi tampilan aplikasi



Untuk pengguna **Mac** :

- **Unduh** Docker Desktop untuk Mac dari tautan sebelumnya atau dari situs web resmi.
- **Klik dua kali** file DMG, dan **drag-and-drop** Docker ke *folder* Aplikasi Anda.
- Anda perlu **authorize** instalasi dengan **password sistem** Anda.
- **Klik dua kali** Docker.app untuk memulai Docker.
- *Whale/paus* di bilah status Anda menunjukkan Docker sedang berjalan dan dapat diakses.
- Docker menyajikan beberapa informasi tentang menghilangkan tugas umum dan tautan ke dokumentasi.
- Anda dapat mengakses pengaturan dan opsi lain dari paus di bilah status. sebuah. Pilih Tentang Docker untuk memastikan Anda memiliki versi terbaru.

Untuk Pengguna **Linux** :

- Jika Anda menjalankan Linux, Anda perlu **menginstal** Docker **secara langsung**. Anda harus **masuk** sebagai **pengguna** dengan **sudo privileges**. Pertama, Anda perlu **memastikan** bahwa Anda memiliki **cURL** utilitas **command line**. Lakukan ini dengan membuka terminal dan mengetik:

```
$ which curl
```

Jika **cURL** tidak diinstal, perbarui manajer paket Anda dan **instal**, menggunakan:

```
$ sudo apt-get update
$ sudo apt-get install curl
```

- Sekarang setelah Anda memiliki cURL, Anda dapat menggunakannya untuk mendapatkan Docker *package* terbaru:

```
$ curl -fsSL https://get.docker.com/ | sh
```

- Tambahkan akun Anda ke grup docker.

```
sudo usermod -aG docker <your_username>
```

Langkah ini diperlukan untuk dapat menjalankan perintah Docker sebagai pengguna *non-root*. Anda harus keluar dan masuk kembali agar perubahan diterapkan.

- Sekarang Anda harus memiliki Docker! Verifikasi bahwa itu diinstal dengan menjalankan kontainer hello-world:

```
$ docker run hello-world
```

Hasil

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b901d36b6f2f: Pull complete
0a6ba66e537a: Pull complete
Digest:
sha256:8be990ef2aeb16dbcb9271ddfe2610fa6658d13f6dfb8bc72074cc1ca36966a
7
Status: Downloaded newer image for hello-world:latest
```

```
Hello from Docker.
```

Pesan ini menunjukkan bahwa instalasi Anda tampaknya berfungsi dengan benar.

- Jika Anda melihat pesan "*Cannot connect to the Docker daemon*", Anda mungkin perlu *restart* layanan Docker.

```
$ sudo service docker restart
```

## Contoh Docker

- Buka **Docker Desktop** dan **command line** atau **terminal** sebagai **administrator**.
- Tulis kode berikut ke **Terminal**:

```
docker run -d -p 80:80 docker/getting-started
```

Dari perintah tersebut, ada beberapa info lebih lanjut tentang mereka:

-d berarti **menjalankan kontainer dalam mode terpisah (di latar belakang)**

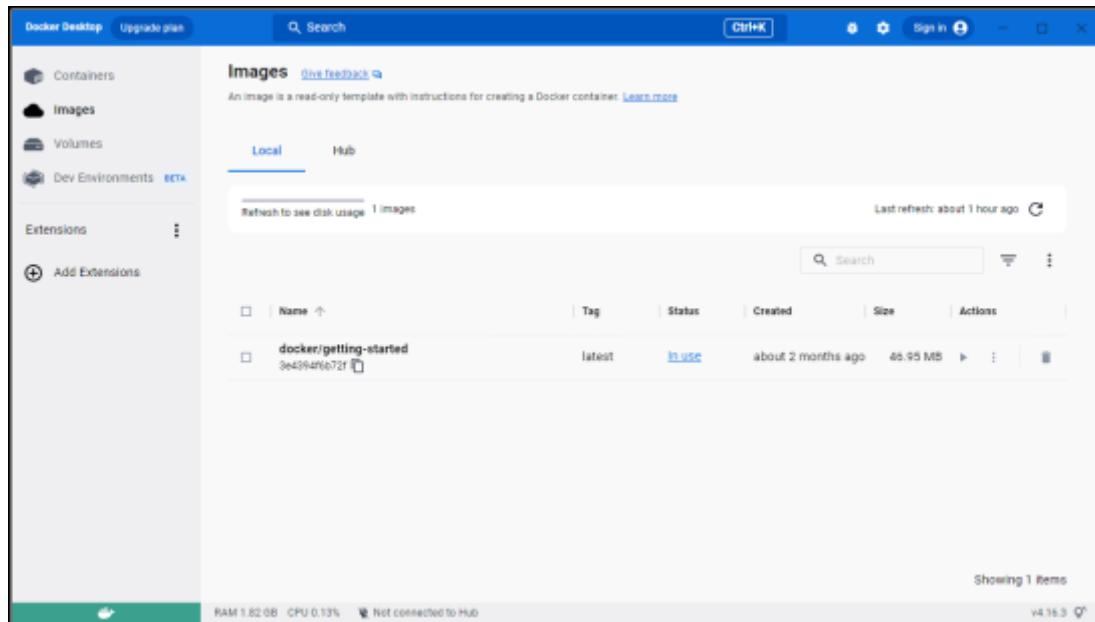
-p 80:80 berarti peta port 80 **dari host ke port 80 dalam kontainer**

docker/getting-started berarti **image yang akan digunakan**

Respons berikut akan ditampilkan:

```
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
c158987b0551: Pull complete
1e35f6679fab: Pull complete
cb9626c74200: Pull complete
b6334b6ace34: Pull complete
f1d1c9928c82: Pull complete
9b6f639ec6ea: Pull complete
ee68d3549ec8: Pull complete
33e0cbbb4673: Pull complete
4f7e34c2de10: Pull complete
Digest:
sha256:d79336f4812b6547a53e735480dde67f8f8f7071b414fb9297609ffb989abc1
Status: Downloaded newer image for docker/getting-started:latest
84e7c9cc80dbd3063151e6ecb3cf53f18f9544fb7046863a75fd25b2db823a23
```

- Buka bagian **Images** di Docker Desktop Anda dan gambar berikut adalah hasil dari pembuatan gambar docker



- Kita dapat mengakses apa yang baru saja kita buat dengan **membuka bagian Containers** dan **mengklik tautan Port(s)** yang statusnya berjalan seperti gambar berikut

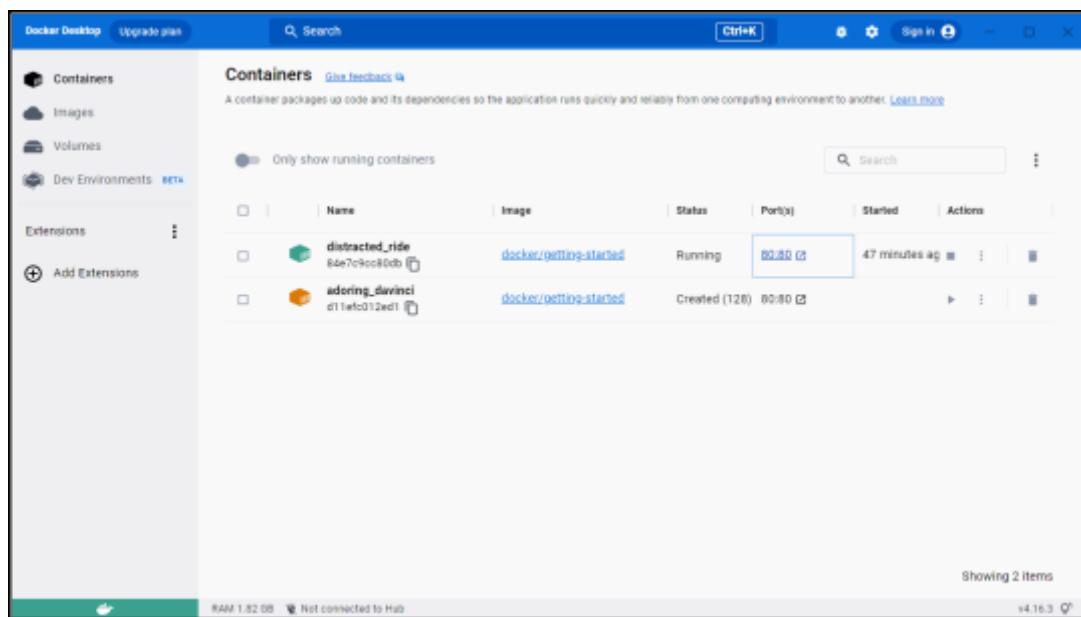
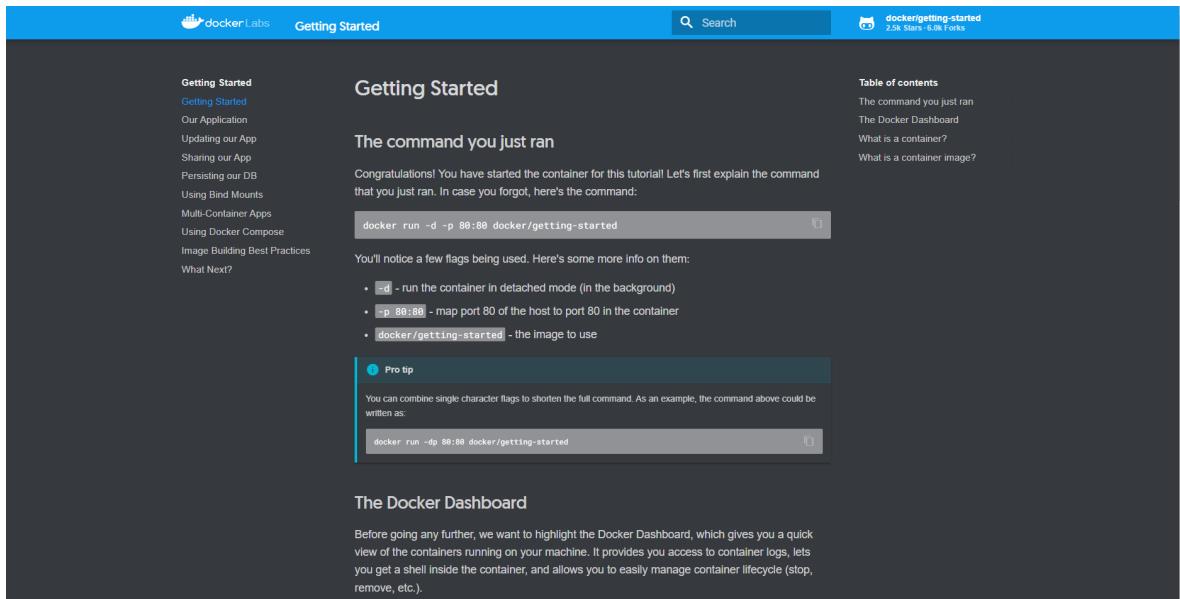


Image yang baru saja kita buat akan seperti gambar berikut



The screenshot shows a web browser displaying the Docker Getting Started tutorial. The URL is [https://docs.docker.com/get-started/01\\_start/](https://docs.docker.com/get-started/01_start/). The page has a dark theme with a blue header. On the left, there's a sidebar with a 'Getting Started' section containing links to 'Getting Started', 'Our Application', 'Updating our App', 'Sharing our App', 'Persisting our DB', 'Using Bind Mounts', 'Multi-Container Apps', 'Using Docker Compose', 'Image Building Best Practices', and 'What Next?'. The main content area has a title 'Getting Started' and a sub-section 'The command you just ran'. It shows the command `docker run -d -p 80:80 docker/getting-started`. Below the command, there's a 'Pro tip' section with the text: 'You can combine single character flags to shorten the full command. As an example, the command above could be written as: `docker run -dp 80:80 docker/getting-started`'.

## Hentikan dan Hapus Docker Container

Dalam pengembangan sebuah aplikasi, tentu saja akan ada *update* atau *debugging* yang berarti kita perlu meng-*update* aplikasi kita. Docker tidak dapat langsung diperbarui terutama ketika Anda mengubah *map port*. Itu sebabnya untuk melakukannya, Anda harus menghentikan *container* dan kemudian mengeluarkan *container*.

- Anda dapat memeriksa **daftar container** Anda menggunakan perintah berikut

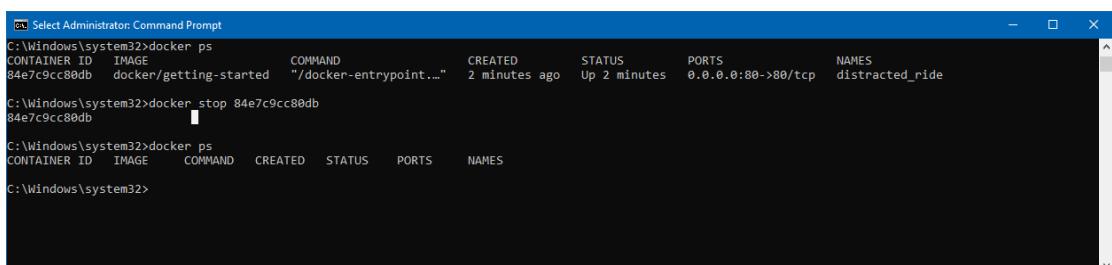
```
docker ps
```

- Untuk **menghentikan** kontainer, Anda perlu mengetahui **ID kontainer**. Kemudian Anda dapat menggunakan perintah berikut.

```
docker stop <the-container-id>
```

Tukar <the-container-id> dengan ID dari docker ps

Ini adalah contoh:



The screenshot shows a Windows Command Prompt window titled 'Select Administrator: Command Prompt'. The command `docker ps` is run, showing a single container with ID `84e7c9cc80db` and name `distracted_ride`. The port `0.0.0.0:80->80/tcp` is mapped. The command `docker stop 84e7c9cc80db` is then run to stop the container. Finally, `docker ps` is run again, showing that the container has been stopped.

```
C:\Windows\system32>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
84e7c9cc80db docker/getting-started "/docker-entrypoint..." 2 minutes ago Up 2 minutes 0.0.0.0:80->80/tcp distracted_ride

C:\Windows\system32>docker stop 84e7c9cc80db
84e7c9cc80db

C:\Windows\system32>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
C:\Windows\system32>
```

Anda dapat memeriksanya di Docker Desktop bahwa status kontainer berubah dari berjalan menjadi *exited* seperti gambar berikut

|  | Name                            | Image                                  | Status | Port(s) |
|--|---------------------------------|----------------------------------------|--------|---------|
|  | distracted_ride<br>84e7c9cc80db | <a href="#">docker/getting-started</a> | Exited | 80:80   |

- Kemudian Anda dapat **menghapusnya** dengan menggunakan perintah berikut

```
docker rm <the-container-id>
```

Jika Anda memeriksa bagian kontainer, Anda akan melihat bahwa kontainer sebelumnya dihapus. Meskipun Anda telah menghapus kontainer, Anda selalu dapat membuat kontainer lain untuk gambar docker yang sama dengan menggunakan perintah berikut

```
docker run -d -p 100:80 docker/getting-started
```

Kita dapat mengubah *port* dari sebelumnya 80:80 menjadi 100:80. Karena contoh ditetapkan pada *port* 80 di *container* dari *build image*-nya, kita hanya dapat mengubah *port* untuk orang luar yang *port* 100.

## List Command

Karena docker sendiri memiliki banyak fungsi dengan kegunaan dan karakteristiknya sendiri, kami akan mencantumkan beberapa perintah penting yang akan diperlukan untuk membuat layanan mikro. Anda dapat memeriksanya di <https://docs.docker.com/reference/>

## Docker Container Command List

| Command                 | Deskripsi                                                                                       |
|-------------------------|-------------------------------------------------------------------------------------------------|
| docker container attach | Melampirkan input standar, output, dan aliran kesalahan lokal ke kontainer yang sedang berjalan |
| docker container commit | Membuat <i>image</i> baru dari perubahan kontainer                                              |
| docker container cp     | Menyalin <i>file/folder</i> antara kontainer dan sistem file lokal                              |
| docker container create | Membuat kontainer baru                                                                          |
| docker container diff   | Memeriksa perubahan pada <i>file</i> atau <i>direktori</i> pada sistem <i>file</i> kontainer    |

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| docker container exec    | Menjalankan perintah dalam kontainer yang sedang berjalan                      |
| docker container export  | Mengekspor sistem <i>file</i> kontainer sebagai arsip tar                      |
| docker container inspect | Menampilkan informasi terperinci pada satu atau beberapa kontainer             |
| docker container kill    | Mematikan satu atau beberapa kontainer yang sedang berjalan                    |
| docker container logs    | Mengambil <i>log</i> kontainer                                                 |
| docker container ls      | Daftar kontainer                                                               |
| docker container pause   | Menjeda semua proses dalam satu atau beberapa kontainer                        |
| docker container port    | Mencantumkan pemetaan <i>port</i> atau pemetaan khusus untuk kontainer         |
| docker container prune   | Menghapus semua kontainer yang dihentikan                                      |
| docker container rename  | Mengganti nama kontainer                                                       |
| docker container restart | Memulai ulang satu atau beberapa kontainer                                     |
| docker container rm      | Menghapus satu atau beberapa kontainer                                         |
| docker container run     | Membuat dan menjalankan kontainer baru dari <i>image</i>                       |
| docker container start   | Memulai satu atau beberapa kontainer yang dihentikan                           |
| docker container stats   | Menampilkan <i>live stream</i> statistik penggunaan sumber daya kontainer      |
| docker container stop    | Menghentikan satu atau beberapa kontainer yang sedang berjalan                 |
| docker container top     | Menampilkan proses kontainer yang sedang berjalan                              |
| docker container unpause | Membatalkan semua proses dalam satu atau beberapa kontainer                    |
| docker container update  | Memperbarui konfigurasi satu atau beberapa kontainer                           |
| docker container wait    | Blokir hingga satu atau beberapa kontainer berhenti, lalu cetak kode keluarnya |

### Docker Image Command List

| Command              | Deskripsi                                                                                |
|----------------------|------------------------------------------------------------------------------------------|
| docker image build   | Membangun <i>image</i> dari <i>Dockerfile</i>                                            |
| docker image history | Memperlihatkan riwayat <i>image</i>                                                      |
| docker image import  | Impor konten dari <i>tarball</i> untuk membuat <i>filesystem image</i>                   |
| docker image inspect | Menampilkan informasi terperinci pada satu atau beberapa <i>image</i>                    |
| docker image load    | Memuat <i>image</i> dari arsip tar atau STDIN                                            |
| docker image ls      | Daftar <i>image</i>                                                                      |
| docker image prune   | Menghapus <i>image</i> yang tidak digunakan                                              |
| docker image pull    | Mengunduh <i>image</i> dari <i>registry</i>                                              |
| docker image push    | Mengunggah <i>image</i> ke <i>registry</i>                                               |
| docker image rm      | Menghapus satu atau beberapa <i>image</i>                                                |
| docker image save    | Simpan satu atau beberapa <i>image</i> ke arsip tar (dialirkan ke STDOUT secara default) |
| docker image tag     | Membuat tag TARGET_IMAGE yang merujuk ke SOURCE_IMAGE                                    |

### Docker Volume Command List

| Command               | Deskripsi                                                                 |
|-----------------------|---------------------------------------------------------------------------|
| docker volume create  | Membuat <i>volume</i>                                                     |
| docker volume inspect | Menampilkan informasi terperinci tentang satu atau beberapa <i>volume</i> |
| docker volume ls      | Daftar <i>volume</i>                                                      |
| docker volume prune   | Menghapus semua <i>volume</i> lokal yang tidak digunakan                  |
| docker volume rm      | Menghapus satu atau beberapa <i>volume</i>                                |
| docker volume update  | Memperbarui <i>volume</i> (hanya <i>volume</i> kluster)                   |

### Docker Network Command List

| Command                | Deskripsi                                 |
|------------------------|-------------------------------------------|
| docker network connect | Menyambungkan kontainer ke <i>network</i> |

|                                        |                                                                         |
|----------------------------------------|-------------------------------------------------------------------------|
| <code>docker network create</code>     | Membuat <i>network</i>                                                  |
| <code>docker network disconnect</code> | Memutuskan sambungan kontainer dari <i>network</i>                      |
| <code>docker network inspect</code>    | Menampilkan informasi terperinci pada satu atau beberapa <i>network</i> |
| <code>docker network ls</code>         | Daftar <i>network</i>                                                   |
| <code>docker network prune</code>      | Menghapus semua <i>network</i> yang tidak digunakan                     |
| <code>docker network rm</code>         | Menghapus satu atau beberapa <i>network</i>                             |

## **MODUL 7**

# **MENGENAL MICROSERVICES**

---

Materi praktikum microservice akan sangat banyak memanfaatkan pemahaman pada materi di modul sebelumnya (Docker). Oleh karena itu, sangat disarankan untuk kembali ke modul sebelumnya untuk memahami lagi bagaimana docker bekerja dan bagaimana pemanfaatannya guna mempermudah berjalannya praktikum.

Modul ini juga akan banyak membahas terkait teori tentang microservice. Sehingga, janganlah bosan dan perhatikanlah baik-baik materi yang diberikan. Karena microservice sejatinya hanyalah pendekatan pengembangan aplikasi, bukan solusi mutlak yang wajib atau harus diimplementasikan. Implementasinya bergantung pada masalah yang dihadapi. Jangan menutup mata untuk melihat kenyataan bahwa tidak semua pengembangan aplikasi membutuhkan arsitektur microservice.

Mari kita mulai materi ini dengan sebuah cerita. Sebelum adanya microservices, dunia mengenal pembangunan aplikasi menggunakan arsitektur *monolithic* (monolitik). Ya seperti namanya “mono” yang artinya satu, dimana istilah monolithic atau monolith ini merupakan metode pengembangan aplikasi di mana arsitekturnya meletakkan semua komponennya dalam satu kesatuan unit.

Sulit membayangkannya?

Harusnya kamu familiar dengan itu, tapi coba bayangkan ilustrasi berikut. Bayangkan sebuah sport center yang punya fasilitas olahraga lengkap. Semua fasilitas olahraga ada di situ, mulai dari renang, badminton, futsal, atletik, dan sebagainya ada di satu tempat. Menguntungkan di satu sisi karena pembangunan dan pengelolaan berada di satu bangunan yang sama. Namun di lain sisi, terlalu banyak fasilitas dalam satu bangunan akan membuat bangunan tersebut jadi terasa sempit dan tak lagi punya ruang yang cukup untuk hal lain.

Coba bayangkan gimana jadinya kalau kita ingin menambah satu fasilitas lagi di bangunan itu? Tentu sulit karena luas bangunannya terbatas. Jadi pilihannya adalah buat bangunan baru. Intinya, skalabilitas pada bentuk seperti itu kurang dinamis.

Itu juga berlaku pada arsitektur monolith di arsitektur IT. Penempatan semua komponen di satu lokasi yang sama membawa keunggulan dan kelemahannya sendiri. Keunggulannya yaitu membuat pengembangan dan pengelolaannya terpusat di satu bagian. Kelemahannya yaitu membuat skalabilitas jadi tidak fleksibel. Misal jika kita ingin meningkatkan kapasitas dari suatu komponen, maka kita harus scale (menyesuaikan kapasitas) seluruh unitnya.

Bahkan kalau saja satu komponen rusak, maka bagian komponen yang lainnya juga bisa terpengaruh dan bahkan bisa membuat seluruh sistem down. Sangat tidak lucu bukan karena bug di satu tempat membuat seluruh aplikasi menjadi tidak dapat digunakan. Itu akan mengurangi pengalaman pengguna (user experience). Kemudian datanglah microservice, sebuah arsitektur yang memisahkan dan memecah sistem aplikasi menjadi beberapa unit yang lebih kecil yang dikembangkan dan digunakan secara mandiri.

Karena setiap unit independen, maka satu sama lain tidak akan saling mempengaruhi. Kalau saja terjadi masalah, hanya satu unit saja yang terkena dampaknya, sementara unit yang lain tetap dapat digunakan. Juga ketika kita ingin melakukan peningkatan kapasitas, maka hanya unit yang memerlukan saja yang ditingkatkan, tidak perlu scale seluruh unit. Mari kita lanjutkan kelas ini dengan sebuah pertanyaan, apa sih microservices itu?

### **A. Tujuan Praktikum**

- Memahami apa Microservice dan perbedaannya dengan Monolithic
- Mengimplementasikan arsitektur microservice menggunakan Docker

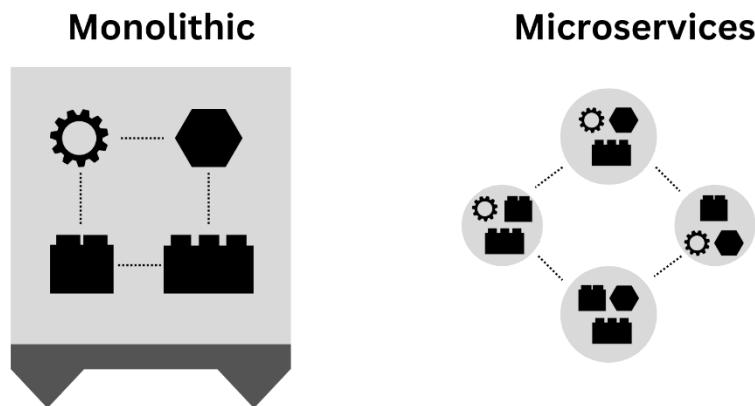
### **B. Alokasi Waktu**

2 x 60 menit

### **C. Dasar Teori**

#### 1. Apa itu Microservices

Arsitektur microservices (atau cukup microservices saja) merupakan sebuah pendekatan untuk membangun aplikasi sebagai serangkaian/sekumpulan “service” (layanan, unit, atau komponen) yang dapat dikembangkan, di-deploy, dan dikelola secara independen atau mandiri.



Sebuah unit atau service di dalam arsitektur microservices umumnya merupakan web service (layanan web) yang bertanggung jawab atas satu bagian domain (area logis) bisnis tertentu. Service tersebut kemudian digabungkan menjadi satu sistem utuh yang lebih besar dan masing-masing servicenya menyediakan fungsionalitas untuk domain bisnis tertentu dalam sistem tersebut.

Setiap service saling berinteraksi satu sama lain melalui perantara API, bisa saja berupa [REST](#) atau [gRPC](#). Namun mereka semua tidak punya pengetahuan tentang cara kerja internal masing-masing service. Nah interaksi yang harmonis antar service inilah yang disebut dengan arsitektur microservices.

Karakteristik microservices:

- Terdiri dari beberapa komponen atau service
- Mudah dikelola dan diuji
- Diatur berdasarkan fungsionalitas bisnis
- Infrastruktur yang terotomatisasi

Microservice bukan solusi untuk semua masalah yang ada. Namun seringkali mampu memecahkan sejumlah masalah terkait pengembangan perangkat lunak. Berikut keunggulannya:

- Menambah ketangkasan

Karena tim dapat dipecah menjadi tim-tim kecil untuk mengurus tiap service yang ada, membuat pengembangan aplikasi jadi lebih cepat dan waktunya rilisnya jadi lebih singkat. Selain itu tim juga dapat diberdayakan untuk mendeploy perubahan kode lebih sering.

- **Dinamis dalam skalabilitas**

Arsitektur microservices adalah praktik dari distributed system yang dikelola secara cluster. Secara alamiah membuat proses skalabilitas jadi lebih fleksibel dan dinamis. Contoh apabila suatu service tak sanggup menangani traffic yang masuk, maka instance baru yang identik dengan service tersebut dapat segera diluncurkan untuk membantu mengurangi beban.

- **Meningkatkan frekuensi rilis**

Penerapan bersama dengan CI/CD memungkinkan tim untuk bereksperimen dengan fitur baru dan melakukan rollback apabila ada yang tidak beres. Ini membuat developer menjadi lebih mudah dalam memperbarui kode dan mempercepat delivery fitur baru ke pengguna.

- **Fleksibilitas teknologi**

Setiap service bisa dikembangkan dengan tech stack yang berbeda karena tiap service dikembangkan dan di *deploy* secara independen. Misal, service A menggunakan NodeJs, Service B menggunakan Go, dan service C menggunakan python. Bisa saja dilakukan, tapi kami tidak merekomendasikan karena kompleksitas dapat meningkat karenanya.

- **Fokus pada kualitas**

Pada microservices, satu service hanya punya satu tanggung jawab terkait fungsionalitas bisnis tertentu. Itu artinya, tim yang mengurus service tersebut dapat berfokus pada satu tujuan sehingga dapat meningkatkan kualitas secara pasti.

- **Sangat andal**

Masalah yang terjadi pada satu service takkan merusak keseluruhan sistem aplikasi karena fungsionalitasnya telah terpisahkan. Selain itu tim juga termudahkan untuk mengisolasi dan memperbaiki error atau bug yang terjadi pada satu service sehingga kendala aplikasi dapat ditingkatkan.

Masih banyak kelebihan lainnya, tapi segitu dulu yang perlu kamu tahu. Kamu akan mengetahuinya lagi lebih banyak ketika kamu mengimplementasikannya langsung dan terbiasa menggunakan arsitektur microservices.

Buka matamu dan pahamilah bahwa pengembangan aplikasi menggunakan arsitektur microservice juga memiliki tantangan yang perlu kamu hadapi. Berikut beberapa tantangan yang mungkin akan kamu hadapi saat mengembangkan aplikasi menggunakan arsitektur *microservices*:

- **Pengembangan yang lebih kompleks**

Dua, tiga, atau sepuluh service, mungkin kamu masih bisa dengan mudah mengelolanya. Bagaimana jika ada seratus service, atau bahkan seribu, kamu harus sanggup mengelolanya. Karena apabila pengelolaannya tidak baik, kerumitan semacam itu malah menjadi *senjata makan tuan*. Yang awalnya ingin mempercepat proses pengembangan, kamu malah memperlambatnya dengan melahirkan performa yang buruk.

- Kepemilikan yang kurang jelas

Komunikasi dan koordinasi yang tepat amat diperlukan bagi semua tim sehingga setiap orang yang terlibat dapat dengan mudah mengerjakan tugasnya masing-masing.

- Biaya infrastruktur yang mungkin meningkat

Setiap service baru yang ditambahkan ke lingkungan production mungkin perlu serangkaian pengujian, proses *deployment, hosting, monitoring tools*, dan lain lain. Yang mana masing-masing hal tersebut bisa saja punya biaya tersendiri.

- Menambah kerumitan antar tim

- *Debugging* lebih sulit

- Perlunya respons yang cekatan terhadap suatu insiden

Penting untuk memiliki mekanisme yang cekatan dalam merespons bila suatu insiden terjadi pada *microservices*. Berikut contoh informasi yang dapat dicakup.

- Siapa yang telah menggunakan *service A*?

- Di mana *service A* di-*deploy*?

- Bagaimana *service A* di-*deploy*, kok bisa rusak?

- Siapa yang harus dihubungi jika terjadi kesalahan pada *service A*?

- Meningkatkan *latency*

Arsitektur *microservice* berskala besar pastinya punya *service* yang tersebar di beberapa *node*. Kalau ga mampu mengelola komunikasi antar *service*-nya, itu bisa meningkatkan *latency* dan akhirnya berefek pada penurunan pengalaman pengguna.

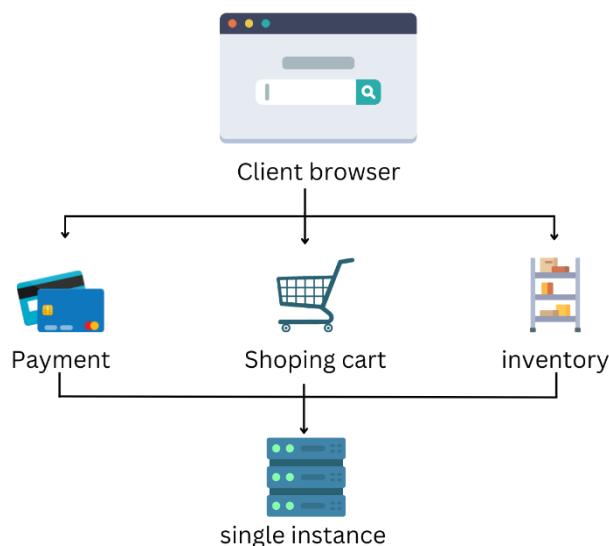
Teknologi yang digunakan dalam membangun arsitektur *microservices*:

- Container
- Docker
- Kubernetes
- API Gateway

- Messaging dan event streaming
  - Logging dan monitoring
  - CI/CD
2. Apa bedanya Microservice dengan Monolithic

Agar wawasan kamu lebih terbuka, kita coba bahas arsitektur monolithic dari berbagai sisi terlebih dahulu. Harapannya kamu bisa membandingkan dan menilai kedua arsitektur ini untuk kemudian memutuskan apakah pilihan yang kamu ambil sudah sesuai kebutuhan atau belum.

Arsitektur monolithic merupakan model tradisional dalam proses pengembangan aplikasi, di mana setiap komponen aplikasi dibangun sebagai unit terpadu. Kata "monolith" sering dikaitkan dengan sesuatu yang besar dan solid karena selaras dengan pengertian arsitektur monolithic yang kita maksud. Arsitektur monolithic adalah sistem komputer tunggal dengan satu codebase (basis kode) yang menyatukan semua urusan bisnis.



Masalah paling umum yang kerap kali terjadi adalah apabila ingin mengubah sedikit kode pada aplikasi, kita perlu mengakses keseluruhan code aplikasi dan mendeploy seisi aplikasi. Akibatnya proses pembaruan aplikasi menjadi terbatas dan memakan waktu.

Di lain sisi, arsitektur ini mungkin akan sangat terasa nyaman pada awal pengembangan aplikasi karena kode dan *deployment* mudah untuk dilakukan. Kenapa bisa demikian? Karena dalam sekali deploy semua komponen aplikasi

langsung tersedia ke pengguna. Namun, saat aplikasi berkembang jadi semakin kompleks, itu akan membuat pengelolaan kode jadi tidak semudah awalnya.

#### Keunggulan Monolithic

- Deployment lebih mudah

Hanya dengan satu directory atau satu executable file, deployment dapat berhasil. Mudah kan?

- Aplikasi mudah dikembangkan

Aplikasi yang dibangun dengan arsitektur monolith hanya perlu satu codebase, itu membuat pengembangan jadi lebih mudah dilakukan

- Performa yang mungkin tak jauh berbeda dengan microservices

- Pengujian lebih cepat

Karena hanya terdapat unit tunggal terpusat, pengujian dapat dilakukan lebih cepat dibanding menggunakan microservices.

- Proses debugging mudah

Sebab semua kode berada di satu tempat, itu membuat kita menjadi lebih mudah dalam menemukan masalah.

#### Tantangan Monolithic

- Pengembangan aplikasi lebih lambat

Coba bayangkan ketika kamu ingin mengubah sebuah bagian karena typo misalnya, yang perlu kamu lakukan yaitu mengubah *typo* itu lalu melakukan *deploy* keseluruhan aplikasi “lagi”. *Downtime* pasti ada, padahal bisa saja hanya karena kesalahan satu huruf menyebabkan seluruh aplikasi harus *di-deploy* ulang.

- Skalabilitas tidak fleksibel

- Keandalan yang tak pasti

Kalau terjadi kesalahan, maka *availability* seluruh aplikasi dapat terpengaruh. Hingga parahnya mungkin menyebabkan down seisi sistem.

- Hambatan dalam mengadopsi teknologi

Semisal ada dependency atau bahkan versi bahasa pemrograman yang *deprecated* (usang). Jelas, hal itu akan membuat gempar seisi perusahaan karena harus meningkatkan versinya. Mungkin tidak jadi masalah kalau hanya *update* versi. Namun, bagaimana jika terjadi *error* karena versi yang kita pakai saat ini sudah tertinggal jauh dengan versi terbaru sehingga harus memperbarui keseluruhan *codebase*. Repot sekali

Perbedaan paling terlihat antara arsitektur monolith dengan microservices adalah bahwa arsitektur monolithic merupakan sistem komputer tunggal dengan satu codebase (basis kode) yang menyatukan semua urusan bisnis, sedangkan microservices adalah suatu pendekatan pengembangan aplikasi yang memecah domain bisnis besar menjadi basis kode yang lebih kecil, terpisah, dan independen.

Perlu kamu ingat bahwa microservices itu sejatinya tidak mengurangi kompleksitas. Akan tetapi, ia membuat kompleksitas menjadi terlihat sehingga lebih mudah dikelola, yakni dengan memisahkan suatu tugas menjadi proses yang lebih kecil yang berfungsi secara independen satu sama lain dan berkontribusi pada keseluruhan sistem.

### 3. Langkah-langkah membangun Arsitektur *Microservices*

Kamu sudah paham tentang *microservices* dan berbagai karakteristiknya. Dari situ, kamu siap untuk memulai perjalanan membangun aplikasi dengan arsitektur microservices. Selanjutnya apa langkah yang perlu dilalui?

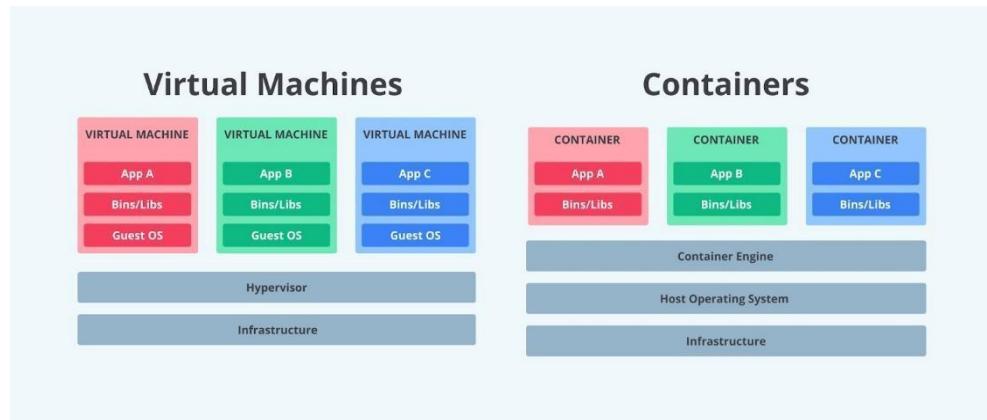
1. Mulai dari monolithic terlebih dahulu
2. Restrukturisasi tim
3. Pecah monolithic untuk membangun arsitektur microservices
  - a. Jaga komunikasi di antara service tetap sederhana dengan RESTful API
  - b. Bagi data ke dalam konteks/domain data
  - c. Persiapkan mitigasi kegagalan yang matang
  - d. Tekankan Monitoring demi kemudahan testing
  - e. Implementasi CI/CD untuk mengurangi risiko kegagalan saat deployment

**Praktikum ini hanya mengenalkan, bukan memberikan solusi pasti yang harus diikuti langkahnya untuk mengatasi semua masalah atau kasus yang dihadapi. Tidak ada cara yang pasti benar dalam hal ini. Satu-satunya cara yang bisa kamu lakukan adalah menganalisis aplikasi dan membuat keputusan sendiri.**

### 4. Apa itu *container*

*Container* merupakan bentuk dari virtualisasi sistem operasi. Bingung? Mari kita perjelas. Apa sistem operasi yang kamu gunakan di laptop atau komputer? Windows, macOS, atau Linux? Apapun OSnya, kamu bisa membuat *container* dengan jumlah satu, sepuluh, atau sebanyak apapun yang bisa ditangani komputermu. Ini bisa terjadi karena *container engine* (*platform* untuk membuat

dan mengelola *container*) akan memvirtualisasikan sistem operasi, kemudian membuat satu atau sejumlah container sesuai keinginan Kamu.



Di dalam sebuah *container*, umumnya berisi berbagai hal seperti executable file, binary, library, serta configuration file, tergantung kasus aplikasi yang kamu punya. Dibandingkan dengan pendekatan VM (virtual machines), container tidak berisi guest OS. Ini menjadikan *container* jauh lebih ringan dan *portable* ketimbang VM. Dalam kasus aplikasi yang lebih besar, beberapa container dapat digunakan sebagai satu atau beberapa cluster. *Cluster* ini dikelola oleh container orchestrator seperti Kubernetes (akan kita bahas di modul berikutnya).

#### Keunggulan *Container*

- Lebih hemat sumber daya
- Meningkatkan portabilitas
- Operasional yang konsisten
- Lebih ringan dan cepat
- Ekosistem yang komprehensif

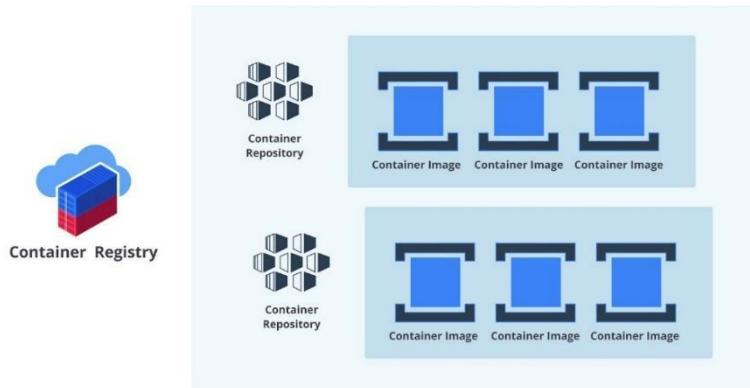
#### 5. Apa itu *container registry*

Dibandingkan dengan pendekatan VM (virtual machines), container tidak berisi guest OS. Ini menjadikan container jauh lebih ringan dan *portable* ketimbang VM. Dalam kasus aplikasi yang lebih besar, beberapa container dapat digunakan sebagai satu atau beberapa *cluster*. *Cluster* ini dikelola oleh container orchestrator seperti Kubernetes (akan kita bahas di beberapa modul berikutnya).

Sebuah *container registry* pada dasarnya bertindak sebagai tempat bagi Developer untuk menyimpan dan mendistribusikan *container image* melalui

proses *uploading/pushing* (pengunggahan) ke *container registry* dan *downloading/pulling* (pengunduhan) ke sistem lain, contohnya Kubernetes *cluster*. Setelah image terunduh, aplikasi di dalamnya dapat dijalankan di sistem tersebut.

Namun, sebenarnya lokasi fisik tersimpannya *container image* adalah di *repository*. Setiap *repository* menyimpan sekumpulan *image* (yang saling terkait) dengan nama yang sama.



Setiap *container image* dalam sebuah repository pada umumnya merepresentasikan versi yang berbeda dari deployment aplikasi yang sama. Misalnya, pada container registry populer seperti Docker Hub, NGINX adalah sebuah nama repository berisi sejumlah versi yang berbeda dari Docker image untuk instalasi web server NGINX. Umumnya, setiap perbedaan versi tersebut diidentifikasi dengan sebuah tag, misalnya `nginx:latest` atau `nginx:1.23.1-alpine-perl`.

Terdapat dua tipe *container registry* jika dilihat dari sisi *deployment*-nya. Yaitu *Self-hosted* dan *hosted remotely*. *Self-hosted* digunakan ketika kamu memiliki kebutuhan tentang keamanan, peraturan, atau juga ingin punya *access control* dan privasi yang ketat terhadap *image*. *Hosted remotely* digunakan bila kamu tidak mau repot dengan penginstalan, pengelolaan, dan pengonfigurasian server secara mandiri.

Opsi pilihan *self-hosted*

- Docker registry
- Harbor
- Sonatype Nexus Repository

- Red hat Quay

Opsi pilihan Hosted remotely

- Amazon ECR
- Docker Hub
- Github Packages

## D. Langkah Praktikum

Berikut merupakan tahapan proses yang akan dilaksanakan dalam praktikum ini.

1. Membuat dan menjalankan image sebagai sebuah container.
2. Deploy Todo App ke Docker disertai sebuah database.
3. Menjalankan aplikasi menggunakan Docker Compose.
4. Membuat akun Docker Hub.
5. Membuat *repository* di Docker Hub.
6. Menggunakan container image ke Docker Hub.

Kamu sudah paham docker kan? Jadi seharusnya kamu bisa menyelesaikan modul ini tanpa hambatan. Jika belum begitu paham penggunaannya, tenang saja, materi di modul ini akan memperdalam pemahamanmu. Baru dengar docker compose? Kita bahas nanti.

Mari kita mulai.

**Catatan:** OS manapun tidak menjadi masalah atau hambatan dalam mengerjakan modul kelas ini. Hanya saja, linux sangat direkomendasikan apabila kamu memiliki device laptop atau pc dengan spesifikasi rendah. Lakukan saja dual boot (>40gb storage untuk linux sangat direkomendasikan)

**Tutorial:** <https://itsfoss.com/install-ubuntu-1404-dual-boot-mode-windows-8-81-uefi/>

1. Buka terminal/CMD, lalu **clone** repo ke directory yang diinginkan.

```
git clone https://github.com/1pizzaifupn/microservices.git
```

Hasilnya:

```

Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023
$ cd tutorials/

Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials
$ git clone https://github.com/1pizzaifupn/microservices.git
Cloning into 'microservices'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (52/52), done.
remote: Total 57 (delta 3), reused 57 (delta 3), pack-reused 0
Receiving objects: 100% (57/57), 1.78 MiB | 1.91 MiB/s, done.
Resolving deltas: 100% (3/3), done.

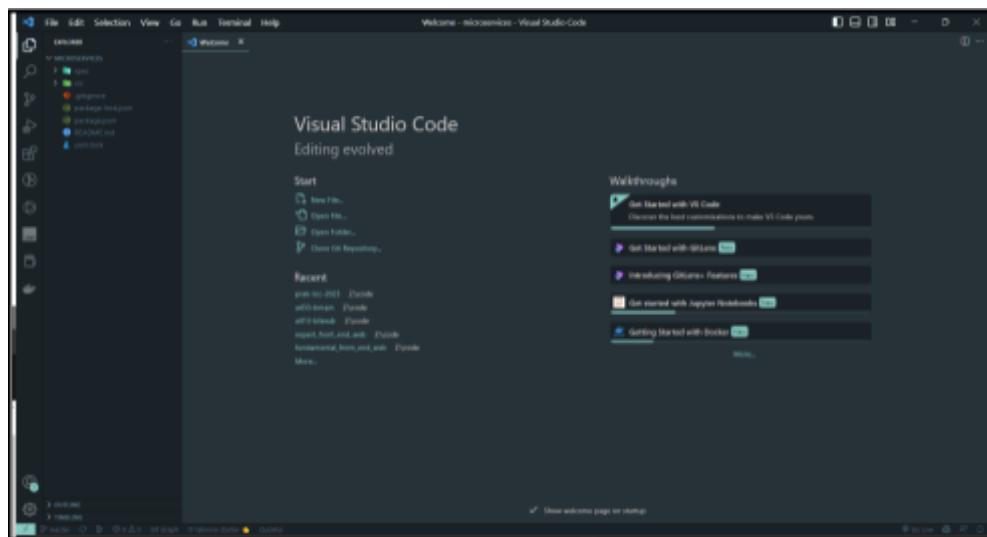
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials
$ █

```

*Catatan: git harus sudah terinstall di komputermu untuk menggunakan perintah di atas.*

*Jika belum ada, install terlebih dahulu dari <https://git-scm.com/downloads>*

2. Buka code editormu (dalam hal ini contohnya adalah Visual Studio Code), open folder pilih microservices yang baru saja di-clone, open.



3. Buatlah file bernama **Dockerfile** (tanpa ekstensi apapun) yang akan digunakan untuk melakukan build image. Salin *code* berikut ke dalam Dockerfile.

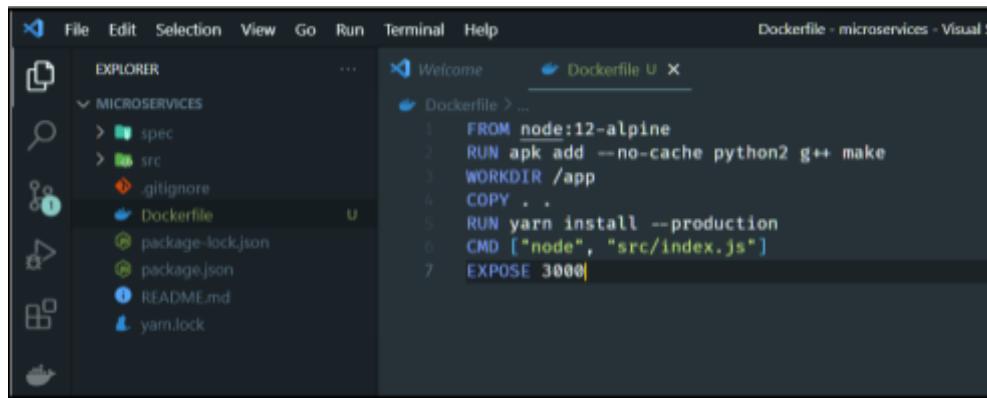
```

FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY .
RUN yarn install --production
CMD ["node", "src/index.js"]

```

```
EXPOSE 3000
```

Hasilnya:



```
FROM node:12-alpine
RUN apk add --no-cache python2 g++ make
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

Penjelasan:

- **FROM node:12-alpine:** Memberitahu Docker untuk mengunduh/mengambil image bernama node dari Docker Hub yang memiliki tag 12-alpine. Kalau sudah pernah diunduh, Docker mengambil dari lokal.  
Catatan: *Tag di Docker Hub berperan layaknya version.*
- **RUN apk add --no-cache python2 g++ make:** Memasang beberapa package, diantaranya adalah python2, g++, dan make tanpa menyimpan cache.
- **WORKDIR /app:** Kita membuat sebuah directory baru bernama **app** di dalam container dan menjadikannya sebagai *working directory*. Instruksi apapun setelah baris ini akan dijalankan pada directory /app ini.
- **COPY . . :** perhatikan bahwa ada dua buah titik di baris kode ini. Tanda titik pertama menunjukkan source (sumber), sementara titik kedua menandakan destination. Jika diartikan, kita menyalin semua yang ada di directory saat ini (misal Z:\code\prak-tcc-2023\tutorials\microservices yang berisi dua folder: spec dan src; serta tiga berkas: Dockerfile, package.json, dan yarn.lock) ke container working directory (yakni /app seperti yang kita definisikan pada baris sebelumnya).
- **RUN yarn install --production :** menjalankan perintah **RUN yarn install --production** untuk menginstall dependencies yang dibutuhkan oleh aplikasi seperti yang tertera pada berkas package.json. Selain bagian devDependencies.

- **CMD ["node", "src/index.js"]** : mengeksekusi perintah untuk menjalankan aplikasi.
  - **EXPOSE 3000**: src/index.js didefinisikan menjalankan aplikasi pada port 3000. Ingat bagaimana port forwarding di docker? Agar aplikasi dapat berkomunikasi dengan dunia luar, maka kita perlu mengekspos port yang digunakan oleh container tersebut. Dalam hal ini port 3000.
4. Simpan perubahan yang ada dengan **CTRL + S**
  5. Buka terminal di vscode
  6. Buatlah sebuah Docker image menggunakan Dockerfile yang tadi dibuat dengan nama **todo-app** dan tag **v1**.

```
docker build -t todo-app:v1 .
```

*Catatan: perhatikan bahwa terdapat titik di akhir perintah. Itu menandakan source/path Dockerfile ada di current working directory*

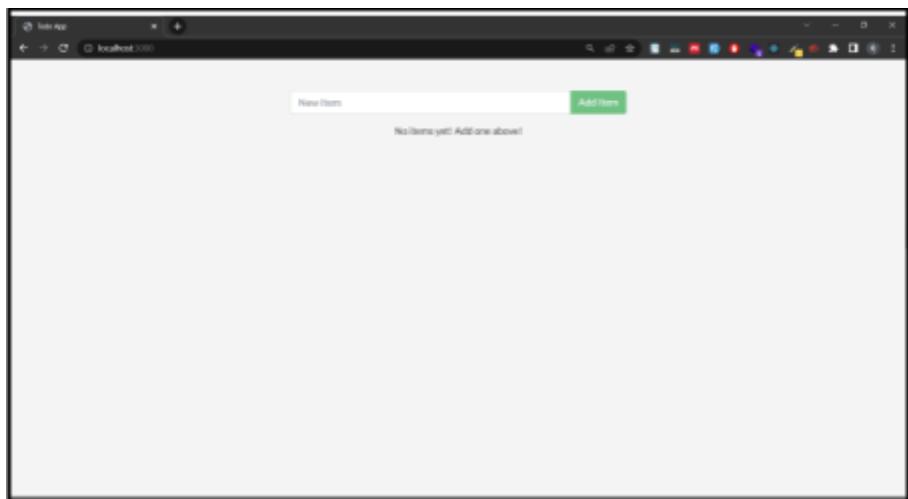
7. Sekarang cobalah untuk menjalankan sebuah *container* dari *image* yang sudah dibuat tersebut

```
docker run -dp 3000:3000 --name todo-app todo-app:v1
```

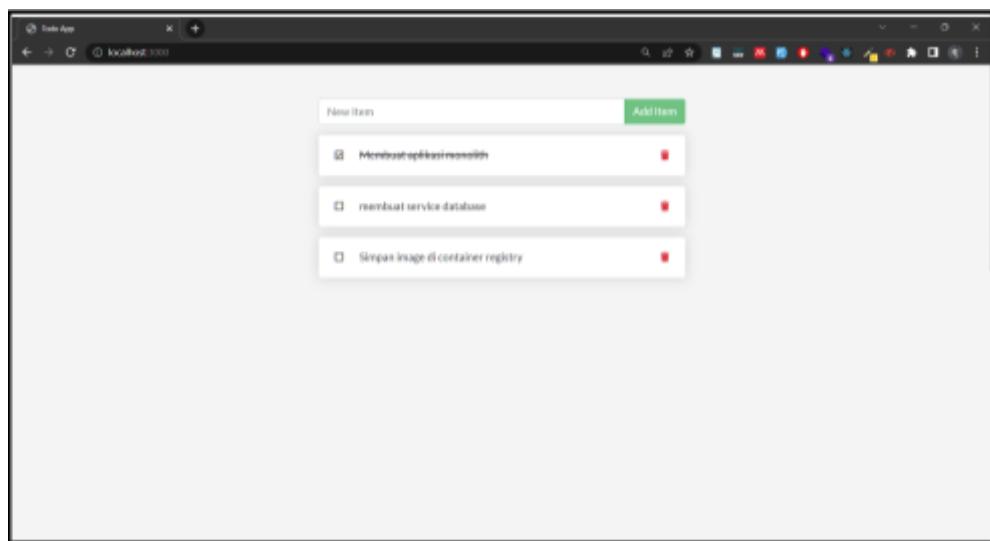
Perintah diatas menjalankan container baru dalam mode **detach** (berjalan di background, parameter d), lalu membuat **port mapping** (parameter p) antara port 3000 di host dan port 3000 di *container*, lalu memberikan nama todo-app, dan menggunakan image todo-app:v1.

Port mapping menggunakan **Docker\_host:Container\_port**, kalau kamu mengubah port mapping di atas menjadi 5000:3000, berarti kamu tidak bisa menggunakan localhost:3000 di browser untuk mengakses aplikasimu, melainkan localhost:5000.

8. Buka alamat <http://localhost:3000> di browser. Aplikasi Todo App akan tampil di sana.

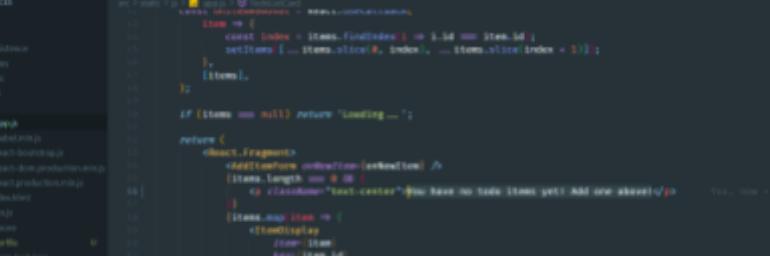


9. Silahkan tambahkan beberapa item dan coba eksplorasi fitur yang ada.



Dengan begini artinya frontend telah dapat berfungsi dengan baik dan dapat menyimpan data di backend.

10. Aplikasi sudah berjalan dengan baik nih, wah tapi ada typo. Mari kita modifikasi sedikit aplikasi kita. Coba buka berkas **src/static/js/app.js**.
11. Setelah itu, ganti teks "**No items yet! Add one above!**" menjadi "**You have no todo items yet! Add one above!**".



```
 if (items === null) return "Loading ...";

 return (
 <React.Fragment>
 <div style={gridStyle}>
 {items.map(item =>
 <Card key={item.id}>
 <CardImage alt={item.name}>

 </CardImage>
 <CardBody>
 <CardTitle>{item.name}</CardTitle>
 <CardText>{item.description}</CardText>
 <CardText>${item.price}</CardText>
 <CardText>{item.rating}</CardText>
 </CardBody>
 </Card>
)}
 </div>
 </React.Fragment>
);
}

export default Dashboard;
```

Jangan lupa simpan. CTRL + S

12. Lanjut pada terminal, build image untuk menerapkan modifikasi yang baru saja dilakukan dengan nama **todo-app** dan tag **v2**.

```
docker build -t todo-app:v2 .
```

13. Lalu sekarang kita jalankan container baru dengan nama yang identik. Karena identik, container sebelumnya harus dihentikan dan dihapus terlebih dahulu agar tidak terjadi kegagalan karena port yang digunakan sama persis.

```
docker rm -f todo-app
```

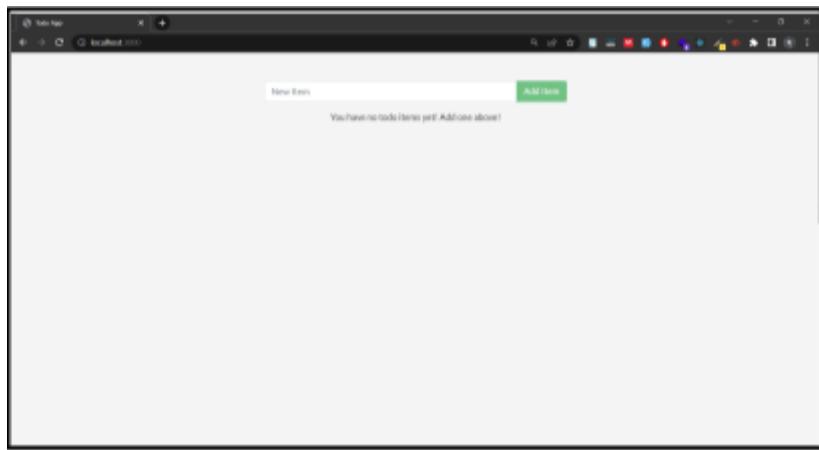
rm berarti menghapus (remove)

-f berarti force atau paksa lakukan perintah terhadap container yang sedang berjalan

14. Jalankan lagi container dengan perintah berikut

```
docker run -dp 3000:3000 --name todo-app todo-app:v2
```

15. Buka kembali alamat <http://localhost:3000> di web browser. Perubahan yang kamu lakukan akan tampil di sana.



16. Nice, kamu sudah berhasil memperbarui aplikasi. Tapi tunggu sebentar, kamu sadarkah ada yang terjadi? Semua item catatan yang sudah dibuat sebelumnya menghilang! Kita tidak ingin ini terjadi. Ini terjadi karena kita sudah menghapus container sebelumnya dan menjalankan container baru dengan image yang baru. Maka dari itu, ketika diakses, aplikasi Todo App ini seperti bayi baru lahir.

Defaultnya aplikasi yang kita miliki sekarang menyimpan data di sebuah SQLite Database pada `/etc/todos/todo.db` dalam filesystem milik container (silakan periksa berkas `src/persistence/sqlite.js`

SQLite adalah database relasional yang menyimpan semua data ke dalam satu berkas. SQLite acapkali digunakan untuk development, demo, atau aplikasi skala kecil sehingga cocok untuk Todo App kita saat ini. Dengan membuat volume dan melampirkannya (mount) ke container, SQLite akan dapat menyimpan data ke volume tersebut sehingga data takkan hilang meski container dihapus. Mari kita terapkan pemahaman kita tentang volume di docker agar item pada aplikasi tetap ada meski container sebelumnya telah dihapus.

17. Buat volume dengan nama todo-db

```
docker volume create todo-db
```

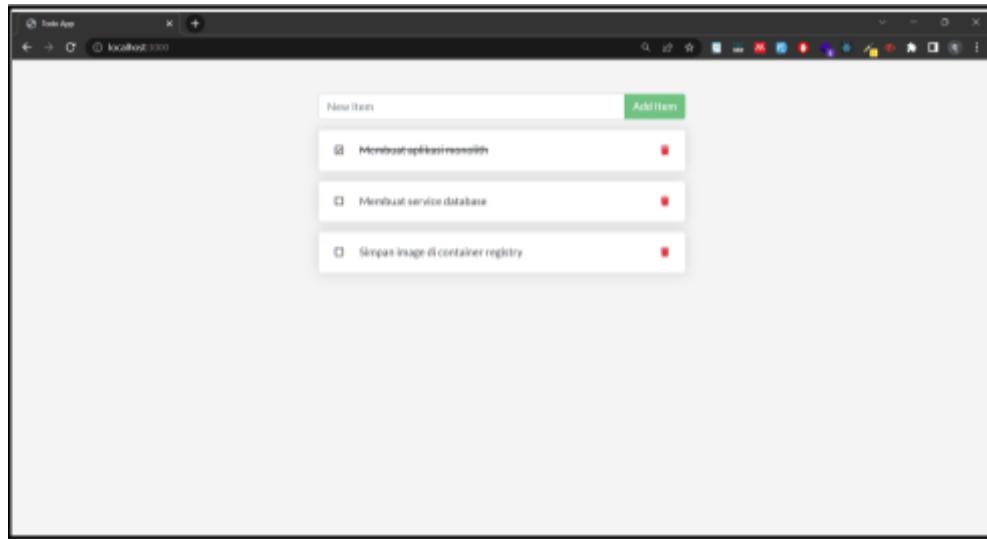
18. Hentikan dan hapus container todo-app sebelumnya karena kita akan gunakan volume ke dalamnya

```
docker rm -f todo-app
```

19. Jalankan container baru untuk todo app, namun sekarang tambahkan -v untuk menyertakan volume yang akan digunakan

```
docker run -dp 3000:3000 --name todo-app -v todo-db:/etc/todos
todo-app:v2
```

20. Akses kembali, tambahkan beberapa item sesuka hati kamu



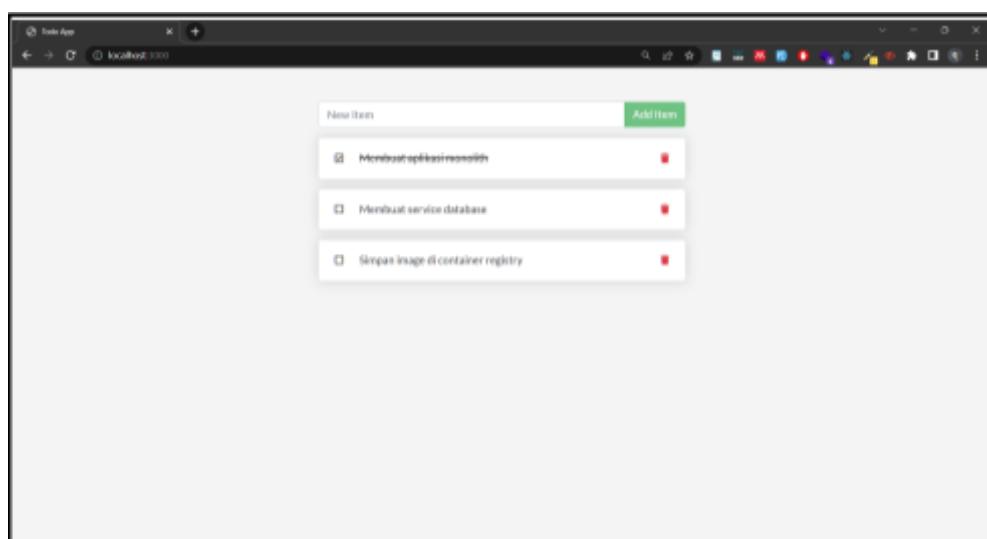
21. Mari kita uji dengan hapus container untuk memastikan apakah item yang sudah diinputkan benar tetap ada

```
docker rm -f todo-app
```

22. Jalankan lagi container baru seperti sebelumnya

```
docker run -dp 3000:3000 --name todo-app -v todo-db:/etc/todos
todo-app:v2
```

23. Akses kembali Todo App, kamu akan melihat bahwa item-item yang telah dibuat sebelumnya masih ada di tempat yang sama.



24. Bagus, aplikasi yang kita bangun sudah dapat berjalan dengan baik. Namun hingga saat ini aplikasi masih belum sepenuhnya menerapkan arsitektur microservices. Berikutnya kita akan buat database service berupa service mysql untuk melengkapi arsitektur aplikasi kita.
25. SQLite memang salah satu opsi yang bagus untuk menyimpan data relational untuk apps kita saat ini, selain itu kita juga harus belajar database engine lain yang umumnya dipakai untuk aplikasi skala besar. Contohnya MySQL. Kita akan perbarui arsitektur Todo App kita menjadi:



Dua container akan dihubungkan melalui network yang sama agar bisa saling berkomunikasi satu sama lain. Kita akan lakukan di langkah-langkah selanjutnya.

Sebelum memulai, hapus terlebih dahulu container todo-app.

```
docker rm -f todo-app
```

26. Buat custom network bertipe bridge dengan nama todo-app

```
docker network create todo-app
```

27. Cek network yang ada apakah tipe network yang kita buat bertipe bridge

```
docker network ls
```

Hasilnya:

```

Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker network create todo-app
4f838aa3ed459b8a2677e47b86a9db1db2f48cbf98b4a5d55652b150fe41b719

Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker network ls
NETWORK ID NAME DRIVER SCOPE
2507ebb453e8 bridge bridge local
f1993593841b host host local
9293c64117e0 none null local
4f838aa3ed45 todo-app bridge local

```

28. Jalankan sebuah container baru untuk MySql bernama **mysql-db**, lampirkan pula network yang sudah dibuat (tambahkan juga alias padanya bernama **mysql**), serta buat volume baru bernama **todo-mysq-data** dan pasangkan langsung ke directory **/var/lib/mysql** di dalam container

Opsi untuk beberapa os:

Linux / gitbash

```

docker run -d \
 --name mysql-db \
 --network todo-app --network-alias mysql \
 -v todo-mysql-data:/var/lib/mysql \
 -e MYSQL_ROOT_PASSWORD=tcc2023 \
 -e MYSQL_DATABASE=todo-db \
 mysql:5.7

```

macOS dengan processor berbasis ARM (i.e. apple silicon chip)

```

docker run -d \
 --name mysql-db \
 --network todo-app --network-alias mysql \
 --platform "linux/amd64" \
 -v todo-mysql-data:/var/lib/mysql \
 -e MYSQL_ROOT_PASSWORD=tcc2023 \
 -e MYSQL_DATABASE=todo-db \
 mysql:5.7

```

Windows

```

docker run -d \

```

```
--name mysql-db
--network todo-app --network-alias mysql
-v todo-mysql-data:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=tcc2023
-e MYSQL_DATABASE=todo-db
mysql:5.7
```

-e pada perintah di atas menandai pendefinisian *environment variable* diantaranya berupa root password dan nama database yang dibuat. Selain itu kita juga mendefinisikan penggunaan *image* mysql versi 5.7 yang diambil dari Docker Hub.

29. Kalau *container* sudah sukses dibuat, lanjutkan dengan menguji apakah database sudah berjalan dengan normal dengan masuk ke dalamnya menggunakan exec

```
docker exec -it mysql-db mysql -u root -p
```

30. Jalankan perintah di bawah untuk menampilkan daftar database yang tersedia, pastikan terdapat database bernama todo-db yang sudah dibuat sebelumnya.

```
SHOW DATABASES;
```

Hasilnya:

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| todo-db |
+-----+
5 rows in set (0.01 sec)
```

31. Kalau aman, *Exit*
32. Perhatikan bahwa MySQL database sudah aktif. Lalu bagaimana agar ia dapat diakses oleh Todo App? Sebelumnya kita menggunakan --network-alias mysql saat membuat MySQL container, nah karena todo app dengan MySQL berada

di dalam network yang sama, keduanya bisa berkomunikasi menggunakan IP address container, nama container, atau bahkan nama network. Langsung kita uji coba.

33. Konfigurasi yang perlu diterapkan pada environment variable agar todo app dapat terkoneksi ke MySQL diantaranya sebagai berikut (silakan cek berkas src/persistence/mysql.js):

- **MYSQL\_HOST**: hostname di mana MySQL server berjalan (dalam kasus kita, bisa IP address dari mysql-db container, nama container mysql-db, atau nama network/network alias).
- **MYSQL\_USER**: username yang digunakan untuk terhubung ke MySQL database.
- **MYSQL\_PASSWORD**: password yang digunakan untuk terhubung ke MySQL database.
- **MYSQL\_DB**: nama database yang akan dihubungkan.

34. Jalankan container untuk Todo App

Linux

```
docker run -dp 3000:3000 --name todo-app \
 -w /app -v "$(pwd):/app" \
 --network todo-app \
 -e MYSQL_HOST=mysql \
 -e MYSQL_USER=root \
 -e MYSQL_PASSWORD=tcc2023 \
 -e MYSQL_DB=todo-db \
 node:12-alpine \
 sh -c "yarn install && yarn run dev"
```

Windows

```
docker run -dp 3000:3000 --name todo-app \
 -w /app -v "$(pwd):/app" \
 --network todo-app \
 -e MYSQL_HOST=mysql \
 -e MYSQL_USER=root \
 -e MYSQL_PASSWORD=tcc2023 \
 -e MYSQL_DB=todo-db \
 node:12-alpine \
 sh -c "yarn install && yarn run dev"
```

Berikut penjelasannya:

- **docker run -dp 3000:3000 --name todo-app** : *Container* dijalankan dengan nama **todo-app** dengan mengekspos **port 3000** dengan mode **detach**
- **-w /app -v "\$(pwd):/app"**: menentukan *working directory* di dalam container adalah `/app` lalu melampirkan penyimpanan dengan tipe bind mount dan memasangkan **working directory saat ini di host** ke directory `/app` di container.
- **--network todo-app**: menggunakan *bridge network* bernama `todo-app`
- **-e MYSQL\_HOST=mysql**: set *environment variable* untuk hostname Mysql dengan nilai `mysql`
- **-e MYSQL\_USER=root**: set env variable untuk username mysql dengan nilai `root`
- **-e MYSQL\_PASSWORD=tcc2023**: set env variable untuk password mysql dengan nilai `tcc2023`
- **-e MYSQL\_DB=todo-db**: set nama database dengan nilai `todo-db`
- **node:12-alpine**: penentuan image yang digunakan bernama `node` dengan tag `12-alpine` dari docker hub
- **sh -c "yarn install && yarn run dev"**: jalankan perintah pada *container* ketika container berhasil diluncurkan.

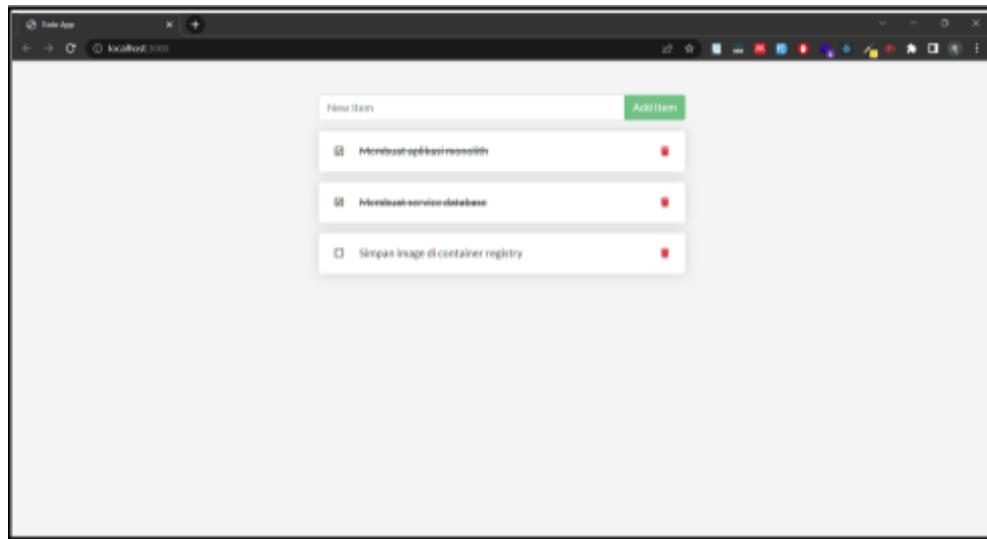
35. Periksa log dari *container* untuk melihat bagaimana aplikasi Todo App berjalan dan membuktikan apakah ia berhasil menggunakan MySQL database

```
docker logs todo-app
```

```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker logs todo-app
yarn install v1.22.18
warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not to mix package managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear this warning, remove package-lock.json.
[1/4] Resolving packages...
warning Resolution field "ansi-regex@5.0.1" is incompatible with requested version "ansi-regex@^2.0.0"
success Already up-to-date.
Done in 0.54s.
yarn run v1.22.18
$ nodemon src/index.js
[nodemon] 2.0.13
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
Waiting for mysql:3306.
Connected!
Connected to mysql db at host mysql
Listening on port 3000
```

Dari log di atas kita tahu bahwa apps terhubung pada network dengan alias mysql pada port 3306 dan menjalankan Todo App via port 3000 di dalam container.

36. Buka lagi Todo App via <http://localhost:3000/> dan tambahkan beberapa item.



37. Coba masuk ke dalam container mysql-db untuk memeriksa apakah item yang tadi telah ditulis telah tersimpan di MySQL

```
docker exec -it mysql-db mysql -p todo-db
```

Hasilnya

```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker exec -it mysql-db mysql -p todo-db
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.41 MySQL Community Server (GPL)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> ^
```

38. Jalankan query berikut ketika di dalam container

```
select * from todo_items;
```

Hasil:

```
mysql> select * from todo_items;
+----+-----+-----+
| id | name | completed |
+----+-----+-----+
4caf2de-c614-4ab7-bce5-211ae83f5d13	Membuat aplikasi monolith	1
41f043d9-c2c7-43ca-8a54-860a1ffbe941	Membuat service database	1
0d7210ec-0547-4d23-98c3-36827d604fd2	Simpan image di container registry	0
+----+-----+-----+
3 rows in set (0.00 sec)
```

39. Silahkan keluar dari container dengan ketikkan exit.

40. Keren! Kamu sudah bisa membuat aplikasi dengan multi-container. Hmm, proses *deploy* yang kita lakukan sejauh ini cukup rumit ya, padahal kita Cuma berurusan dengan dua *container*. Cukup sulit pula untuk diingat step-by-step nya. Maka dari itu, setelah ini kita akan mencoba mengimplementasikan cara yang lebih *simple* untuk melakukan *deploy* suatu *container*. Kita akan belajar tentang Docker Compose.

Dengan docker compose, kita perlu membuat file YAML untuk mendefinisikan berbagai hal yang perlu dilakukan untuk membuat *service*.

Sebelum mulai, mari kita hapus semua container yang berkaitan dengan Todo App agar tidak terjadi *conflict* saat mencoba docker compose.

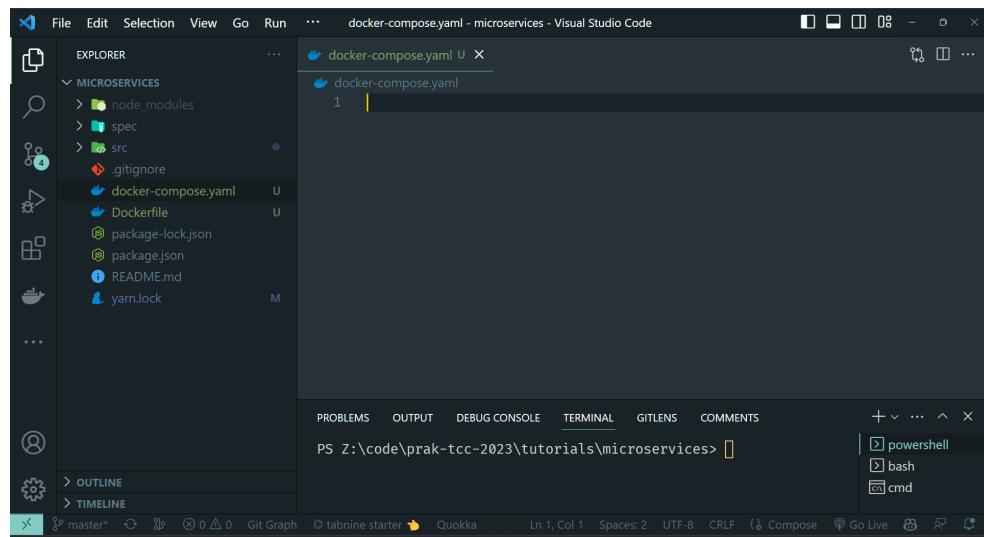
Linux:

```
docker rm -f todo-app mysql-db \
 && docker volume rm todo-db todo-mysql-data \
 && docker network rm todo-app \
 && docker image rm todo-app:v1 todo-app:v2 mysql:5.7
node:12-alpine
```

Windows powershell

```
docker rm -f todo-app mysql-db `
 ; docker volume rm todo-db todo-mysql-data `
 ; docker network rm todo-app `
 ; docker image rm todo-app:v1 todo-app:v2 mysql:5.7
node:12-alpine
```

#### 41. Buat berkas docker-compose.yaml



#### 42. Tuliskan seperti berikut pada berkas docker-compose.yaml!

```
version: "3.7"

services:
```

Schema version dari docker yang digunakan adalah 3.7. silahkan kunjungi tautan berikut untuk melihat informasi lengkap terkait schema version:  
<https://docs.docker.com/compose/compose-file/>

43. Service pertama yang didefinisikan adalah Todo App itu sendiri dengan nama app

```
version: "3.7"

services:
 app:
 image: node:12-alpine
 command: sh -c "yarn install && yarn run dev"
 ports:
 - 3000:3000
 working_dir: /app
 volumes:
 - ./:/app
 environment:
 MYSQL_HOST: mysql
 MYSQL_USER: root
 MYSQL_PASSWORD: tcc2023
 MYSQL_DB: todo-db
```

Semua yang didefinisikan di sini harusnya kamu familiar karena sama dengan yang sebelumnya digunakan pada deploy manual yang kita lakukan. **Image** yang digunakan adalah **node** dengan **versi 12-alpine**. Container akan menjalankan command **sh -c "yarn install && yarn run dev"** ketika container berhasil diluncurkan. Mengakses port **3000**. Workdir diset menjadi **/app**. Memasangkan **current directory** di host ke **/app** di **container**. Serta menentukan **environment variable** apa saja yang digunakan. Nantinya untuk nama container akan tampak seperti ini: **microservices-app-1**

44. Service selanjutnya yang didefinisikan adalah mysql database dengan nama mysql

```
version: "3.7"

services:
 app:
 image: node:12-alpine
 command: sh -c "yarn install && yarn run dev"
 ports:
```

```

- 3000:3000
working_dir: /app
volumes:
- ./:/app
environment:
 MYSQL_HOST: mysql
 MYSQL_USER: root
 MYSQL_PASSWORD: tcc2023
 MYSQL_DB: todo-db

mysql:
 image: mysql:5.7
 volumes:
 - todo-mysql-data:/var/lib/mysql
 environment:
 MYSQL_ROOT_PASSWORD: tcc2023
 MYSQL_DATABASE: todo-db

volumes:
 todo-mysql-data:

```

Secara sederhana, di sini image yang kita gunakan adalah mysql dengan versi 5.7 yang akan menggunakan volume bernama todo-mysql-data dan memasangkannya ke directory /var/lib/mysql di container, menentukan sejumlah environment variable serta membuat volume baru bernama todo-mysql-data. Nantinya untuk nama container akan tampak seperti ini: **microservices-mysql-1**

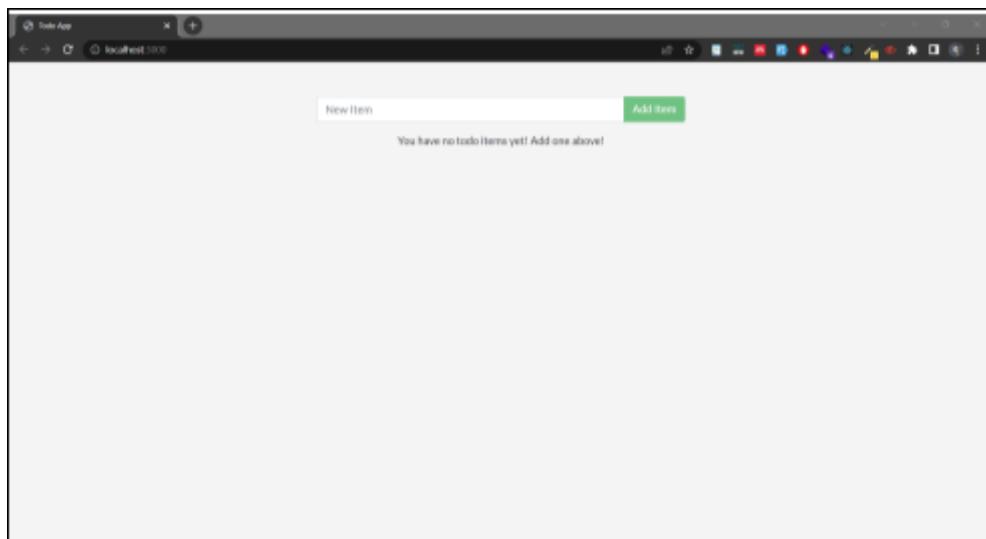
45. Simpan berkas dengan CTRL+S
46. Jalankan perintah berikut untuk memulai aplikasi Todo App menggunakan docker compose

```
docker compose up -d
```

Hasil:

```
[+] Running 17/17
- mysql Pulled
- e048d0a38742 Pull complete
- c7847c8a41cb Pull complete
- 351a550f260d Pull complete
- 8ce196d9d34f Pull complete
- 17feb6f2030 Pull complete
- d4e4268a1fba Pull complete
- fda410389f6f Pull complete
- f47aaac56ba1b Pull complete
- a4a90c369737 Pull complete
- 97091252395b Pull complete
- 84fac29d61e9 Pull complete
- app Pulled
- df9b938bf604a Already exists
- 3bf6d7380705 Already exists
- 7939e601ee5e Already exists
- 31f0fb9de071 Already exists
...
[+] Running 4/4
- Network microservices_default Created
- Volume "microservices_todo-mysql-data" Created
- Container microservices-mysql-1 Started
- Container microservices-app-1 Started
```

47. Silahkan akses todo app via <http://localhost:3000/> .



48. Untuk menghapus resource, caranya jalankan perintah berikut.

```
docker compose down --volumes
```

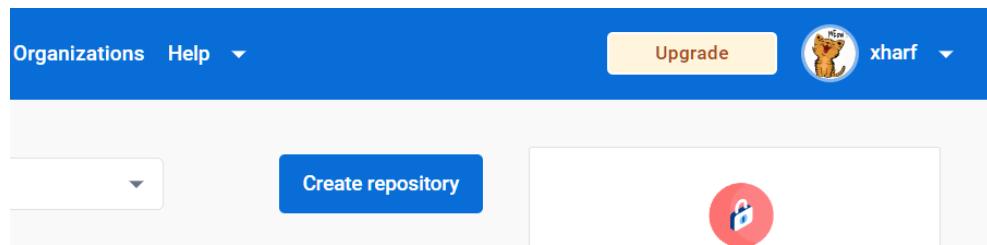
--volumes memberitahu bahwa resource volume juga perlu dihapus. Jika tidak ingin datanya hilang, maka jangan sertakan --volumes pada perintah compose down.

49. Sangat mudah sekali bukan melakukan deploy menggunakan docker compose.

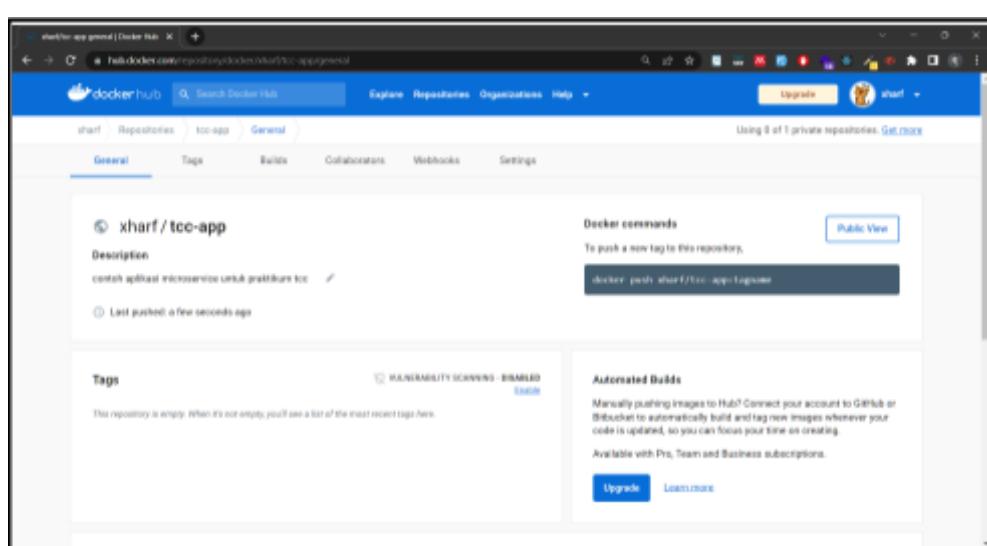
50. Sekarang saatnya kita lanjutkan untuk menyimpan image yang telah kita buat ke Docker Hub. Pertama silahkan buat akun terlebih dahulu dengan mengikuti instruksi pada halaman berikut <https://docs.docker.com/docker-id/>. Bila sudah memiliki akun, silahkan login ke Docker Hub pada laman berikut <https://hub.docker.com/>

51. Pastikan saat ini kamu ada di halaman repository pribadi.

52. Buatlah repository baru dengan klik tombol create repository



53. Isi *repository name* dengan **tcc-app**, kolom *description* diisi bebas, *visibility* isilah dengan **Public**.
54. Sekarang kamu sudah berhasil membuat repo (sebutan singkat untuk repository) baru di Docker Hub.



55. Repo yang kita punya masih kosong, maka dari itu unggahlah image yang ada di local.
56. Buatlah image dari apps kita. Ingat bahwa untuk membuat image, kita gunakan Dockerfile.

```
docker build -t tcc-app .
```

57. Periksa dan pastikan kita sudah punya tcc-app di local.

```
docker images
```

Hasil:

```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
tcc-app latest 817ac222d7b2 About a minute ago 91MB
mysql 5.7 be16cf2d832a 4 days ago 455MB
ubuntu latest 58db3edaf2be 10 days ago 77.8MB
nginx latest a99a39d070bf 3 weeks ago 142MB
node 12-alpine bb6d28039b8c 9 months ago 91MB
```

58. Sebelum menyimpan ke registry, kita perlu mengubah nama image tersebut.

```
docker tag tcc-app:latest <username>/tcc-app:v1
```

Sesuaikan username dengan Docker ID yang kamu buat.

59. Cek kembali image yang kita punya

```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
tcc-app latest 817ac222d7b2 11 minutes ago 91MB
xharf/tcc-app v1 817ac222d7b2 11 minutes ago 91MB
mysql 5.7 be16cf2d832a 4 days ago 455MB
ubuntu latest 58db3edaf2be 10 days ago 77.8MB
nginx latest a99a39d070bf 3 weeks ago 142MB
node 12-alpine bb6d28039b8c 9 months ago 91MB
```

Perhatikan bahwa perintah docker tag di atas tidak menghapus image bernama tcc-app, melainkan membuat repository baru dengan nama yang kita inputkan. Akan tetapi sebenarnya merujuk pada image yang sama. Perhatikan bahwa image ID antar keduanya juga sama.

60. Selanjutnya kita tinggal unggah image baru tersebut ke Docker Hub. Sebelum itu, login terlebih dahulu. Kamu bisa gunakan Docker Desktop atau via Terminal. Jika melalui terminal:

```
docker login --username your_username --password your_password
```

61. Setelah login berhasil, saatnya mengunggah image ke Docker Hub. Silahkan ketikkan perintah berikut:

```
Docker push xharf/tcc-app:v1
```

Hasilnya:

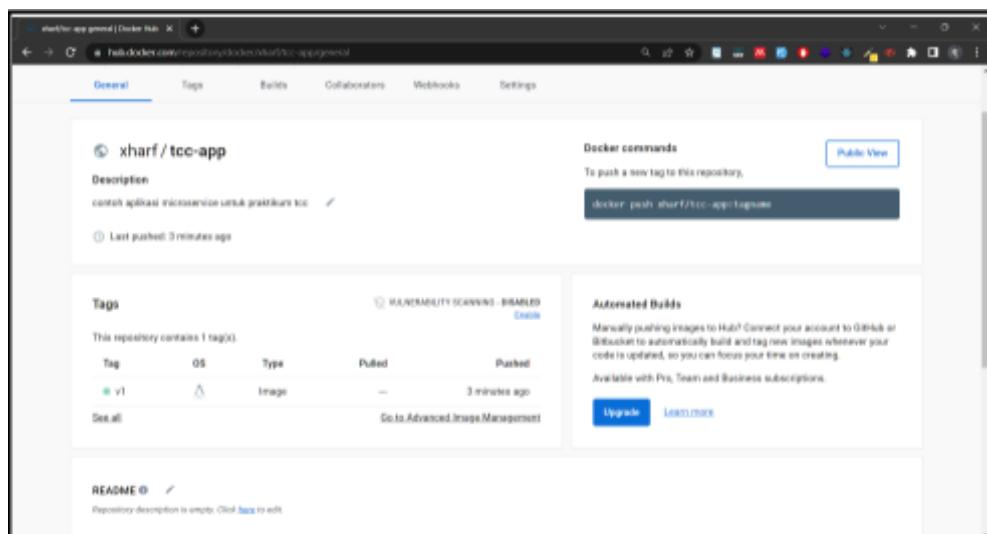
```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ Docker push xharf/tcc-app:v1
The push refers to repository [docker.io/xharf/tcc-app]
02c9e0a10b77: Pushed
7f30cde3f699: Mounted from library/node
fe810f5902cc: Mounted from library/node
dfd8c046c602: Mounted from library/node
4fc242d58285: Mounted from library/node
v1: digest: sha256:e40291126e8cdc8598f29656fdda02562e1dada82daa1817984b75eb3d1d9de9 size: 1365
```

62. NICE! Wait, sebelum lanjut, coba perhatikan baris ini terlebih dahulu.

```
The push refers to repository [docker.io/xharf/tcc-app]
```

Itu artinya, image kita diunggah ke registry bernama **docker.io** (alias Docker Hub), dengan namespace **xharf** (username), dan repository **tcc-app**. Ingat-ingat ya struktur ini.

63. Oke sekarang cek halaman Docker Hub.



64. Mari kita uji coba. Kembali ke terminal, hapus semua image yang berkaitan dengan tcc-app.

```
docker image rm xharf/tcc-app:v1 tcc-app:latest
```

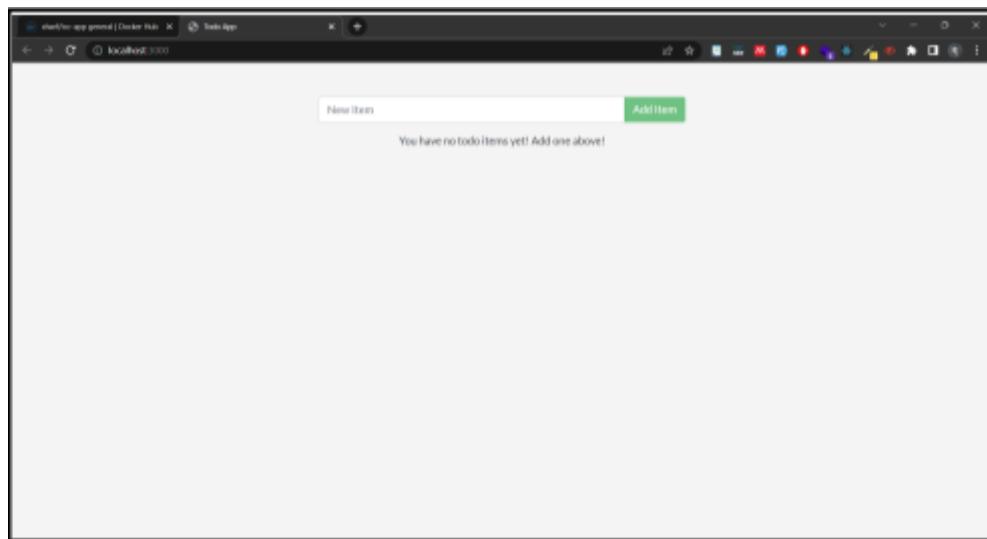
65. Lalu jalankan container menggunakan image dari public repository yang kamu punya

```
docker run -d --name tcc-app -p 3000:3000 xharf/tcc-app:v1
```

Hasil:

```
Shazi@xharf_nitro_5 MINGW64 /z/code/prak-tcc-2023/tutorials/microservices (master)
$ docker run -d --name tcc-app -p 3000:3000 xharf/tcc-app:v1
Unable to find image 'xharf/tcc-app:v1' locally
v1: Pulling from xharf/tcc-app
df9b9388f04a: Already exists
3bf0d7380205: Already exists
7939e601ee5e: Already exists
31f0fb9de071: Already exists
7dd00792b7bd: Already exists
4a6a3000d9e2: Already exists
bdc8dfa97930: Already exists
424c6f0ed21b: Already exists
Digest: sha256:05374364bfedf6edb4d2613b2fff1fb8be513f04d2d57a42f44addff761bec7
Status: Downloaded newer image for xharf/tcc-app:v1
76c03fa476d8ae6a6b0bc659ba7b5e6a2c80f3c5086c31028bd2aa0664f66b09
```

66. Kemudian akses kembali aplikasi kamu melalui alamat <http://localhost:3000/>



Keren! Kamu sudah berhasil membuat repository di Docker Hub, mengunggah Docker image ke Docker Hub, dan menjalankan container menggunakan image dari Docker Hub.

Jangan lupa untuk menghapus semua resource yang kamu buat. Harusnya kamu sudah familiar dengan perintah hapus, jadi untuk itu tidak akan dicontohkan.

Keep in mind bahwa Dockerfile dan docker-compose punya penggunaan yang berbeda.

**Dockerfile** dimaksudkan untuk membuat image, untuk menentukan langkah-langkah untuk membuat image kamu.

Sedangkan,

**Docker-compose** adalah sebuah tool untuk memulai dan mengorkestrasikan container untuk membuat aplikasimu.

Nah setelah kamu berhasil menyimpan image milikmu di repository public seperti Docker Hub, sekarang orang lain dapat menggunakan image yang kamu buat dengan lebih mudah. Kamu juga bisa mengganti pendefinisian image pada docker-compose pada service app yang sebelumnya **node:12-alpine** menjadi image milikmu yaitu **username/tcc-app:v1** yang akan diambil dari Docker Hub

# MODUL 8

## KUBERNETES

---

Materi praktikum kubernetes memiliki cukup banyak teori dengan istilah yang mungkin asing bagimu. Karenanya kamu perlu memberikan fokusmu lebih banyak jika memang ingin memahami kubernetes. Sebagai disclaimer, modul ini hanya memberikan praktik yang sangat sederhana untuk mengenal bagaimana kubernetes bekerja dan dimanfaatkan. Kubernetes sendiri memiliki fitur dan fungsi yang sangat berlimpah yang dapat dimanfaatkan sesuai kebutuhan industri. Demikian sangat disarankan bagimu untuk tidak malas dalam mencari solusi sesuai kebutuhanmu di internet. Karena tidak ada cara pasti yang bisa menjadi solusi untuk menjawab semua masalah. Satu-satunya cara adalah cermat menganalisa kebutuhan dan menyesuaikan apa yang harus dilakukan.

Langkah praktikum modul ini menggunakan lingkungan cloud secara spesifik pada layanan yang ada di Google Cloud Platform (GCP). Sebagai hasil pertimbangan dari kebutuhan resource yang cukup tinggi dan adanya salah satu layanan GCP yang basisnya adalah kubernetes.

Bagi yang memiliki sumberdaya komputasi (laptop / PC) yang mumpuni, kamu bisa menggunakan manfaat dari minikube dengan menginstallnya dari laman berikut: <https://minikube.sigs.k8s.io/docs/start/>. Jangan lupa juga untuk menginstall kubectl apabila belum bisa menjalankannya pada laman berikut <https://kubernetes.io/docs/tasks/tools/>.

Jangan khawatir karena kamu bisa menyelesaikan dan memahami materi ini dengan menjalankan sepenuhnya pada lingkungan cloud.

### A. Tujuan Praktikum

- Mahasiswa dapat memahami kubernetes dan penggunaannya
- Dapat mendeploy aplikasi dengan memanfaatkan kubernetes sebagai container orchestration.

### B. Alokasi Waktu

2 x 60 menit

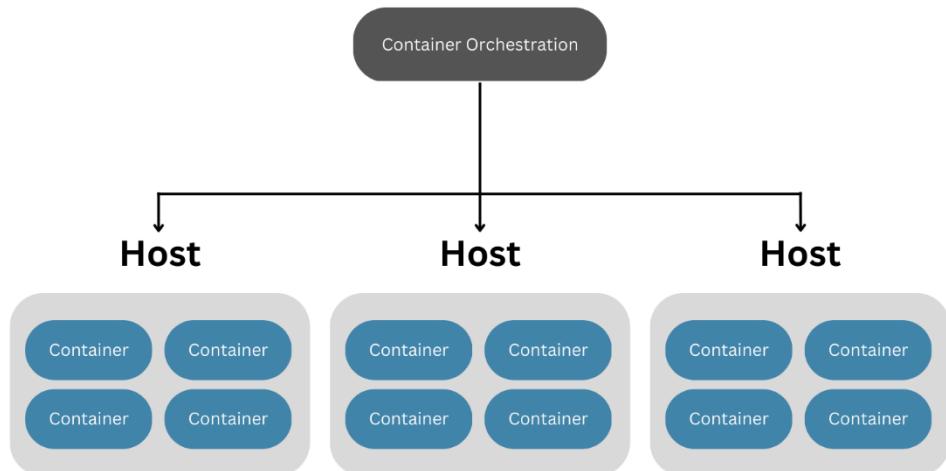
## C. Dasar Teori

### 1. Container orchestration

Masih ingat materi pada modul sebelumnya? Yap microservices. Dalam arsitektur microservices, aplikasi dipecah menjadi sejumlah service (unit atau layanan) yang dikemas dalam container secara independen dan terisolasi satu dari lainnya. Dengan begitu, kamu dapat menambah atau mengurangi jumlah container sesuai kebutuhan dengan mudah. Ini meningkatkan skalabilitas dari aplikasi.

Skalabilitas menjadi tantangan yang nyata dari sisi operasional. Kalau hanya 10 container, mungkin itu bukan masalah. Namun bagaimana jika ada 1000 container? Yakin masih bisa bersantai?

Oleh karenanya container orchestration menjadi praktik otomatisasi deployment, management, scaling, networking, dan availability yang mumpuni terhadap container milikmu. Intinya container orchestration umumnya berhubungan dengan hal terkait siklus hidup container terutama di environment yang memiliki skala besar dan dinamis.



Container orchestration seringkali menjadi opsi ketika perusahaan ingin menerapkan arsitektur microservices, karena cara kerja dari container orchestration sangat mendukung implementasi microservice seakan keduanya tak terpisahkan. Namun demikian, bukan berarti untuk menerapkan microservice harus menggunakan container orchestration. Bisa saja microservice dicapai tanpa menggunakan container orchestration, seperti contohnya serverless microservice dengan memanfaatkan serverless architecture. Modul ini hanya akan berfokus pada

container orchestration karena sampai saat ini masih cukup ideal untuk mengimplementasikan arsitektur microservices.

Berikut beberapa tugas yang mampu ditangani oleh container orchestration:

- Pengelolaan container
- Perpindahan container dari satu host ke host lain
- Alokasi sumber daya antar container
- Ekspos container ke dunia luar
- Load balancing
- Monitoring kondisi container dan host

Saat menggunakan container orchestration tools (disebut juga container orchestrator) seperti Kubernetes (lebih lanjut tentang ini nanti), biasanya kita akan lebih sering untuk mendefinisikan konfigurasi dalam berkas YAML atau JSON (tergantung container orchestrator yang digunakan).

## 2. Opsi container orchestration

Container orchestration tools (kita sebut saja container orchestrator) pada dasarnya menyediakan kerangka kerja atau framework untuk mengelola container dan arsitektur microservices dalam skala besar. Banyak sekali opsi yang dapat digunakan untuk mengelola siklus hidup container. Beberapa diantaranya adalah Kubernetes, Docker Swarm, Red Hat OpenShift, Apache Mesos, dan Hashicorp Nomad.

Mari kita cari tahu beberapa diantaranya dengan lebih detail.

- Kubernetes
- Docker Swarm
- Red Hat OpenShift
- Apache Mesos
- Hashicorp Nomad

## 3. Pengenalan Kubernetes

Nama “Kubernetes” berasal dari bahasa Yunani yang berarti juru mudi atau pilot. Kubernetes juga terkadang disingkat sebagai k8s, yakni hasil dari menghitung delapan huruf antara "k" dan "s".

Awalnya Kubernetes dikembangkan oleh google yang terinspirasi dari platform internal mereka (Borg) yang sangat membantu dalam mengelola ribuan aplikasi dan

service. Borg mampu menyederhanakan development dan manajemen serta membantu pemanfaatan infrastruktur yang lebih tinggi.

Kubernetes akhirnya didonasikan oleh google ke Cloud Native Computing Foundation (CNCF) pada tahun 2014 sebagai teknologi rintisan. Kubernetes juga merupakan container runtime agnostic. Itu berarti, kita bisa menggunakan container runtime yang berbeda dalam berinteraksi dengan Kubernetes. Beberapa opsi container runtime yang didukung oleh Kubernetes adalah containerd, CRI-O, Docker Engine, dan Mirantis Container Runtime.

Kemampuan kubernetes:

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Self-healing
- Secret and configuration management

Kubernetes itu sangat fleksibel, bahkan ia memberikan beberapa opsi deployment sesuai keinginan dan kebutuhan. Beberapa metode dalam men-deploy Kubernetes diantaranya sebagai berikut:

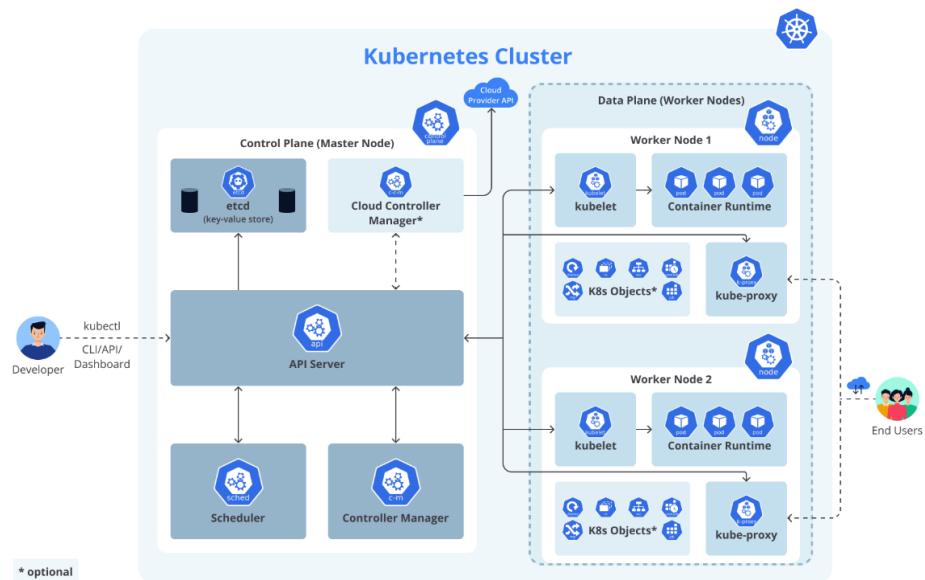
- Single-Node Kubernetes Cluster. Beberapa opsi untuk menerapkan metode ini diantaranya adalah menggunakan Docker Desktop, menggunakan minikube, menggunakan kubeadm.
- Single-Node Kubernetes Cluster for Continuous Integration. Untuk metode ini bisa menggunakan kind (Kubernetes-in-Docker)
- Multi-Node Kubernetes Cluster. Dua metode sebelumnya cocok untuk development dan testing. Sementara untuk production environment, umumnya dibutuhkan Kubernetes Cluster yang disertai dengan beberapa node untuk mengantisipasi kegagalan atau down di suatu node. Beberapa tools yang cocok diantaranya adalah kubespray, kops, kubeadm, Amazon EKS, dan Google Kubernetes Engine (GKE).

#### 4. Arsitektur Kubernetes

Pada dasarnya apabila kita berbicara tentang arsitektur Kubernetes, maka yang kita bicarakan adalah bentuk Kubernetes Cluster. Minimal sebuah Cluster setidaknya terdiri dari satu *Control Plane* dan satu Node. *Control Plane* juga sering disebut sebagai Master Node, sedangkan Node acapkali disebut Worker Node.

*Control Plane* bertanggung jawab untuk mengkoordinasikan semua aktivitas di Kubernetes Cluster, termasuk mengelola Node, menempatkan container ke Node, dan sebagainya. Dalam melakukan tugasnya itu, Control Plane dibantu oleh komponen-komponen, seperti API Server, Scheduler, Controller-Manager, dan etcd.

Sementara itu, Node berlaku bak pekerja (worker) yang meng-hosting Pod (grup yang terdiri dari satu atau lebih container) untuk menjalankan aplikasi. Ia memiliki beberapa komponen, antara lain kubelet, Kubernetes proxy (kube-proxy), dan container runtime.



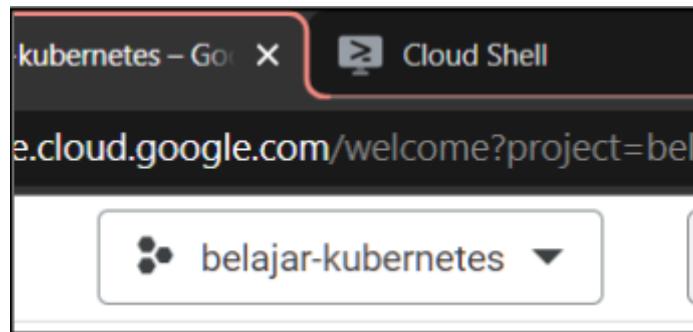
Implementasi Kubernetes akan jauh lebih kompleks di *production environment*. Namun kita tidak perlu terlalu jauh melangkah ke sana. Kita hanya fokus pada arsitektur Kubernetes yang paling sederhana agar mudah dipahami.

## D. Langkah Praktikum

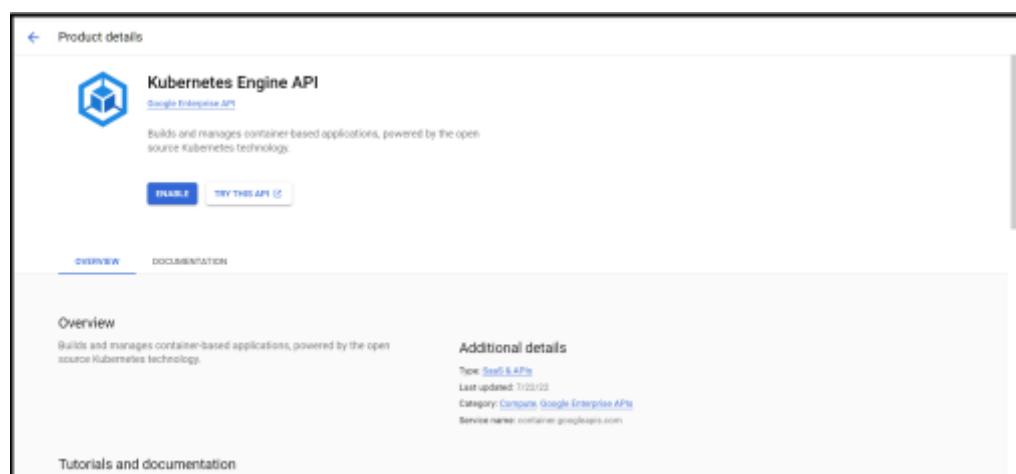
Semua langkah praktikum akan dijalankan pada lingkungan *cloud*, sehingga pastikan kamu memiliki koneksi internet dan akun Google Cloud Platform.

*\*Jika kamu ingin menjalankan praktikum pada lingkungan local, silahkan install minikube dan kubectl pada komputermu melalui laman yang disebutkan di awal modul ini. Pastikan perangkatmu minimal memiliki cpu dengan 2 core atau lebih, tersisa ram 2GB dari total ukuran RAM (misal dari 6 gb harus tersisa 2gb yang free), tersisa 20GB pada ruang penyimpanan, koneksi internet, dan terinstall container atau virtual machine manager, seperti Docker, VirtualBox, VMware, Hyper-V, atau KVM.*

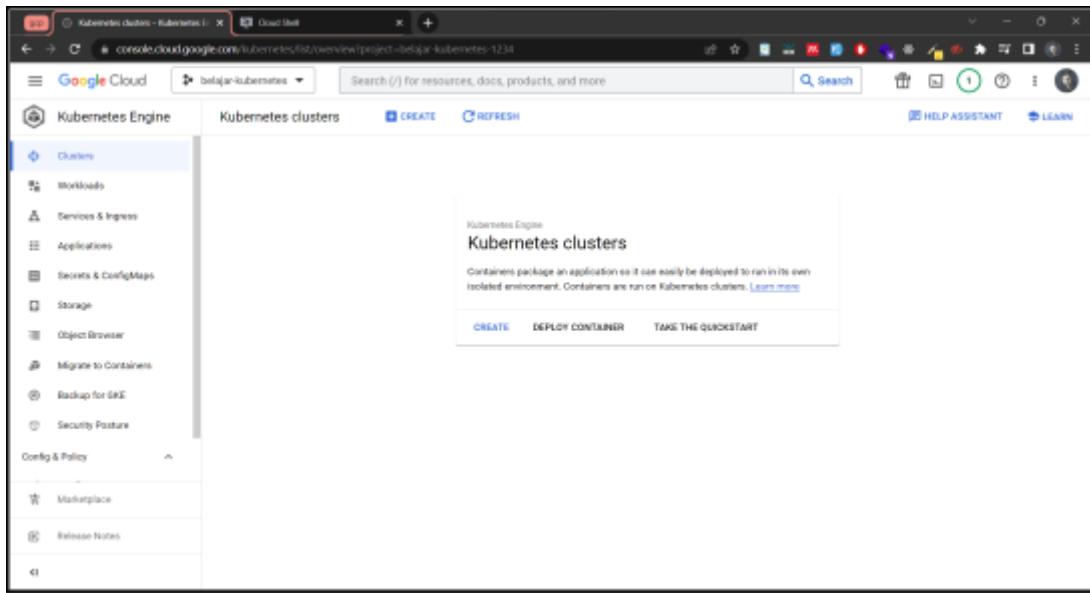
1. Masuklah ke dalam akun GCP.
2. Pilih project yang akan digunakan. (dalam hal ini modul menggunakan project bernama belajar-kubernetes).



3. Cari layanan GCP bernama Kubernetes Engine dan enable API layanan tersebut



Setelah di-enable maka tampilan akan seperti di bawah ini



4. Untuk membuat sebuah cluster, kamu bisa menggunakan GUI atau script. Untuk proses belajar agar terbiasa, mari gunakan script. Jalankan perintah berikut pada cloud shell

Set project terlebih dahulu

```
gcloud config set project your-project-name
```

Set zone terlebih dahulu

```
gcloud config set compute/zone us-central1-b
```

Setelah itu jalankan perintah berikut

```
gcloud container \
--project "belajar-kubernetes-1234" \
clusters create "tcc-cluster" \
--zone "us-central1-b" \
--machine-type "e2-medium" \
--disk-type "pd-standard" \
--disk-size "50" \
--num-nodes "1" \
--node-locations "us-central1-b"
```

Penjelasan:

- Project yang digunakan memiliki id “belajar-kubernetes-1234”. **Ganti dengan projectId milikmu.**
- Nama cluster **“tcc-cluster”**
- Berada di zone **“us-central1-b”**
- Virtual machine bertipe g1-small dengan 2Vcpu dan 4GB RAM
- Persistent disk bertipe standard
- Ukuran disk 50GB
- Jumlah node yang diluncurkan 1
- Dan lokasi node sama dengan lokasi cluster, yaitu us-central1-b

Hasil:

```
NAME: tcc-cluster
LOCATION: us-central1-b
MASTER_VERSION: 1.24.8-gke.2000
MASTER_IP: 35.224.189.137
MACHINE_TYPE: e2-medium
NODE_VERSION: 1.24.8-gke.2000
NUM_NODES: 1
STATUS: RUNNING
```

5. Clone repository yang telah disediakan untuk memudahkan praktik pada modul ini.

```
Git clone https://github.com/1pizzaifupn/kubernetes.git
```

Hasil:

```
shaziawaludin01@cloudshell:~ (belajar-kubernetes-1234)$ git clone
https://github.com/1pizzaifupn/kubernetes.git
Cloning into 'kubernetes'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 21 (delta 2), reused 21 (delta 2), pack-reused 0
Receiving objects: 100% (21/21), 5.17 KiB | 5.17 MiB/s, done.
Resolving deltas: 100% (2/2), done.
```

6. Masuklah ke dalam folder yang baru saja diclone dari github

```
cd kubernetes
```

7. Lihatlah daftar file konfigurasi yang ada di dalamnya

```
ls -1
```

Hasil:

```
app-tier-secret.yaml
app-tier.yaml
data-tier-configmap.yaml
data-tier.yaml
deployment-ns.yaml
hpa.yaml
metric-server.yaml
mysql-pvc.yaml
mysql-pv-pvc.yaml
mysql-pv.yaml
mysql-secret.yaml
mysql-service.yaml
mysql-statefulset.yaml
mysql-svc-deploy.yaml
namespace.yaml
pod.yaml
service.yaml
stateful-ns.yaml
support-tier.yaml
```

Banyak sekali file bertipe yaml di sana. Itu semua yang akan kita gunakan untuk pembelajaran nantinya.

8. Periksalah file bernama pod.yaml yang akan kita gunakan untuk membuat pod selanjutnya

```
cat pod.yaml
```

Hasil:

```
shaziawaludin01@cloudshell:~/kubernetes (belajar-kubernetes-1234)$ cat
pod.yaml
apiVersion: v1
kind: Pod
metadata:
 name: mypod
 labels:
 app: webserver
spec:
 containers:
 - name: mycontainer
 image: nginx:latest
 resources:
 requests:
 memory: "128Mi" # 128Mi = 128 mebibytes
 cpu: "500m" # 500m = 500 milliCPUs (1/2 CPU)
 limits:
 memory: "128Mi"
 cpu: "500m"
 ports:
 - containerPort: 80
```

9. Jalankan pod dengan perintah berikut

```
kubectl apply -f pod.yaml
```

Hasil:

```
pod/mypod created
```

10. Lihat pod yang sedang berjalan dengan perintah berikut

```
kubectl get pods
```

Hasil:

```
shaziawaludin01@cloudshell:~/kubernetes (belajar-kubernetes-1234)$
kubectl get pods
NAME READY STATUS RESTARTS AGE
mypod 1/1 Running 0 9s
```

Apabila status pending, cobalah untuk menjalankannya lagi setelah beberapa menit

11. Lihat pod dengan lebih detail dengan perintah berikut

```
kubectl describe pod
```

12. Cobalah akses container melalui pod dengan perintah berikut (ip didapatkan dari detail saat menjalankan perintah sebelumnya)

```
kubectl exec mypod curl http://<Pod-IP>:80
```

## Hasil:

```
shaziawuludin01@cloudshell:~/kubernetes (belajar-kubernetes-1234)$ kubectl exec mypod curl http://10.44.0.10:80
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
 % Total % Received % Xferd Average Speed Time Time Current
 Dload Upload Total Spent Left Speed
100 615 100 615 0 0 600k 0 --:--:-- --:--:-- 600k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
<p>For online documentation and support please refer to
http://nginx.org/.

Commercial support is available at
http://nginx.com/.</p>
<p>Thank you for using nginx.</p>
</body>
</html>
```

Kita belum bisa mengakses apa yang telah kita buat dari luar lingkungan pod dan kubernetes cluster. Di langkah selanjutnya kita akan membuat service agar apa yang sudah kita buat dapat diakses dari luar.

13. Catatlah Ip milik pod saat ini, lalu cobalah untuk menghapus pod lalu jalankan kembali pod dan cek ip milik yang baru saja dijalankan. Jalankan perintah berikut untuk menghapus pod.

```
kubectl delete pods mypod
```

Dari proses ini harusnya kamu sadar bahwa IP berubah. Kubernetes bisa saja me-restart pod jika terjadi sesuatu yang tidak semestinya. Saat pod di-restart, bisa jadi ip addressnya berbeda dengan yang sebelumnya. Apalagi jika kita ingin melakukan scaling terhadap pod, maka ipnya akan berbeda-beda. Lalu ip address mana yang digunakan? Tidak ideal sekali untuk production environment. Tidak mungkin kan kita selalu merubah ip yang ada di frontend tiap kali pod backend berganti ip.

Untuk mengatasi itu kubernetes punya object bernama service. Ia adalah sebuah abstraksi yang mengekspos aplikasi yang berjalan pada satu atau lebih Pod sebagai sebuah layanan jaringan sehingga aplikasi kamu bisa diakses dari internal (misalnya, sesama Pod) maupun eksternal (misalnya, dari host atau bahkan internet). Selain itu, Service juga berguna untuk load balancing (mendistribusikan traffic di antara Pod) dan service discovery (mekanisme di mana Pod dapat menemukan Pod lain secara dinamis).

Sebuah Service dapat menggunakan selector untuk mengidentifikasi Pod yang memiliki label tertentu. Sebagai contoh, sebuah Service memiliki selector “**app: webserver**”. Service akan mengidentifikasi Pod mana saja yang memiliki label “**app: webserver**”. Setiap Pod yang memiliki label tersebut, secara otomatis akan ditemukan dan diasosiasikan oleh Service.

14. Pada langkah sebelumnya kita sudah mendefinisikan Pod **mypod** dengan label **app:webserver**. Akan kita gunakan label tersebut untuk mengasosiasikannya dengan Service. Periksalah **service.yaml** yang sudah disediakan dengan menjalankan perintah berikut:

```
cat service.yaml
```

Hasil:

```
apiVersion: v1
kind: Service
metadata:
 labels:
 app: webserver
 name: webserver
spec:
 ports:
 - port: 80
 selector:
 app: webserver
 type: NodePort
```

15. Deploy berkas manifest tersebut untuk membuat Service

```
kubectl apply -f service.yaml
```

Hasil:

```
service/webserver created
```

16. Periksa daftar service yang kita miliki

```
kubectl get service
```

Hasil:

```
shaziawaludin01@cloudshell:~/kubernetes (belajar-kubernetes-1234)$
kubectl get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.48.0.1 <none> 443/TCP 31m
webserver NodePort 10.48.5.10 <none> 80:31459/TCP 5s
```

Fokus pada Service bernama **webserver**. Ia merupakan Service yang baru saja kita buat dengan tipe NodePort. Terlihat bahwa Service tersebut memiliki sebuah private IP address dan port mapping. Angka 80 adalah port yang diekspos Service; sedangkan 32599 adalah port yang dialokasikan otomatis dari Port range untuk NodePort, umumnya antara 30.000 hingga 32.767.

Perhatikan bahwa kita memiliki **External-IP** bernilai none. Ini karena NodePort tidak menyediakan public Ip Address. Bagian ini akan terisi bila menggunakan tipe **LoadBalancer**.

17. Ubah terlebih dahulu type pada file service.yaml dengan cara

```
nano service.yaml
```

Kemudian ubah filenya menjadi seperti berikut

```
apiVersion: v1
kind: Service
metadata:
 labels:
 app: webserver
 name: webserver
spec:
 ports:
 - port: 80
 selector:
 app: webserver
 type: LoadBalancer
```

Tekan **CTRL + X** lalu ketikkan **Y** lalu enter

18. Hapus terlebih dahulu service web server bila ada

```
kubectl delete svc webserver
```

19. Expose service agar memiliki eksternal ip dengan apply lagi file service.yaml

```
kubectl apply -f service.yaml
```

20. Lihat detail service yang ada

```
kubectl get service
```

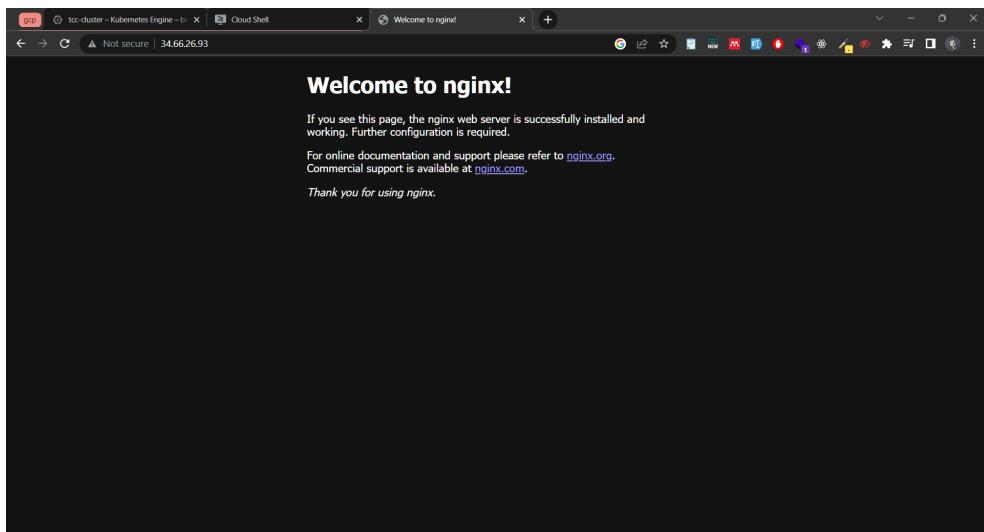
Hasil:

```
shaziawaludin01@cloudshell:~/kubernetes (belajar-kubernetes-1234)$ kubectl
get service
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.48.0.1 <none> 443/TCP 73m
webserver LoadBalancer 10.48.2.106 34.66.26.93 80:31288/TCP
3m50s
```

Pastikan external-ip sudah ada. Baru lanjut ke langkah selanjutnya.

21. Cobalah hit external ip melalui browser

Hasilnya:



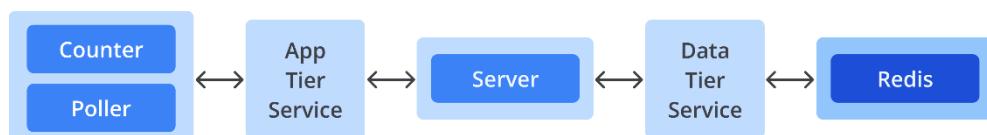
22. Kamu bisa coba hapus pod yang ada lalu hit lagi external ip milik Service via browser. Harusnya kamu akan mendapatkan hal yang sama berupa welcome to nginx setelah beberapa saat bahkan ketika ip Pod berubah.

Selamat, kamu telah berhasil mendeploy suatu layanan hingga go public. Jangan lupa untuk menghapus resource yang digunakan ya. Mudahnya, kamu bisa menggunakan `gcloud container clusters delete tcc-cluster --zone=us-central1-b` untuk menghapus cluster beserta semua isinya.

Sejurnya, praktik pembuatan Pod secara langsung seperti yang kita lakukan ini bukanlah best practice. Seharusnya kita menggunakan Deployment untuk membuat Pod. Sejatinya Deployment merepresentasikan sejumlah Pod yang identik (disebut replica). Dengan membuat sebuah Deployment, kita menentukan desired state (kondisi

yang diinginkan) yang harus dicapai oleh Kubernetes. Sebagai contoh, kita menginginkan 3 buah replica Pod dengan Node.js image. Kubernetes akan berupaya untuk mengubah actual state (kondisi saat ini) menjadi desired state (kondisi yang diinginkan) yang telah ditentukan. Itu artinya apabila saat ini belum ada Pod dengan Node.js sama sekali, Kubernetes akan segera meluncurkan 3 buah Pod dengan Node.js image untuk memenuhi desired state.

Pembahasan ke arah sana akan sangat panjang dan tidak akan cukup jika dipadatkan di waktu praktikum selama 2 jam. Oleh karenanya silahkan cek pembahasan materi pada bagian readme di <https://github.com/1pizzaifupn/kubernetes> jika kamu tertarik dengan pembahasan tentang Kubernetes secara lebih menyeluruh. Di sana akan ada studi kasus untuk membuat sebuah arsitektur microservice yang lebih nyata dengan bagan seperti berikut.



Daftar pembahasan yang bisa kamu caritahu lebih lanjut di repository di atas diantaranya adalah Deployment, HorizontalPodAutoScaler, Volume, Persistent Volume, ConfigMap, Secret, dan StatefulSet. Bahkan itu hanya segelintir kecil yang bisa dilakukan oleh Kubernetes. Nyatanya Kubernetes jauh lebih kompleks dan lebih powerful dari yang kita pelajari di praktikum ini.

Sesuatu yang besar bisa dimulai dari hal-hal kecil, dan ini awalan yang bagus untuk terus berkelana mendalami cloud computing.

~ Semoga perjalananmu dan hari-harimu menyenangkan ~

## Daftar Pustaka

- “Writing Your Web Service with Node.js | Google App Engine standard environment docs.” Google Cloud, <https://cloud.google.com/appengine/docs/standard/nodejs/building-app/writing-web-service>. Accessed 4 February 2023.
- Verbanic, Mike, and Kevin Holbrook. “Deploy a website with Cloud Run.” Google Codelabs, January 2022, <https://codelabs.developers.google.com/codelabs/cloud-run-deploy#0>. Accessed 4 February 2023.
- Asikpo, E. (2022, Januari 19). Tips CKAD: Cara membuat menghubungkan ke Google Cloud VM menggunakan SSH di Mac/Linux. Lab Duta Besar. Diakses tanggal February 5, 2023, from <https://blog.getambassador.io/ckad-tips-how-to-create-connect-to-a-google-cloud-vm-using-ssh-on-mac-linux-22fd016887ca>
- Bhardwaj, S., Jain, L., & Jain, S. (n.d.). Komputasi awan: Studi tentang infrastruktur sebagai layanan (IAAS). Jurnal Internasional teknik dan teknologi informasi, 2(1), 60-63.
- Panduan, S. (N.D.). Mesin Komputasi: Komputer Virtual (VM). Google Awan. Diakses tanggal January 31, 2023, from <https://cloud.google.com/compute>
- Kaelin, M. W. (2022, 8 November). Cara membuat mesin virtual di Google Cloud Platform. TechRepublic. Diakses tanggal January 31, 2023, from <https://www.techrepublic.com/article/how-to-create-a-virtual-machine-in-google-cloud-platform/>
- Buat kunci SSH Anda secara manual di macOS. (2022, 9 November). Dokumentasi. Diakses tanggal February 1, 2023, from <https://docs.tritondatacenter.com/public-cloud/getting-started/ssh-keys/generating-an-ssh-key-manually/manually-generating-your-ssh-key-in-mac-os-x>
- Ramakrishnan, N. (2019, Mei 10). Menghubungkan ke Mesin Virtual Google Cloud dengan SSH menggunakan PUTTY. Sedang. Diakses tanggal February 1, 2023, from [https://medium.com/@narayanan\\_ramakrishnan/connecting-to-a-google-cloud-virtual-machine-with-ssh-using-putty-7b6f0c0465cb](https://medium.com/@narayanan_ramakrishnan/connecting-to-a-google-cloud-virtual-machine-with-ssh-using-putty-7b6f0c0465cb)
- Wilayah dan zona | Dokumentasi Mesin Komputasi. (n.d.). Google Awan. Diakses tanggal February 1, 2023, from <https://cloud.google.com/compute/docs/regions-zones>
- Tau VM dengan performa terdepan di industri. (n.d.). Google Awan. Diakses tanggal January 31, 2023, from <https://cloud.google.com/tau-vm>

- Install Docker on Linux. (n.d.). Seven Bridges. Retrieved February 9, 2023, from <https://docs.sevenbridges.com/docs/install-docker-on-linux>
- Install Docker on macOS. (n.d.). Runnable. Retrieved February 9, 2023, from <https://runnable.com/docker/install-docker-on-macos>
- Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017, March). An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*, 17(3), 228-235.
- Reference documentation. (n.d.). Docker Documentation. Retrieved February 9, 2023, from <https://docs.docker.com/reference/>
- Tutorial Docker: Cara Install dan Setup Docker dengan Benar! (2022, November 24). Petani Kode. Retrieved February 9, 2023, from <https://www.petanikode.com/docker-install/>