



CI/CD

Chapter 5 - Topic 4

**Selamat datang di Chapter 5 Topic 4 online course
React Native dari Binar Academy!**

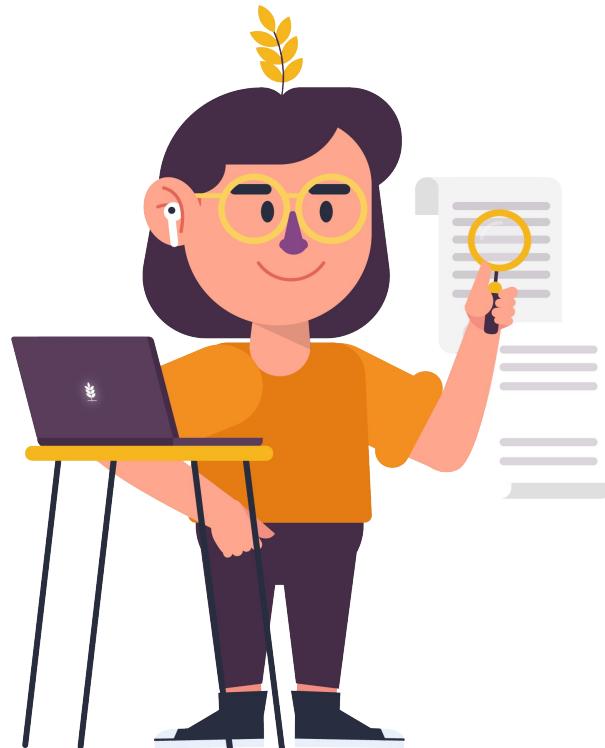




Hello guys!

Kita lanjutkan perjalanan Chapter 5 ini ke topik 4 yuk. Di topik sebelumnya kamu sudah belajar tentang Automation Testing kan? Nah di topic 4 ini kamu akan belajar pengenalan tentang CI/CD nih.

Penasaran nggak dengan CI/CD ini? Yaudah, yuklah langsung kita bahas~





Detailnya, kita bakal bahas hal-hal berikut ini:

1. Memahami proses CI/CD
2. Memahami maksud penulisan .gitlab-yml
3. Memahami Pipeline & Job Status





Guys, menurut kamu kenapa kita harus belajar CI/CD ini?

Biar nggak penasaran dan makin semangat, yuk langsung aja kita bahas Why-nya nih! Cekidot~





Pembahasannya kita mulai dari sini ya~

Guys, di proses software development, ada tahap pengujian atau testing yang dilakukan oleh QA team alias tim penguji aplikasi secara End-to-End.

Jadi, ketika kamu selesai mengembangkan suatu fitur dan melakukan penggabungan fitur ke branch development. Langkah selanjutnya adalah melakukan build apk yang nantinya akan dikirimkan ke QA team untuk diuji.





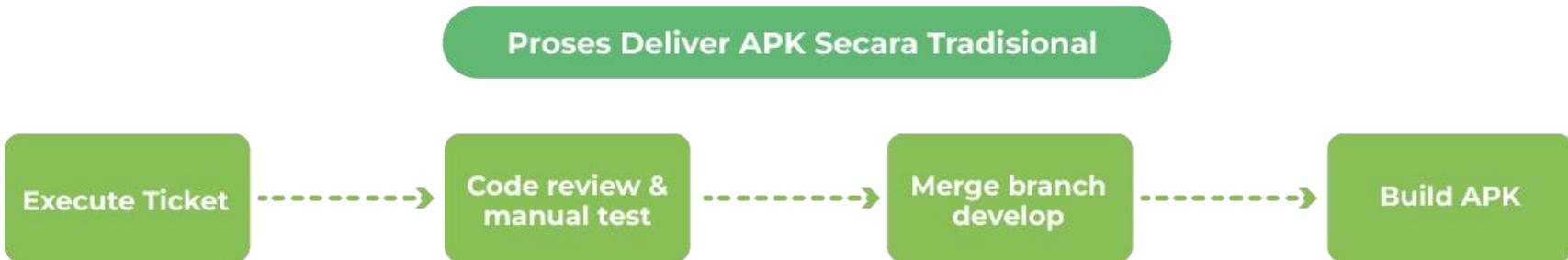
Biasanya, kamu **harus melakukan build manual** apk yang akan diberikan ke QA untuk direview.





Jika divisualisasikan, proses inilah yang kamu lakukan:

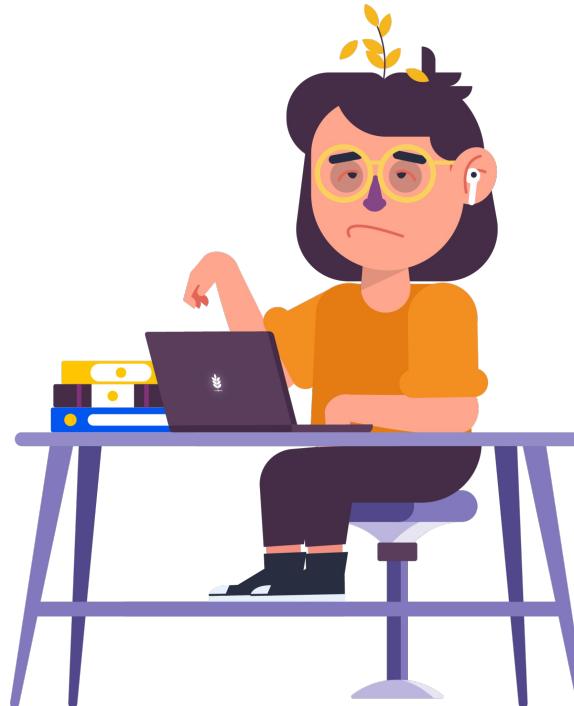
1. Mendapatkan tiket atas task yang dikerjakan
2. Melakukan code review dan manual test
3. Merge dengan branch development, sehingga kita mendapat code terbaru dari upstream
4. Build APK





Testing bisa dilakukan manual, tapi proses ini bakal bikin capek banget~

Yaps, jika proses yang dilakukan secara tradisional dilakukan terus menerus, maka nggak akan efektif dan efisien.





Contoh kasusnya kayak gini

Ketika QA melakukan pengetesan pada aplikasi, maka developer harus menunda terlebih dahulu pekerjaannya dan menyiapkan (build apk) yang akan dites oleh QA.

Hal ini akan mengganggu fokus developer yang sedang mengerjakan tugasnya dalam membuat aplikasi, oleh karena itu solusinya kita harus menerapkan CI/CD sehingga pekerjaan build apk akan dihandle secara otomatis oleh job status dan pipeline.





CI/CD wajib banget!

Perusahaan-perusahaan besar di Indonesia seperti Tokopedia, Traveloka, Bukalapak, dan lainnya juga pakai CI/CD untuk aplikasi mereka. Tapi kenapa ya?





CI/CD harus banget digunakan di era pengembangan aplikasi modern karena proses yang ada di dalamnya udah sangat besar dan banyak, guys!

Jadi harus ada tools yang bisa melakukan otomatisasi otomatisasi dalam membangun, menguji, dan menyebarkan aplikasi. Bayangin aja kalau semua itu masih dilakukan secara manual, ampun deh~





Jadi... CI/CD bisa dibilang dewa penolong untuk para developer aplikasi ya, guys! Karena bikin proses testing aplikasi bisa lebih cepat.

Harus banget nih kita tahu tentang CI/CD!
Cekidot~





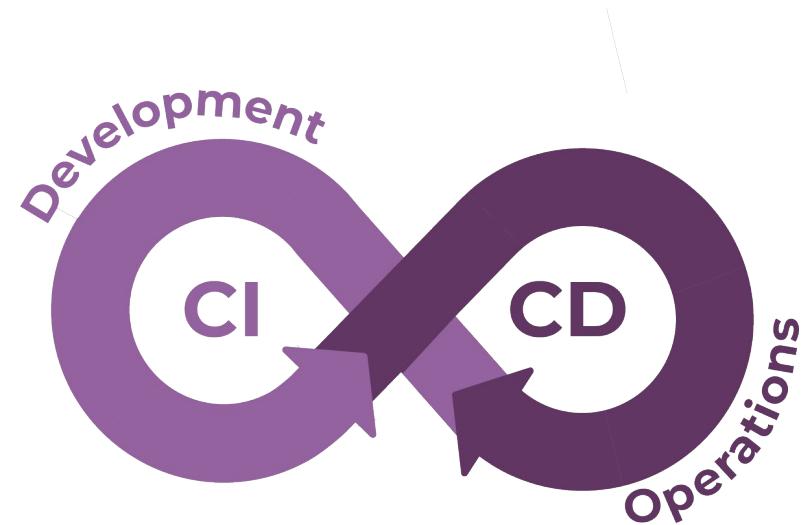
Dasar cara kerja CI/CD nih, guys

Dalam Software Engineering, CI/CD umumnya mengacu pada praktik gabungan dari continuous integration dan continuous delivery.



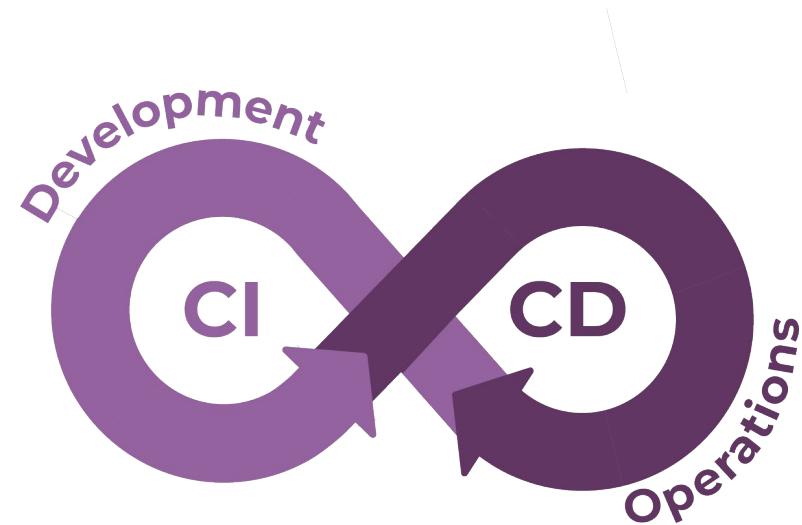


Continuous integration, artinya selama proses development aplikasi, code terintegrasi di dalam satu repository sehingga bisa dilakukan proses pengujian secara otomatis, cepat, dan lebih sering.





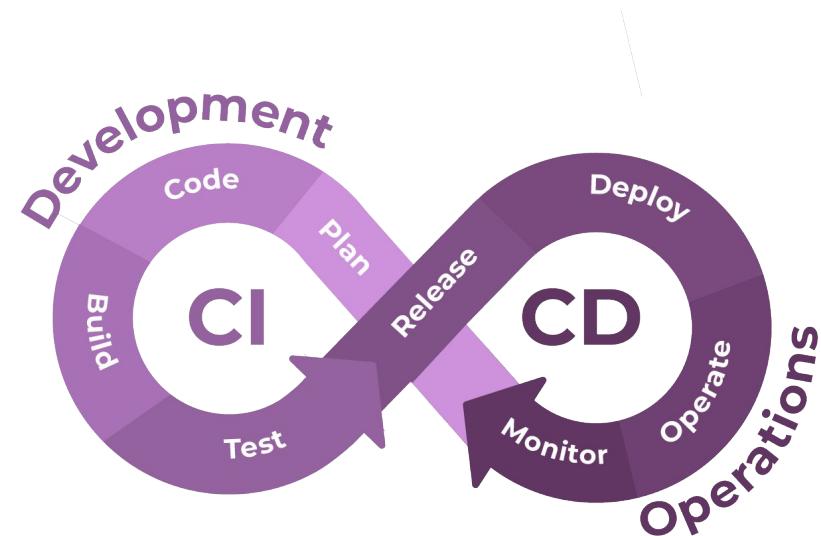
Continuous delivery. adalah praktik yang dilakukan setelah proses CI selesai dan seluruh kode berhasil terintegrasi, sehingga aplikasi bisa dibangun lalu dirilis secara otomatis.





CI/CD sebagai jembatan

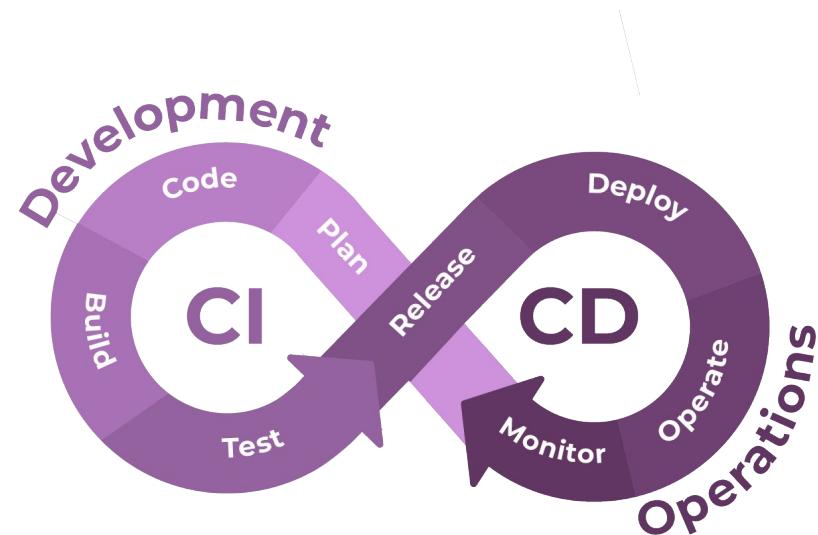
CI/CD menjembatani kesenjangan antara aktivitas development dan Operations dengan menerapkan otomatisasi dalam membangun, menguji, dan menyebarkan aplikasi.





Praktik DevOps modern melibatkan proses development, proses testing, integrasi, penerapan, dan pemantauan aplikasi yang dibuat.

Semua proses itu dilakukan secara berkelanjutan sepanjang proses pengembangan sebuah aplikasi. Praktik CI/CD atau siklus CI/CD membentuk tulang punggung operasi DevOps modern.

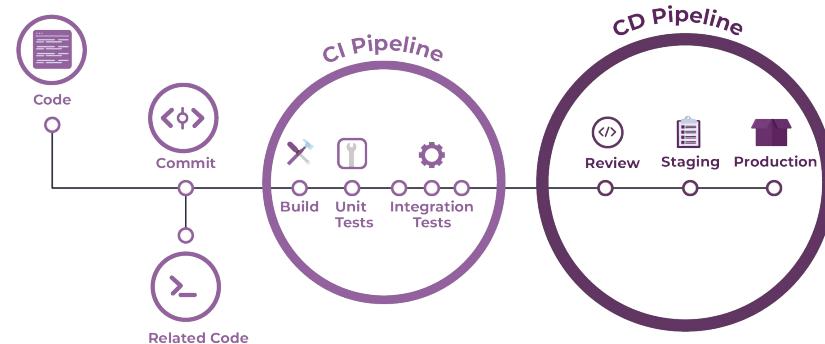




Secara sederhana, begini nih guys pengertian CI/CD

CI/CD merupakan dua proses yang digabung menjadi satu kesatuan yaitu **Continuous Integration** dan **Continuous Deployment**.

Secara lingkup pekerjaan, masing-masing dapat dilihat perbedaannya melalui visualisasi di samping ini:

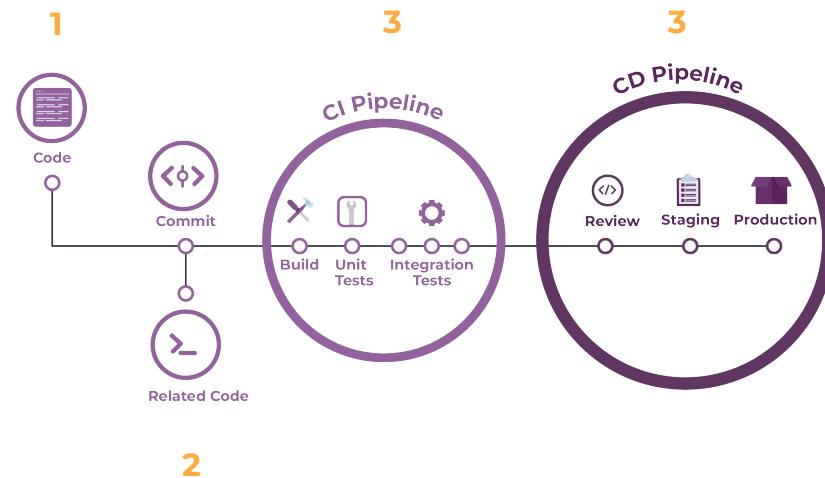




Nah sekarang, begini alur kerja CI/CD

Proses kerja CI/CD utamanya adalah di saat kamu telah mengerjakan suatu code (nomor 1) dan code yang di-upstream akan menerima perubahan baru (nomor 2).

Ketika kamu akan melakukan push terhadap code ke branch yang terhosting oleh Gitlab CI/CD, eksekusi tersebut akan mentrigger ci/cd pipeline(nomor 3).

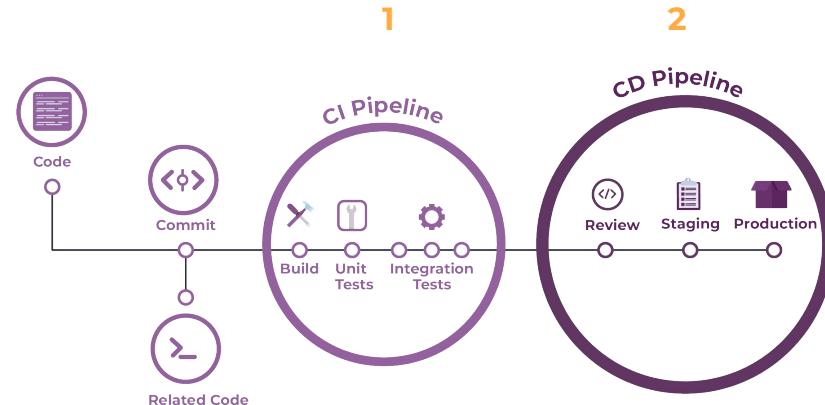


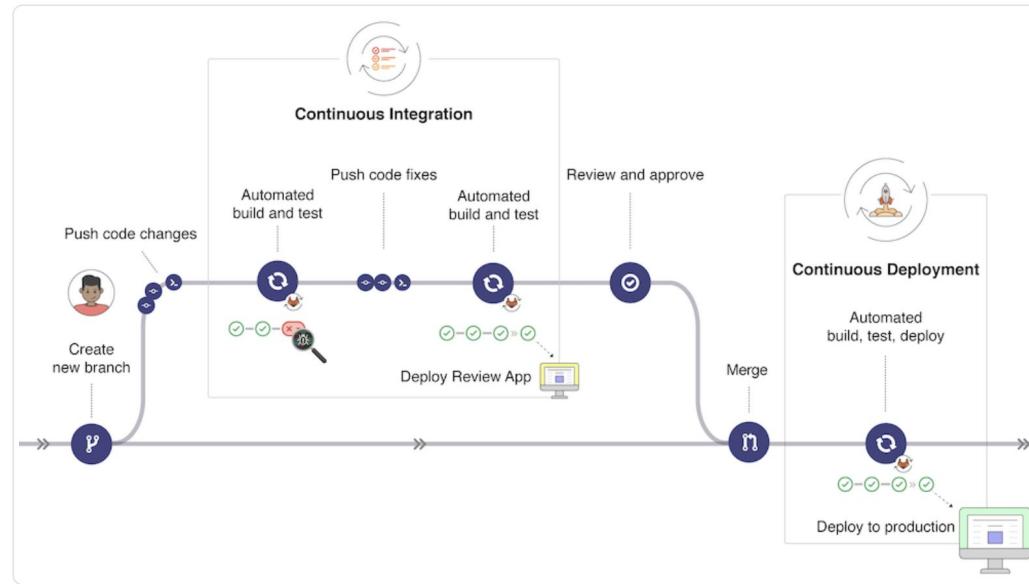


Selanjutnya, Gitlab CI/CD yang sudah dibuat otomatis akan berjalan paralel dengan menjalankan beberapa pekerjaan di antaranya :

1. Build and Test aplikasi
2. Preview perubahan code seperti apa yang telah kamu buat di local komputer

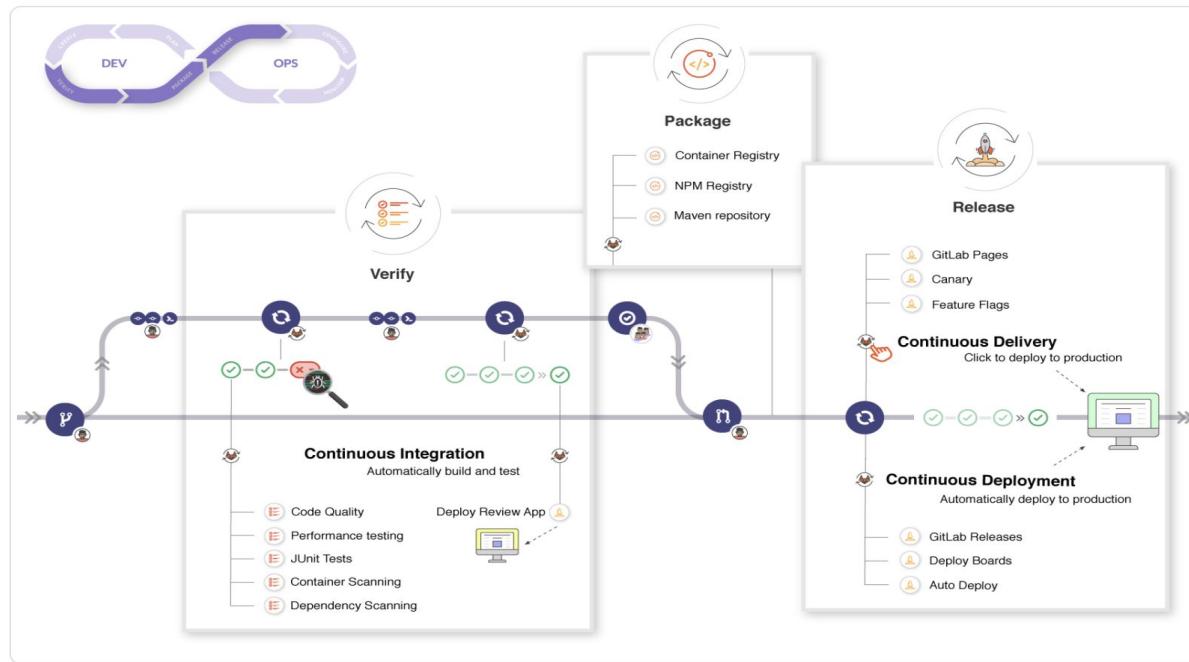
Jika semua sudah sesuai dan memenuhi persyaratan selanjutnya gitlab CI/CD akan melakukan distribute app-mu yang sudah siap untuk di-review.





Jika ada sesuatu kesalahan atau dalam hal ini terjadi bug di production dengan CI/CD, **kamu dapat melakukan rollback** sehingga apps kembali ke versi sebelumnya.

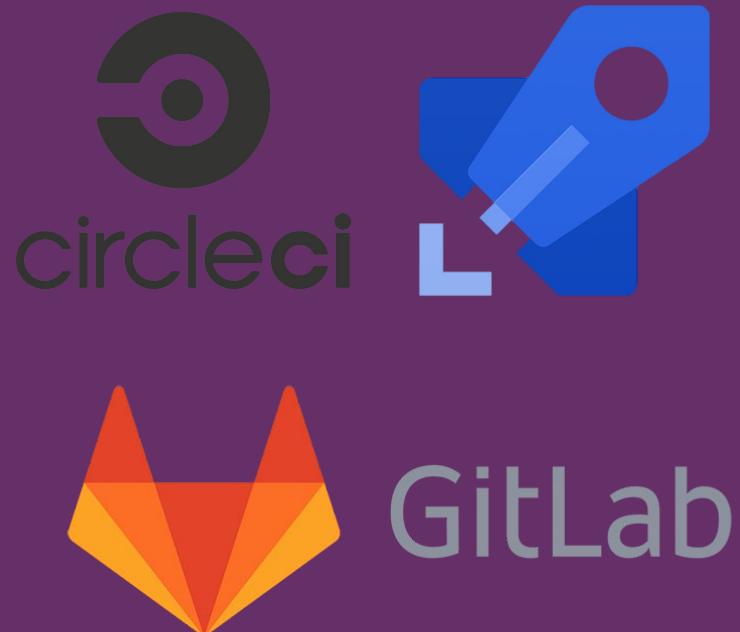
Secara garis besar GitLab CI/CD Process divisualisasikan seperti gambar di bawah ini.



Jika kamu melihat lebih dalam dari process CI/CD, kamu dapat melihat beberapa fitur yang tersedia oleh GitLab di setiap devOps operations.



Guys! Ada banyak platform yang bisa digunakan untuk melakukan CI/CD contohnya CircleCI, Azure pipeline, dan GitLab.





Di materi kali ini kita akan menggunakan GitLab. Lalu bagaimana CI/CD pada GitLab dapat bekerja?

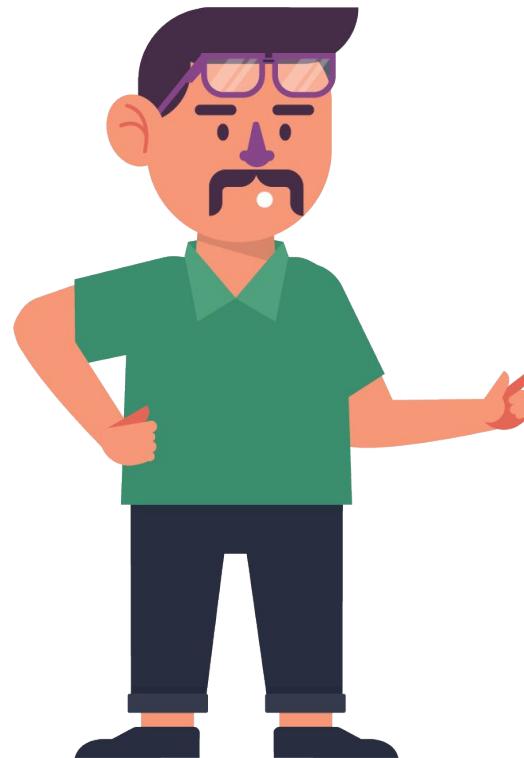
Yuk simak pembahasannya berikut ini!





Untuk bisa menggunakan GitLab CI/CD, yang kamu perlukan adalah:

1. Kode aplikasi yang dihosting di repositori Git
2. Sebuah **file bernama .gitlab-ci.yml** yang diletakkan di root project kamu. File ini berisi konfigurasi CI/CD.

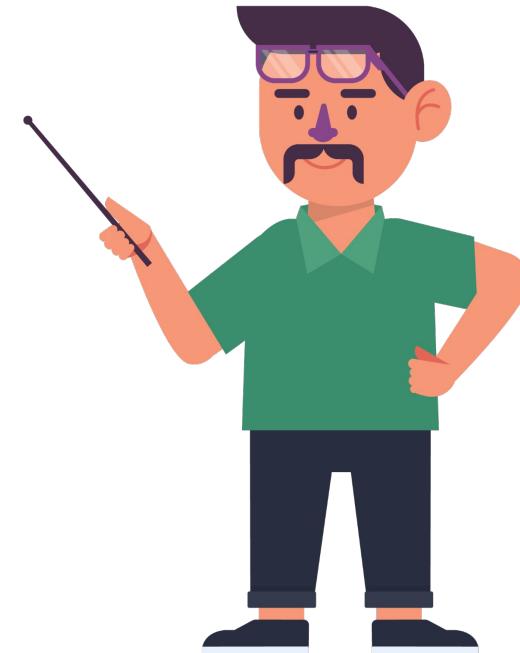




Bagaimana gitlab-ci.yml bisa berjalan?

Sebelumnya kamu sudah mengenal script start yang ada di package.json seperti npm run scriptName (start,test,clear-watch, dan lain-lain). Script ini akan berjalan dengan trigger yang kamu ketik di terminal.

Nah, gitlab-ci.yml adalah script yang akan terpicu setelah kamu melakukan push code, di mana di dalamnya menyimpan command yang akan tereksekusi otomatis.





list process setup

Oke sekarang kita sudah tau beberapa hal yang perlu kita lakukan untuk mencapai app kita agar dapat didistribusikan diantaranya : build stage and deploy stage





Mari kita memahami code yang berada di .gitlab-ci.yml.

Code ini berisikan tahapan yang akan dijalankan setelah CI/CD ter-trigger. Seluruh command yang akan dijalankan disimpan di bagian script untuk build apk sampai dimana apk itu disimpan di root folder kita.

```
17 lines (14 sloc) | 280 Bytes

1 image: reactnativecommunity/react-native-android
2
3 before_script:
4   - npx envinfo
5
6 stages:
7   - build
8
9 build:
10  stage: build
11  script:
12    - yarn install
13    - cd android && chmod +x gradlew
14    - ./gradlew assembleRelease
15  artifacts:
16    paths:
17      - android/app/build/outputs/
```



Oke, sampai di sini kamu pasti sudah punya bayangan kan tentang kegunaan dan keperluan file .gitlab-ci.yml?





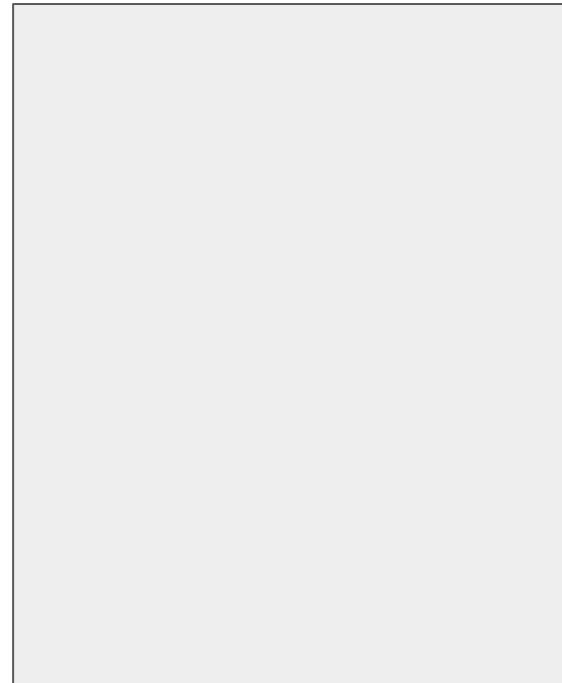
Nah, setelah ini kita akan membahas sebuah komponen yang berjalan mengikuti file `.gitlab-ci.yml` dan ter-trigger setelah developer melakukan push, namanya **Pipeline**.





Pipeline ini bukan pipa ya, Sob!

Pipeline adalah komponen tertinggi dari continuous Integration, delivery, dan deployment.





Apa yang ada di dalam pipeline CI/CD?

Pipeline terdiri dari :

1. Jobs, yang menjelaskan **apa yang sedang dilakukan/dieksekusi.** Contohnya ugas yang mengompilasi code, atau melakukan test code.
2. Stages, menentukan **apa pekerjaan yang harus dijalankan dan kapan harus yang berjalan.** Contohnya tahap yang menjalankan test setelah tahap kompilasi code dijalankan.



Ini dia mekanisme cara kerja Jobs

Jobs dijalankan oleh GitLab runners. Beberapa jobs di stage yang sama dieksekusi secara paralel, jika ada cukup banyak runner yang beroperasi secara bersamaan.





Bagaimana jika semua job berhasil dijalankan dan jika ada jobs yang gagal dijalankan? Yuk simak~

Jika semua pekerjaan dalam satu tahap berhasil, jalur pipeline akan pindah ke tahap berikutnya.

Jika ada jobs dalam satu stage yang gagal, tahap berikutnya (biasanya) tidak dijalankan dan pipeline berakhir lebih awal.





Secara umum, pipeline dijalankan secara otomatis dan tidak memerlukan intervensi setelah dibuat. Namun, ada juga saat Anda dapat berinteraksi secara manual dengan pipeline.



Tahapan dalam sebuah Pipeline

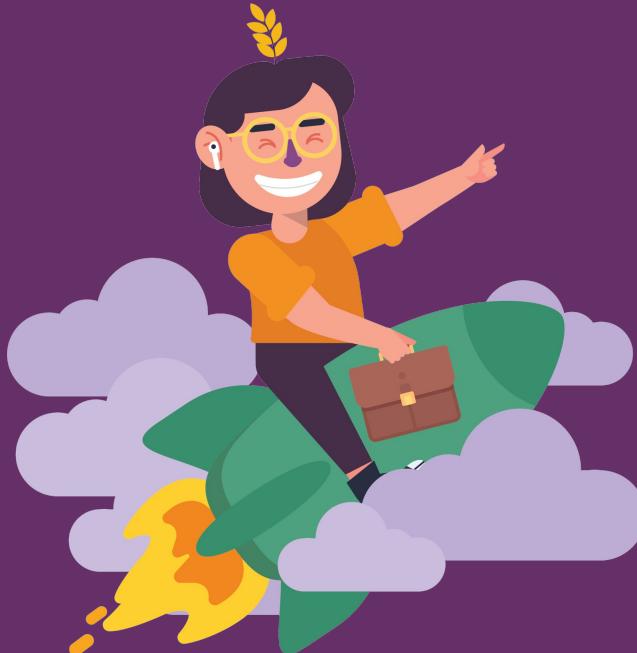
Sebuah pipeline terdiri dari empat tahap, dieksekusi dalam urutan berikut:

1. Tahap build, dengan tugas (job) bernama compile.
2. Tahap test, dengan tugas (job) bernama test1 dan test2.
3. Tahap staging, dengan tugas (job) bernama deploy-ke-staging
4. Tahap production, dengan tugas (job) bernama deploy-ke-production

Note : Stage di atas sesuai dari stage yang kita definisikan pada code .gitlab-ci.yml



Sekarang kamu akan mengetahui lebih jelas apa itu Jobs pada CI/CD





Mari pahami Jobs

Jobs adalah elemen paling mendasar dari file .gitlab-ci.yml. Beberapa pengertian Jobs :

- Didefinisikan dengan batasan yang menyatakan dalam **kondisi apa mereka harus dieksekusi**.
- Terdiri dan diawali oleh **script**
- **Tidak terbatas** mau berapa banyak jobs yang dibuat

```
job1:  
  script: "execute-script-for-job1"  
  
job2:  
  script: "execute-script-for-job2"
```



Guys, perhatikan contoh gambar di samping dulu yuk~

Contoh tersebut menggambarkan Jobs yang dibagi menjadi dua tugas, yang akan berjalan secara terpisah/sendiri-sendiri.

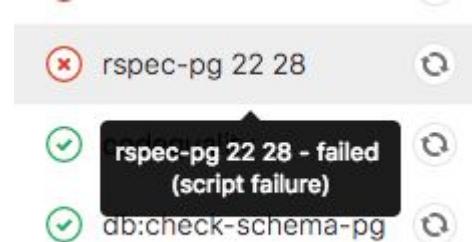
Tentunya Jobs akan menjalankan script atau command eksekusi yang ada dalam project repository, seperti “make install script” yang terdapat di dalam repository.

```
● ● ●  
job1:  
    script: "execute-script-for-job1"  
  
job2:  
    script: "execute-script-for-job2"
```



Jobs juga bisa melakukan ini, guys!

Kamu juga dapat melihat proses job mana yang sedang berjalan, hingga apakah prosesnya berhasil maupun gagal di informasi yang disajikan oleh pipeline.

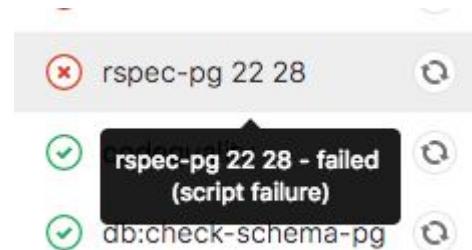




Kayak Gofood, Jobs juga punya informasi status untuk penggunanya

Berikut status yang terdapat pada jobs yang akan diinformasikan di pipeline :

1. warning
2. pending
3. running
4. manual

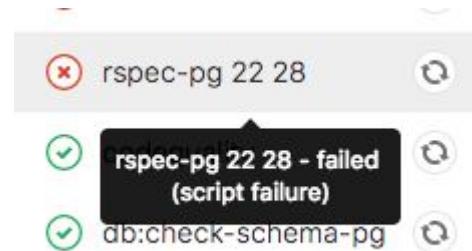




Lanjutannya~

Berikut status yang terdapat pada Jobs yang akan diinformasikan di pipeline:

5. scheduled
6. canceled
7. success
8. skipped
9. created



Saatnya kita Quiz!





1. Apa tujuan dari dilakukan CI/CD?..

- A. Agar menjaga keamanan aplikasi
- B. Mempercepat proses yang integrasi dan delivery
- C. Memastikan aplikasi pass test



2. Pernyataan dibawah ini yang benar adalah..

- A. CI/CD akan menambah waktu proses development yang terus menerus
- B. Memakai CI/CD untuk project yang skala lebih besar lebih efisien dan efektif
- C. Tidak memerlukan .git untuk melakukan ci/cd



3. Apa fungsi dari jobs yang terdapat dalam script .gitlab-yml?

- A. Menjalankan Perintah yang akan tertrigger saat melakukan commit-push
- B. Menyimpan hasil history deployment serta informasi berhasil dan gagal
- C. menampilkan tahapan - tahapan pada uji code



4. Berikut ini yang bukan merupakan versioning control adalah...

- A. Firebase
- B. Bitbucket
- C. Gitlab



1

Apa tujuan dari dilakukan CI/CD?..

- A Agar menjaga keamanan aplikasi
- B Mempercepat proses yang integrasi dan delivery
- C Memastikan aplikasi pass test



2

Pernyataan dibawah ini yang benar adalah..

- A CI/CD akan menambah waktu proses development yang terus menerus
- B Memakai CI/CD untuk project yang skala lebih besar lebih efisien dan efektif
- C Tidak memerlukan .git untuk melakukan ci/cd



3

Apa fungsi dari jobs yang terdapat dalam script .gitlab-yml?

- A Menjalankan Perintah yang akan tertrigger saat melakukan commit-push
- B Menyimpan hasil history deployment serta informasi berhasil dan gagal
- C menampilkan tahapan - tahapan pada uji code



4

Berikut ini yang bukan merupakan versioning control adalah ?

- A Firebase
- B Bitbucket
- C Gitlab

Terima Kasih!



Next Topic

loading...