

KOLEKTOR PENGUMUMAN INFORMATIKA

ELLENA ANGELICA—2015730029

1 Data Skripsi

Pembimbing utama/tunggal: **Pascal Alfadian Nugroho**

Pembimbing pendamping: -

Kode Topik : **PAN4504**

Topik ini sudah dikerjakan selama : **1 semester**

Pengambilan pertama kali topik ini pada : Semester **45 - Ganjil 18/19**

Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**

Tipe Laporan : **B -** Dokumen untuk reviewer pada presentasi dan **review Skripsi 1**

2 Latar Belakang

Pengumuman di jurusan Teknik Informatika UNPAR pada umumnya dilakukan lewat email. Pengumuman lewat email ini praktis karena tidak perlu menunggu email sampai ke tujuan dan dijamin sampai ke tujuan. Selain itu, konten yang disampaikan melalui email fleksibel. Konten tidak harus hanya tulisan tapi dapat ditambah dengan lampiran, dapat diubah gaya tulisannya, dan lain-lain. Namun, email kurang terorganisir dengan baik. Email yang masuk dapat tercampur dengan email lain sehingga mahasiswa kesulitan mencari email yang penting. Dampaknya, pengumuman-pengumuman penting dapat tidak terbaca secara tidak sengaja.

Pada skripsi ini, akan dibuat solusi masalah tadi dengan membangun suatu fitur. Fitur ini akan menangkap email-email pengumuman yang masuk ke sebuah email khusus untuk menangkap pengumuman. Pertama, email yang masuk ke email khusus akan diperiksa pengirimnya. Apabila pengirim adalah email yang terdaftar sebagai email yang berhak melakukan pengumuman, maka email tersebut adalah email pengumuman. Setelah itu, email tersebut akan dibuatkan permanent link dan disisipkan pada basis data. Lalu, mahasiswa akan menerima permanent link tersebut melalui notifikasi dari akun Line@. Line@ adalah layanan dari Line Corporation yang memudahkan pemilik bisnis atau organisasi menyampaikan pesan kepada pengikutnya melalui aplikasi pengirim pesan LINE.

Fitur ini akan dibangun sebagai fitur tambahan pada BlueTape, sebuah website milik jurusan teknik Informatika Unpar. Pembangunan fitur ini membutuhkan modifikasi BlueTape sehingga dapat dijalankan di Heroku. Heroku adalah *cloud platform* yang memungkinkan *developer* untuk membangun, menjalankan, dan mengoperasikan aplikasi pada *cloud*. Selain itu, fitur ini membutuhkan beberapa fitur dari PHP IMAP dan layanan pengirim pesan LINE.

3 Rumusan Masalah

- Bagaimana cara memodifikasi BlueTape agar fitur kolektor pengumuman dapat diimplementasikan dengan bantuan Heroku dan PostgreSQL ?
- Bagaimana cara mengimplementasikan kolektor pengumuman pada BlueTape ?

4 Tujuan

- Melakukan perawatan pada BlueTape agar fitur kolektor pengumuman dapat diimplementasikan dengan bantuan Heroku dan PostgreSQL

- Mengimplementasikan fitur kolektor pengumuman pada BlueTape

5 Detail Perkembangan Pengerjaan Skripsi

Detail bagian pekerjaan skripsi sesuai dengan rencana kerja/laporan perkembangan terakhir :

1. Melakukan studi literatur tentang Heroku, Gmail, PHP IMAP, dan LINE

Status : Ada sejak rencana kerja skripsi, kecuali PHP IMAP

Hasil : Studi literatur tentang Heroku, Gmail, PHP IMAP, dan LINE telah selesai dilakukan. Hasil dari studi literatur :

(a) Heroku

Heroku adalah *cloud platform* yang memungkinkan *developer* untuk membangun, menjalankan, dan mengoperasikan aplikasi pada *cloud*. Heroku mendukung beberapa bahasa pemrograman, meliputi : Ruby, Node.js, Java, Python, Clojure, Scala, Go, dan PHP.

Arsitektur Heroku

Arsitektur Heroku terdiri dari :

- **Aplikasi**

Heroku mendefinisikan aplikasi sebagai gabungan dari *source code* yang ditulis di dalam salah satu bahasa yang didukung Heroku, deskripsi *dependency* yang dipakai, dan Procfile.

- **Dependency**

Developer perlu mendeskripsikan *dependency* tambahan yang diperlukan agar aplikasi dapat dibangun dan dijalankan. Aturan penulisan deskripsi *dependency* berbeda-beda untuk tiap bahasa. Contoh : pada aplikasi dengan bahasa Node.js, deskripsi *dependency* ditulis di dokumen `package.json`.

- **Procfile**

Developer perlu memberitahu Heroku bagian aplikasi yang dapat dijalankan. Jika *developer* menggunakan framework yang sudah ada, Heroku dapat mencari tahu. Contoh : pada aplikasi dengan bahasa Node.js, Heroku dapat mengetahui bagian aplikasi yang dijalankan pada bagian `main` di dalam `package.json`. Untuk aplikasi lain, *developer* mungkin perlu menyatakan apa yang harus dieksekusi secara eksplisit. Caranya dengan menyertakan sebuah dokumen teks bernama Procfile.

Dokumen Procfile tidak memiliki ekstensi dokumen, seperti `.txt`, `.docx`, `.jpg` dan lain-lain. Apabila Procfile diberi ekstensi dokumen (contoh : `Procfile.txt`), maka Procfile tersebut tidak sah. Selain itu, Procfile harus diletakkan di direktori `root`. Jika diletakkan di tempat lain, Procfile tidak akan berfungsi sebagaimana mestinya.

Isi dari Procfile adalah satu atau lebih baris yang menyatakan *process type*. Format penulisan tiap baris Procfile adalah :

```
<process type>: <command>
```

Keterangan :

- `<process type>` : nama perintah yang mengandung huruf dan angka. Contoh : `web`, `worker`, `urgentworker`, `clock`, dan lain-lain. Untuk aplikasi sederhana, *developer* cukup menuliskan *process type* `web` saja.

- `<command>` : perintah yang setiap dyno dari *process type* tersebut harus jalankan pada saat startup.

Contoh isi Procfile :

```
web: java -jar lib/foobar.jar \textdollar PORT
```

Pada contoh, `web` merupakan `<process type>`, sedangkan `web: java -jar lib/foobar.jar $PORT` adalah perintah yang harus dijalankan agar *process type* tersebut berjalan. Perintah tersebut berfungsi untuk menyalakan web server.

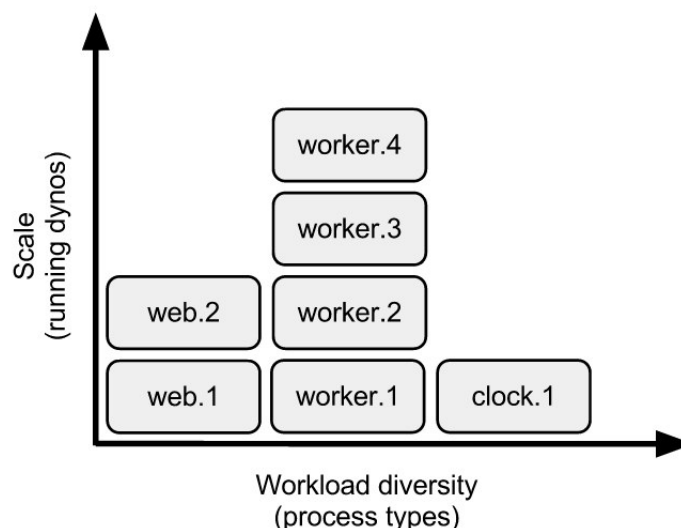
Procfile tidak wajib dibuat untuk sebagian besar bahasa pemrograman yang didukung Heroku. Heroku akan secara otomatis mendeteksi bahasa yang digunakan, dan membuat *process type* `web` untuk menjalankan server aplikasi. Apabila aplikasi menggunakan `heroku.yml` sebagai *build manifest*, Procfile juga tidak diwajibkan. Perintah yang disebutkan di bagian `run` pada `heroku.yml` harus mengikuti format yang sama dengan format Procfile (kecuali *process type* `release`).

• *Process Type*

Ada tiga kelompok *process type* :

- *process type* `web`
- *process type* `worker` : *process type* apapun selain `web`
- *process type* `singleton` : *process type* yang bersifat sementara dan dapat berjalan terpisah

Di antara beragam *process type*, ada dua *process type* spesial : *process type* `web` dan `release`. *Process type* `web` adalah satu-satunya *process type* yang dapat menerima arus HTTP eksternal dari router Heroku. Jika sebuah aplikasi melibatkan web server, *developer* harus menyatakannya sebagai proses `web`. *Process type* `release` adalah *process type* yang digunakan untuk menyebutkan perintah yang dijalankan selama fase `release`.



Gambar 1: Diagram hubungan antara tipe proses dan dyno

Process type dan dyno saling berhubungan. *Process type* adalah prototipe yang menjadi tempat dimana dyno dibentuk. Hubungan *process type* dan dyno dapat dilihat di diagram pada Gambar 1. Sumbu x menyatakan *process type* yang dipakai, sementara sumbu y menyatakan jumlah dyno yang berjalan pada *process type* tersebut. Semakin banyak dyno

pada suatu *process type* maka konkurensi untuk pekerjaan yang ditangani *process type* tersebut akan meningkat. Semakin banyak *process type* maka semakin beragam beban kerja. Untuk mengatur berapa banyak dyno yang bekerja di satu *process type*, perintah yang dapat diketikkan pada command shell adalah :

```
$ heroku ps:scale <process=dyno list>
```

Keterangan :

- `<process=dyno list>` : daftar pasangan *process type* dengan jumlah dyno yang ditugaskan untuk proses tersebut.

Contoh :

```
$ heroku ps:scale web=2 worker=4 clock=1
```

Selain dapat mengatur jumlah dyno yang ditugaskan pada suatu pekerjaan, *developer* dapat menjadwalkan proses yang berjalan pada suatu waktu atau jangka waktu tertentu. Caranya dengan menggunakan add-on Heroku Scheduler atau menggunakan *process type* khusus untuk mengatur jadwal pekerjaan.

• Dyno

Dyno adalah wadah aplikasi berbasis Unix yang terisolasi, tervirtualisasi, dan menyediakan lingkungan yang dibutuhkan untuk menjalankan suatu aplikasi. Umumnya, jika aplikasi di-*deploy* ke Heroku untuk pertama kali, Heroku akan menjalankan satu web dyno secara otomatis.

Setiap dyno termasuk dalam salah satu dari konfigurasi berikut :

- Web dyno
Web dyno adalah dyno dari *process type* **web** yang disebutkan di dalam Procfile. Web dyno adalah satu-satunya dyno yang dapat menerima arus HTTP dari router Heroku.
- Worker dyno
Worker dyno adalah dyno dari *process type* apapun selain **web** yang disebutkan di dalam Procfile. Worker dyno biasanya digunakan untuk pekerjaan di latar belakang, sistem antrian, dan pekerjaan yang memiliki jangka waktu.
- One-off dyno
One-off dyno adalah dyno yang bersifat sementara yang dapat berjalan terpisah atau dengan masukan/keluaran dari terminal lokal. Dyno ini dapat digunakan untuk tugas yang bersifat administratif, contoh : migrasi basis data. Dyno ini juga dapat digunakan untuk melakukan pekerjaan di latar belakang yang bersifat sesekali, contoh : Heroku Scheduler.

Heroku menyediakan beberapa tipe dyno yang berbeda. Tiap tipe dyno memiliki sifat yang unik dan kinerja yang berbeda. Untuk semua pengguna Heroku, tersedia pilihan dyno tipe Free, Hobby, Standard, dan Performance. Ada satu tipe dyno lagi, yaitu tipe Private. Dyno tipe Private hanya tersedia di Heroku Enterprise yang diperuntukkan untuk organisasi.

Fitur-fitur utama dyno adalah :

- *Scalability*
Dyno dapat di-*scale* secara horizontal dan vertikal. Untuk melakukan *scale* secara horizontal (*scale out*), tambahkan lebih banyak dyno. Contoh : menambah web dyno agar dapat

menangani arus yang lebih besar. Untuk melakukan *scale* secara vertikal (*scale up*), gunakan dyno yang lebih besar. Dyno yang lebih besar berarti jumlah pemakaian memori RAM yang lebih besar. Jumlah RAM maksimal yang tersedia untuk aplikasi tergantung dari tipe dyno yang digunakan. *Scale* secara horizontal dan vertikal ini adalah fitur yang tersedia untuk dyno tipe Standard dan Performance saja.

– *Redundancy*

Aplikasi dengan banyak dyno yang berjalan akan memiliki resiko kegagalan yang lebih rendah daripada yang sedikit. Jika ada dyno yang hilang, aplikasi dapat terus memproses permintaan sementara dyno yang hilang diganti. Dyno yang hilang biasanya langsung dimulai ulang, tapi terkadang membutuhkan waktu yang lama.

– *Isolation and security*

Semua dyno terisolasi dari dyno lain untuk alasan keamanan. Walaupun dyno tipe Free, Hobby, dan Standard terisolasi, dyno mungkin berbagi komputasi dasar yang sama. Heroku memiliki teknik tersendiri untuk memastikan penggunaannya adil. Di sisi lain, dyno tipe Performance dan Private tidak berbagi komputasi dasar yang sama dengan dyno lain. Hal ini membuat dyno tipe Performance dan Private memiliki kinerja yang lebih stabil dibanding dengan dyno tipe Free, Hobby, dan Standard. Selain memiliki sumber daya komputasi yang dikhususkan untuknya, dyno tipe Private juga memiliki jaringan virtual yang terisolasi.

– *Ephemeral filesystem*

Tiap dyno memiliki *ephemeral filesystem*, dengan salinan kode dari hasil deploy terbaru. Saat masa hidup dyno, proses yang dijalankannya dapat menggunakan *filesystem* ini sebagai tempat menulis sementara. Namun, dokumen yang ditulis tidak dapat dilihat oleh proses dari dyno lain dan dokumen yang ditulis akan dihapus saat dyno berhenti bekerja atau dimulai ulang.

• **Dyno manager**

Dyno manager adalah bagian dari Heroku yang bertanggungjawab untuk menjaga dyno tetap berjalan. Dyno manager melakukan pekerjaan seperti memastikan dyno didaur ulang setidaknya satu kali sehari atau setiap dyno manager mendeteksi kesalahan di dalam aplikasi yang berjalan. Daur ulang dyno ini berlangsung secara transparan dan otomatis secara teratur dan tercatat.

Aplikasi yang menggunakan dyno tipe Free akan masuk mode **sleep** (tidur) jika tidak ada arus HTTP selama jangka waktu 30 menit. Ketika aplikasi yang tidur menerima arus HTTP, maka aplikasi tersebut akan terbangun. Hal ini menyebabkan aplikasi lebih lambat beberapa detik dari aplikasi yang menggunakan dyno tipe lain. Dyno tipe lain tidak memiliki mode sleep, dan akan selalu terjaga.

• **Config vars**

Konfigurasi aplikasi dapat berubah-ubah tergantung lingkungannya. Misalnya, konfigurasi aplikasi saat pengembangan dapat berbeda saat aplikasi siap dirilis ke pengguna. Konfigurasi aplikasi dapat berupa informasi *database*, informasi kredensial, atau informasi lain yang bersifat spesifik pada aplikasi. Konfigurasi ini harus diletakkan pada *environment variable*, bukan di *source code*. Dengan menggunakan *environment variable*, konfigurasi dapat diubah secara terpisah. Selain itu, konfigurasi yang bersifat kredensial dapat terhindar dari tersimpan pada *version control* (pengontrol versi, contoh : Git).

Heroku memungkinkan *developer* untuk menjalankan aplikasi dengan konfigurasi yang dapat diubah dengan mudah. Konfigurasi tersebut diletakkan di luar dari *source code* aplikasi. Konfigurasi dapat diubah secara independen tanpa harus mengubah *source code*. Konfigurasi

tersebut disimpan di dalam config vars.

Untuk mengatur config vars ada tiga cara :

- Menggunakan Heroku CLI

Config var diatur menggunakan *command shell*. Berikut perintah-perintah untuk mengatur config var menggunakan Heroku CLI:

- Menampilkan seluruh config var beserta nilainya :

```
$ heroku config
```

- Menampilkan nilai dari config var tertentu

```
$ heroku config : get <config var>
```

Keterangan : **config var** : nama config var

- Menambah config var

```
$ heroku config : set <config var> = <config value>
```

Keterangan :

* **config var** : nama config var

* **config value** : nilai dari config var tersebut

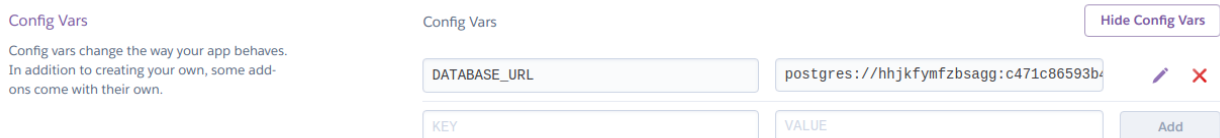
- Menghapus config var

```
$ heroku config : unset <config var>
```

Keterangan :

* **config var** : nama config var

- Menggunakan Heroku Dashboard



Gambar 2: Config vars pada dashboard Heroku

Config var dapat dilihat, ditambah, dan dihapus melalui menu **Settings** bagian config vars (Gambar 2).

- Menggunakan Heroku Platform API

Config var dapat diatur dengan Heroku Platform API menggunakan HTTPS REST client sederhana dan data struktur data JSON. *Developer* perlu Heroku *access token* yang valid yang mewakili pengguna dengan izin yang tepat untuk aplikasi.

Config vars akan diperlakukan sebagai *environment variable* oleh program. Contoh : *environment variable* dapat diakses dengan `getEnv()` pada program dengan bahasa PHP.

Dalam mengatur config var, ada beberapa hal yang harus diperhatikan :

- Setiap config var ditambah atau dihapus, aplikasi akan dimulai ulang dan release baru akan dibuat.
- Jika aplikasi menggunakan add-on, biasanya add-on tersebut akan menambahkan satu atau lebih config var ke aplikasi. Nilai dari config var tersebut mungkin diperbarui oleh penyedia add-on kapan saja.
- Config var data (kombinasi dari semua kunci dan nilainya) tidak dapat melebihi 32kb per aplikasi
- Nama config var tidak boleh diawali dengan garis bawah dua kali (`__`).
- Nama config var tidak bisa diawali dengan `HEROKU_`, kecuali ditambahkan oleh platform Heroku sendiri.

• Add-ons

Aplikasi biasanya memanfaatkan add-ons untuk menyediakan layanan penyokong seperti basis data, sistem antrian, layanan email, dan lainnya. Add-ons disediakan oleh Heroku atau pihak ketiga. *Developer* dapat mencari add-ons di Elements Marketplace (<https://elements.heroku.com/addons>).

Menambah add-ons selain add-ons Heroku Postgres dan Heroku Connect membutuhkan verifikasi akun. *Developer* dapat menambah add-ons melalui tombol Install di Elements Marketplace atau dengan mengetikkan perintah berikut pada *command shell* :

```
$ heroku addons:create <nama addons>:<tipe addons>
```

Keterangan :

- `<nama addons>` : nama addons
- `<tipe addons>` : tipe addons

Contoh :

```
$ heroku addons:create heroku-redis:hobby-dev
```

• Slug

Ketika platform Heroku menerima *source code* aplikasi, heroku akan memulai proses *build* berdasarkan *source code*. Mekanisme *build* biasanya tergantung pada bahasa pemrograman yang dipakai, tapi mengikuti pola yang sama. Mekanisme *build* biasanya mengambil *dependency* yang ditentukan, dan menciptakan aset yang diperlukan. *Source code* untuk aplikasi, *dependency*, dan hasil dari fase *build* digabungkan ke dalam slug.

Slug adalah gabungan dari *source code*, *dependency* yang diambil, *language runtime*, dan hasil kompilasi atau keluaran yang dihasilkan oleh *build system* yang siap untuk dieksekusi. Slug ini adalah aspek dasar dari eksekusi aplikasi. Slug berisi aplikasi yang sudah dikompilasi, digabungkan, dan siap untuk dijalankan.

Slug dikompilasi oleh slug compiler menggunakan buildpack. Buildpack akan mengambil aplikasi, *dependency*, dan *language runtime* dan kemudian menghasilkan slug. Buildpack bersifat *open source*, sehingga memungkinkan *developer* memperluas Heroku ke bahasa pemrograman lain dan *framework*.

Apabila ada dokumen yang tidak diperlukan untuk menjalankan aplikasi, *developer* dapat menambahkannya ke `.slugignore`. Dokumen ini harus dibuat di direktori `root`. Contoh dokumen yang mungkin ingin dimasukkan ke `.slugignore` :

- Dokumen pengolah gambar (contoh : dokumen .psd)
- Dokumen desain (contoh : dokumen .pdf)
- Data untuk pengujian

Contoh isi `.slugignore` :

```
# Heres a comment
*.psd
*.pdf
/test
/spec
```

Ukuran slug dapat terlihat di akhir kompilasi (apabila kompilasi berhasil). Maksimum ukuran slug adalah 500 MB. Ukuran slug bervariasi berdasarkan bahasa atau *framework* yang digunakan, banyak *dependency* yang ditambahkan, dan faktor lain dari aplikasi. Slug yang ukurannya lebih kecil dapat ditransfer ke dyno manager dengan lebih cepat.

• Buildpack

Buildpack bertanggung jawab untuk mengubah *source code* menjadi slug, sehingga dyno dapat mengeksekusinya. Buildpack terdiri dari sekumpulan *script* yang ditulis dalam bahasa pemrograman yang sama dengan *source code*. Script tersebut akan mengambil *dependency*, mengeluarkan aset atau kode yang sudah dikompilasi, dan sebagainya. Keluaran ini akan digabungkan ke dalam slug oleh slug compiler.

Heroku memiliki sekumpulan *officially supported buildpack* yang tersedia secara default untuk semua aplikasi Heroku selama kompilasi slug. Daftar *officially supported buildpack* terdapat pada Gambar 3. Kolom **Buildpack** menyatakan nama buildpack dan kolom **shorthand** menyatakan nama panggil buildpack saat di CLI.

Buildpack	Shorthand
Ruby	heroku/ruby
Node.js	heroku/nodejs
Clojure	heroku/clojure
Python	heroku/python
Java	heroku/java
Gradle	heroku/gradle
Grails 3.x	heroku/gradle
Scala	heroku/scala
Play 2.x	heroku/scala
PHP	heroku/php
Go	heroku/go

Gambar 3: Tabel buildpack heroku

Heroku akan mencari buildpack yang sesuai dan menggunakannya untuk mengompilasi aplikasi. Jika build sukses, buildpack yang sudah terdeteksi sesuai akan secara permanen diatur untuk push selanjutnya. Buildpack yang telah dimodifikasi dapat dipakai untuk mendukung bahasa atau framework yang tidak dapat di cakup oleh buildpack resmi.

Biasanya buildpack yang dipakai oleh aplikasi hanya satu, tapi ada beberapa kasus buildpack yang dipakai tidak cukup hanya satu. Beberapa kasus tersebut adalah :

- Menjalankan buildpack untuk tiap bahasa pemrograman yang aplikasi gunakan. Contohnya, menjalankan JavaScript buildpack untuk aset dan buildpack Ruby untuk aplikasi.
- Menjalankan proses daemon seperti `pgbouncer` dengan aplikasi.
- Menarik *dependency* sistem dengan `apt`.

Berikut adalah perintah-perintah dasar buildpack yang dapat diketikkan pada *command shell* :

- Mengatur buildpack yang dipakai saat aplikasi pertama kali dibuat

```
$ heroku create myapp --buildpack <nama buildpack>
```

Keterangan :

* `<nama buildpack>` : nama panggil buildpack yang ingin dipakai, contoh : `heroku/php`.

Buildpack juga dapat secara eksplisit diatur di dalam `app.json` sehingga aplikasi yang dibuat menggunakan tombol Heroku dapat menggunakan buildpack yang telah dimodifikasi.

- Mengubah dengan mengatur nilai buildpack

```
$ heroku buildpacks:set <nama buildpack>
```

Keterangan :

* `<nama buildpack>` : nama panggil buildpack yang ingin dipakai, contoh : `heroku/php`.

- Menghilangkan buildpack dari aplikasi

```
$ heroku buildpacks:remove <nama buildpack>
```

Keterangan :

* `<nama buildpack>` : nama panggil buildpack yang ingin dipakai, contoh : `heroku/php`.

- Mencari buildpack

```
$ heroku buildpacks:search <kata kunci>
```

Keterangan :

* `<kata kunci>` : kata kunci pencarian, misalnya : bahasa pemrograman yang dipakai.
Contoh : `elixir`.

- Menampilkan informasi buildpack

```
$ heroku buildpacks:info <nama buildpack>
```

Keterangan :

* **<nama buildpack>** : nama panggil buildpack yang ingin dipakai, contoh : **heroku/php**.

- Mengembalikan aplikasi ke buildpack awalnya

```
$ heroku buildpacks:clear
```

- Mengatur urutan eksekusi buildpack

```
$ heroku buildpacks:set --index <index> <nama buildpack>
```

Keterangan :

* **<index>** : urutan eksekusi buildpack

* **<nama buildpack>** : nama panggil buildpack yang ingin dipakai, contoh : **heroku/php**.

- Melihat daftar buildpack

```
$ heroku buildpacks
```

• Stack

Stack adalah sistem operasi yang dikelola dan dipelihara oleh Heroku. Stack biasanya berlandaskan distribusi dari Linux yang ada, seperti Ubuntu. *Developer* dapat menentukan stack yang dipakai, dan buildpack akan mengubah *source code* menjadi paket yang dapat dieksekusi dengan stack tersebut. Saat skripsi ini dibuat, Heroku menyediakan tiga stack : Cedar-14, Heroku-16, dan Heroku-18. Cedar-14 berbasis Ubuntu 14.04 dan didukung sampai bulan April tahun 2019. Heroku-16 berbasis Ubuntu 16.04 dan didukung sampai bulan April tahun 2021. Heroku-18 berbasis Ubuntu 18.04 dan didukung sampai bulan April tahun 2023. Semua buildpack dari Heroku dapat bekerja dengan ketiga stack tersebut, namun buildpack yang merupakan hasil modifikasi belum tentu dapat bekerja dengan semua stack.

Untuk melihat stack yang dipakai oleh aplikasi, *developer* dapat mengetikkan perintah berikut pada *command shell*:

```
$ heroku stack
```

Untuk mengganti stack yang dipakai, *developer* dapat mengetikkan perintah berikut pada *command shell*:

```
$ heroku stack:set <stack>
```

Keterangan :

- <stack> dapat diisi dengan cedar-14, heroku-16, atau heroku-18.

• Region

Aplikasi di dalam Heroku dapat disebarkan ke lokasi geografis yang berbeda. Lokasi yang tersedia untuk suatu aplikasi tergantung pada Runtime yang dipakai oleh aplikasi (Common Runtime atau Private Space). Untuk aplikasi yang memakai Common Runtime, *developer* perlu menyebutkan region aplikasi saat membuat aplikasi. Untuk aplikasi yang memakai Private Space, region diatur saat membuat Private Space. Apabila *developer* tidak menyebutkan region yang dipakai, maka region akan diisi secara otomatis sebagai *us* (apabila memakai Common Runtime) atau *virginia* (apabila memakai Private Spaces).

Region berpengaruh terhadap add-ons. Apabila add-ons tidak tersedia di region yang sama dengan aplikasi, maka add-ons akan gagal terpasang. Region juga dapat mempengaruhi cara kerja SSL.

Berikut perintah-perintah dasar region yang dapat diketikkan pada *command shell* :

- Memeriksa region yang tersedia di Heroku

```
$ heroku regions
```

- Mengatur region aplikasi

```
$ heroku create --region <id region>
```

Keterangan :

- * <id region> : id region yang ingin dipakai, contoh : *eu*. Id region bisa dilihat dengan memeriksa daftar region yang tersedia.
- Memeriksa region yang dipakai oleh aplikasi

```
$ heroku info
```

• Releases

Setiap ada deploy baru, perubahan di config vars, dan perubahan di daftar add-ons, Heroku akan membuat release baru dan memulai ulang aplikasi. Releases adalah buku besar yang mencatat setiap release tersebut. *Developer* dapat melihat catatan releases ini dengan menggunakan perintah :

```
$ heroku releases
```

Isi dari releases adalah satu atau lebih baris dari release yang tiap barisnya memiliki format :

```
<versi deploy> Deploy <commit hash> <username \textit{developer}> <
    waktu deploy>
```

Contoh isi releases :

```
== demoapp Releases
v103 Deploy 582fc95 jon@heroku.com 2013/01/31 12:15:35
v102 Deploy 990d916 jon@heroku.com 2013/01/31 12:01:12
```

Releases ini berguna saat *developer* ingin mengembalikan aplikasi ke deploy lama. Cara mengembalikan aplikasi ke deploy lama dengan mengetikkan perintah berikut pada *command shell*:

```
$ heroku releases:rollback <version>
```

Keterangan :

- <version> : versi deploy

Contoh :

```
$ heroku releases:rollback v102
```

• Log

Log adalah catatan setiap proses yang terjadi di aplikasi. Heroku menggunakan Logplex untuk menyampaikan log ini. Logplex akan secara otomatis menambahkan entri log baru dari semua dyno yang berjalan di aplikasi, dan juga komponen lain seperti router. *Developer* dapat memeriksa log dengan cara mengetikkan perintah berikut pada *command shell*:

```
$ heroku logs
```

Contoh isi log adalah :

```
2013-02-11T15:19:10+00:00 heroku[router]: at=info method=GET path=/
    articles/custom-domains host=mydemoapp.herokuapp.com fwd
    =74.58.173.188 dyno=web.1 queue=0 wait=0ms connect=0ms service
    =1452ms status=200 bytes=5783
2013-02-11T15:19:10+00:00 app[web.2]: Started GET "/" for
    1.169.38.175 at 2013-02-11 15:19:10 +0000
2013-02-11T15:19:10+00:00 app[web.1]: Started GET "/" for
    2.161.132.15 at 2013-02-11 15:20:10 +0000
```

Deploy perangkat lunak

Heroku menggunakan Git sebagai sarana utama untuk melakukan deploy aplikasi. Deploy adalah proses penyebaran aplikasi dari satu lingkungan ke lingkungan lain, misalnya dari lingkungan mesin *developer* aplikasi ke lingkungan heroku. Namun, Heroku juga menyediakan cara lain untuk melakukan deploy :

- Docker
- GitHub
- Tombol Deploy di dashboard Heroku
- WAR deployment

i. Deploy Menggunakan Git

Untuk melakukan deploy menggunakan Git, *developer* harus sudah memasang Git. *Developer* dapat mengikuti petunjuk unduhan pada <https://git-scm.com>. Sebelum *developer* dapat melakukan deploy dengan Git, *developer* perlu menginisialisasi git. Berikut perintah-perintah yang harus dijalankan pada *command shell* :

```
$ git init
$ git add .
$ git commit -m "<message>"
```

Keterangan :

- **<message>** : pesan yang mewakili commit.

Setelahnya, *developer* dapat membuat aplikasi Heroku. Setiap aplikasi Heroku dibuat, maka **git remote** secara otomatis juga dibuat. *Developer* dapat memeriksanya dengan mengetikkan perintah berikut pada *command shell* :

```
$ git remote // Untuk daftar nama remote saja
$ git remote -v // Untuk informasi yang lebih detail
```

Untuk mengubah nama remote, *developer* dapat mengetikkan perintah berikut pada *command shell* :

```
$ git remote rename <nama lama> <nama baru>
```

Keterangan :

- **<nama lama>** : nama remote yang ingin diganti.
- **<nama baru>** : nama baru untuk remote tersebut.

Untuk melakukan deploy, *developer* dapat mengetikkan perintah berikut pada *command shell* :

```
$ git push <nama remote> <nama branch>
```

Keterangan :

- `<nama remote>` : nama remote dari tujuan deploy. Bila *developer* tidak mengubah nama remote, nama remotenya adalah **heroku**.
- `<nama branch>` : nama cabang dari tujuan deploy. Heroku secara otomatis membuat satu cabang bernama **master**.

ii. Deploy Menggunakan Docker

Untuk melakukan deploy menggunakan Docker, *developer* harus sudah memasang Docker dan telah masuk ke akun Heroku (**heroku login**). Setelah itu, *developer* harus mengikuti langkah-langkah ini :

- Masuk ke Container Registry

```
$ heroku container:login
```

- Clone source code contoh dari Alpine

```
$ git clone https://github.com/heroku/alpinehelloworld.git
```

- Membuat aplikasi Heroku baru

```
$ heroku create
```

- Membangun image dan melakukan deploy ke Container Registry

```
$ heroku container:push web
```

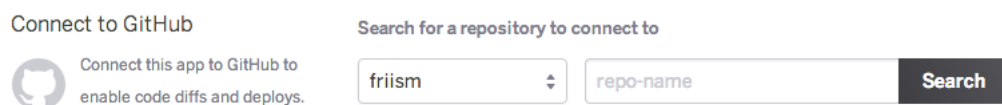
- Melepaskan image ke aplikasi

```
$ heroku container:release web
```

- Membuka aplikasi

```
$ heroku open
```

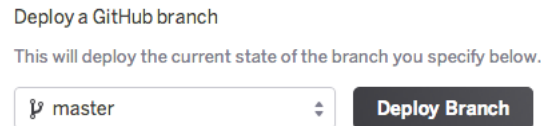
iii. Deploy Menggunakan GitHub



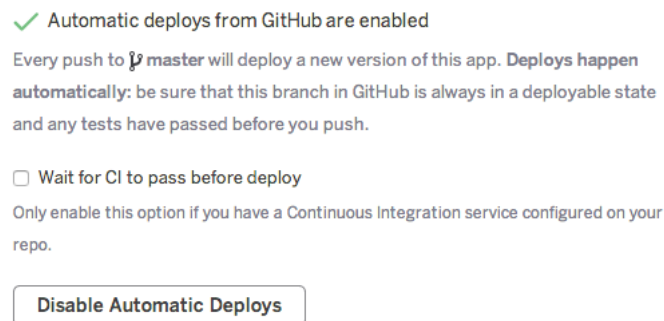
Gambar 4: Deploy menggunakan Github Dashboard

Deploy dengan cara ini membuat Heroku dapat dengan otomatis melakukan deploy ke GitHub apabila build berhasil. *Developer* perlu mengaktifkan GitHub integration terlebih dahulu

sebelum dapat melakukan deploy. Setelah itu, *developer* harus melakukan autentikasi dengan akun GitHub. Autentikasi ini hanya perlu dilakukan satu kali per satu akun Heroku. Setelah itu, *developer* dapat memilih repository yang ingin disambungkan dengan aplikasi Heroku (Gambar 4).



Gambar 5: Deploy menggunakan Github secara manual



Gambar 6: Deploy menggunakan Github secara otomatis

Ada dua cara untuk melakukan deploy, yaitu secara manual dan secara otomatis. Untuk cara manual, *developer* melakukan deploy dari GitHub (Gambar 5). Untuk cara otomatis, *developer* harus mengaktifkan "Automatic deploys from GitHub" (Gambar 6).

iv. Deploy Langsung di situs Heroku

Tombol "Deploy to Heroku" memungkinkan pengguna untuk melakukan deploy aplikasi tanpa meninggalkan situs Heroku dan hampir tidak memerlukan konfigurasi. Penggunaan tombol ini ideal untuk pelanggan, dan pemelihara proyek yang bersifat open-source. Sebelum dapat melakukan deploy dengan cara ini, aplikasi harus memiliki dokumen `app.json` yang sah di direktori root, dan source code aplikasi harus berada di repository GitHub.

`app.json` adalah dokumen berisi deskripsi aplikasi web. Isinya dapat berupa environment variable, add-ons, dan informasi lain yang diperlukan untuk menjalankan aplikasi pada Heroku. Heroku tidak mewajibkan *developer* menuliskan informasi tertentu, tapi Heroku merekomendasikan untuk setidaknya menuliskan nama aplikasi (`name`), deskripsi aplikasi (`description`), dan logo aplikasi (`logo`). Berikut contoh isi dari `app.json` :

```
{
  "name": "Node.js Sample",
  "description": "A barebones Node.js app using Express 4",
  "repository": "https://github.com/heroku/node-js-sample",
  "logo": "https://node-js-sample.herokuapp.com/node.png",
  "keywords": ["node", "express", "static"]
}
```

v. WAR Deployment

WAR (Web Application ARchive) adalah jenis dokumen arsip yang digunakan untuk membungkus aplikasi web. Dokumen ini dapat berisi halaman web statis, dokumen XML, dan lain-lain.

Heroku mendukung deploy dokumen WAR melalui Git deployment dan melalui Heroku Maven plugin. Setelan standar server untuk keduanya adalah Tomcat 8.

Basis data dan manajemen data

Heroku menyediakan tiga layanan data untuk semua pelanggan :

- **Heroku Postgres**

Heroku Postgres adalah basis data SQL yang disediakan secara langsung oleh Heroku. Heroku Postgres dapat diakses oleh bahasa apapun dengan PostgreSQL driver. Heroku secara otomatis menambahkan add-ons Heroku Postgres setiap aplikasi dibuat, sehingga *developer* tidak perlu menambahkannya secara manual. Namun, *developer* dapat menambahkannya secara manual, dengan mengetikkan perintah berikut pada *command shell* :

```
$ heroku addons:create heroku-postgresql:<PLAN_NAME>
```

Keterangan :

- <PLAN_NAME> : nama plan Heroku Postgres yang ingin dipakai. Heroku secara otomatis menggunakan Heroku Postgres tipe *hobby-dev*.

Heroku Postgres memiliki lima plan :

- Hobby Tier : untuk aplikasi dengan toleransi gagal bekerja sampai 4 jam per bulan.
- Standard Tier : untuk aplikasi dengan toleransi gagal bekerja sampai 1 jam per bulan.
- Premium Tier : untuk aplikasi dengan toleransi gagal bekerja sampai 15 menit per bulan.
- Private Tier : untuk pengguna Heroku Enterprise, memiliki toleransi gagal bekerja sampai 15 menit per bulan.
- Shield Tier : untuk pengguna Heroku Enterprise yang menginginkan basis data yang compliance-capable, memiliki toleransi gagal bekerja sampai 15 menit per bulan.

Gambar 7 menunjukkan tabel perbedaan antara plan. Hanya plan Hobby yang gratis. Plan lain memiliki harga yang bervariasi berdasarkan ukuran RAM, batas penyimpanan, dan batas koneksi yang bisa dibuat.

Heroku Postgres tier	Downtime Tolerance	Fork	Follow	Rollback	HA	Disk Encryption
Hobby	< 4 hr downtime per mo.	No	No	No	No	No
Standard	< 1 hr downtime per mo.	Yes	Yes	4 days	No	Yes
Premium	< 15 min downtime per mo.	Yes	Yes	1 week	Yes	Yes
Private	< 15 min downtime per mo.	Yes	Yes	1 week	Yes	Yes
Shield	< 15 min downtime per mo.	Yes	Yes	1 week	Yes	Yes

Gambar 7: Tabel plan Heroku Postgres

Semua plan memiliki fitur yang sama :

- Dapat mengelola layanan basis data secara menyeluruh dengan fitur *automatic health checks*
- Write-ahead log (WAL) menjauh dari tempat penyimpanan setiap 60 detik, memastikan resiko kehilangan data dan kesalahan lainnya seminimal mungkin

- Backup basis data harian menggunakan PG Backups (opsional, tapi gratis)
- Dataclips untuk berbagi data dan query yang mudah dan aman
- Akses psql/libpq dengan SSL-protected
- Menjalankan Postgres 9.4, 9.5, 9.6, atau 10 tanpa modifikasi
- Ekstensi Postgres
- Fitur web UI (<https://data.heroku.com/>)

Developer juga dapat menambahkan versi yang ingin dipakai dengan cara menambahkan `--version` di belakang perintah tersebut, contoh :

```
$ heroku addons:create heroku-postgresql:<PLAN_NAME--version=9.5
```

Secara otomatis, Heroku menggunakan versi paling baru dari Heroku Postgres. Saat skripsi ini ditulis, versi terbaru adalah versi 10.

Setelah dipasang, Heroku akan secara otomatis menambahkan config var `DATABASE_URL` ke aplikasi. Apabila Heroku Postgres yang dipakai ada lebih dari satu, nama config var akan menjadi `HEROKU_POSTGRESQL_<COLOR>_URL` dengan `<COLOR>` adalah nama warna yang dihasilkan secara acak. Contoh : `HEROKU_POSTGRESQL_<BLUE>_URL`.

Apabila *developer* menggunakan lebih dari satu basis data, *developer* dapat mengatur basis data utama. Basis data utama dapat diatur dengan mengetikkan perintah berikut pada *command shell* :

```
$ heroku pg:promote <database_url>
```

Keterangan :

- `<database_url>` : url dari basis data.

Apabila *developer* ingin berbagi Heroku Postgres kepada banyak aplikasi, *developer* dapat mengetikkan perintah berikut pada *command shell* :

```
$ heroku addons:attach <originating_app>::DATABASE --app <receiver-app>
```

Keterangan :

- `<originating_app>` : nama aplikasi yang memiliki basis data yang ingin dibagi ke aplikasi lain.
- `<receiver-app>` : nama aplikasi yang akan menerima basis data dari aplikasi lain.

Developer dapat berhenti berbagi basis data dengan mengetikkan perintah berikut pada *command shell* :

```
$ heroku addons:detach <database_url> --app <application_name>
```

Keterangan :

- `<database_url>` : url dari basis data

- `<application_name>` : nama aplikasinya.

Berikut adalah perintah-perintah dasar dari Heroku Postgres yang dapat diketikkan pada *command shell*:

- Melihat semua basis data milik aplikasi dan karakteristiknya

```
$ heroku pg:info
```

- Mengawasi status basis data secara terus menerus

```
$ watch heroku pg:info
```

- Mengadakan sesi `psql` dengan basis data

```
$ heroku pg:psql
```

atau

```
$ heroku pg:psql <database_name>
```

Keterangan :

* `<database_name>` diisi dengan nama basis data atau cukup warna basis data (misal : `gray`).

- Menarik data dari basis data Heroku Postgres ke basis data di mesin lokal

```
$ heroku pg:pull
```

- Memasukkan data dari basis data di mesin lokal ke basis data di Heroku Postgres

```
$ heroku pg:push <nama_db_lokal> <nama_db_heroku> --app <
nama_aplikasi>
```

- Melihat daftar query yang berjalan

```
$ heroku pg:ps
```

```
// Contoh hasil :
procpid | source | running_for | waiting | query
-----+-----+-----+-----+-----
 31776 | psql | 00:19:08.017088 | f | <IDLE> in transaction
 31912 | psql | 00:18:56.12178 | t | select * from hello;
 32670 | Heroku Postgres Data Clip | 00:00:25.625609 | f | BEGIN
      READ ONLY; select 'hi'
(3 rows)
```

- Menghentikan query yang berjalan

```
$ heroku pg:kill <procpid>
```

- Menghentikan query yang berjalan secara paksa

```
$ heroku pg:kill --force <procpid>
```

- Menghentikan semua query yang berjalan

```
$ heroku pg:killall
```

- Menghapus semua data di dalam basis data

```
$ heroku pg:reset <nama database>
```

Basis data Heroku Postgres dapat diakses secara langsung oleh komputer. Informasi yang dibutuhkan untuk mengakses data dari komputer dapat dilihat dengan mengetikkan perintah :

```
$ heroku pg:credentials DATABASE
```

atau

```
$ heroku config | grep HEROKU_POSTGRESQL
```

Pada saat akan melakukan koneksi langsung di komputer, *developer* harus memastikan pengaturan `sslmode=require` pada pengaturan SSL.

• Heroku Redis

Heroku Redis adalah basis data berbasis *key-value store* yang bersifat *in-memory*. Heroku dijalankan oleh Heroku dan dikelola sebagai add-on Heroku Redis dapat diakses oleh bahasa apapun dengan Redis driver. Cara memasang add-on Heroku Redis pada *command shell*:

```
$ heroku addons:create heroku-redis: <PLAN_NAME>
```

Keterangan :

- <PLAN_NAME> : tipe Heroku Redis yang ingin dipakai. Heroku Redis memiliki dua tipe : Hobby Dev dan Premium. Hobby Dev gratis, sedangkan Premium berbayar. Perbedaannya terletak pada jumlah memori dan batas koneksi yang dapat dibuat.

Heroku Redis memiliki kelebihan sebagai berikut :

- Memiliki analisa performa yang dapat membantu menemukan masalah basis data dengan mudah
- Heroku dapat diskala sesuai kebutuhan memori dan koneksi.

• Apache Kafka

Apache Kafka adalah salah satu add-on di Heroku yang disediakan oleh Kafka yang berintegrasi penuh dengan Heroku. Apache Kafka dideskripsikan Kafka dideskripsikan oleh Heroku sebagai add-on yang memungkinkan *developer* mendistribusikan aplikasi yang dapat menangani jutaan event dan miliaran transaksi. Kafka didesain untuk memindahkan *ephemeral data* yang sangat besar dengan reliabilitas yang tinggi dan toleran akan kerusakan.

Developer harus memasang Python 2.7, node 8.x, .NET Framework, dan Visual C++ Build Tools terlebih dahulu sebelum memasang Apache Kafka. Setelah itu, *developer* mengetikkan perintah :

```
$ heroku plugins:install heroku-kafka
```

Heroku juga menyediakan pilihan lain untuk pelanggan Heroku Enterprise, yaitu Heroku Connect. Selain itu, Heroku juga memungkinkan penggunaan layanan data dari pihak ketiga. Layanan data dari pihak ketiga ini tersedia sebagai add-ons.

Verifikasi akun

Heroku membutuhkan identitas terpercaya dan kontak dari pengguna. Heroku menganggap mempunyai informasi kartu kredit adalah cara yang paling dapat diandalkan untuk mendapatkan informasi kontak yang terverifikasi. Verifikasi akun juga membantu Heroku untuk menghindari penyalahgunaan.

Verifikasi akun dibutuhkan untuk :

- Menggunakan lebih dari satu dyno di dalam aplikasi.
- Menambah add-on, termasuk yang gratis. Pengecualian untuk Heroku Postgres dan Heroku Connect.
- Mengubah domain aplikasi.
- Menerima transfer dari aplikasi yang memiliki sumber daya berbayar.
- Menambah batas standar penggunaan one-off dyno.
- Memiliki lebih dari 5 aplikasi dalam satu waktu. Akun yang terverifikasi dapat memiliki sampai 100 aplikasi.

Cara melakukan verifikasi akun Heroku :

- Pergi ke Account Settings (<https://dashboard.heroku.com/account>)
- Menekan tab Billing
- Menekan tombol Add Credit Card

Kartu kredit yang diterima oleh Heroku adalah kartu Visa, MasterCard, American Express, Discover dan JCB. Kartu debit juga diterima untuk kartu Visa, MasterCard atau JCB. Kartu lain tidak diterima. Beberapa bank mungkin mensyaratkan penahanan satu dollar oleh pelaku verifikasi sebelum kartu dapat dikonfirmasi.

(b) Gmail

Gmail adalah layanan email yang disediakan oleh Google LLC. Gmail API dapat digunakan untuk mengakses email Gmail.

- **Resource**

Gmail API menyediakan beberapa jenis *resource* :

- Message
Message merepresentasikan pesan dalam email. Message hanya bisa dibuat atau dihapus. Tidak ada properti dari message yang bisa diubah selain label yang diberikan ke message.
- Label
Label berfungsi sebagai sarana utama untuk mengelompokkan dan mengatur message dan thread. Label mempunyai hubungan banyak ke banyak dengan message. Artinya, satu message dapat memiliki beberapa label dan satu label dapat diberikan ke beberapa message.
Label ada dua jenis : label sistem dan label pengguna. Contoh label sistem adalah label INBOX, TRASH, dan SPAM. Label sistem dibuat secara internal dan tidak dapat dibuat, dihapus, dan dimodifikasi. Namun, beberapa label sistem dapat diberikan ke message atau dilepaskan dari message. Label pengguna dapat ditambah, dihapus, dan dimodifikasi oleh pengguna atau aplikasi.
- Draft
Draft merepresentasikan message yang belum dikirim. Message tidak bisa dimodifikasi setelah dibuat, tapi message yang terdapat di dalam draf dapat dimodifikasi. Mengirimkan draft secara otomatis akan menghapus draft tersebut dan membuatnya menjadi message dengan label sistem SENT.
- History
History adalah riwayat modifikasi message yang diurutkan secara kronologis. History hanya menyimpan perubahan dalam jangka waktu 30 hari.
- Thread
Thread adalah kumpulan message yang merepresentasikan percakapan. Thread dapat memiliki label. Thread tidak dapat dibuat, tapi dapat dihapus. Message dapat dimasukkan ke Thread.
- Setting
Setting mengontrol perilaku fitur pada Gmail kepada User. Setting tersedia untuk akses POP dan IMAP, forward email, filter, vacation auto-response, send-as aliases, signatures, dan delegates.

- **Scope**

Gmail API menggunakan OAuth 2.0 untuk menangani autentikasi dan authorization. *Developer* harus menyebutkan scope yang dipakai di aplikasi. Scope adalah string yang mengidentifikasi

resource yang ingin di akses. Scope ini digunakan bersama dengan token untuk mengamankan akses ke resource pengguna. Token tersebut memiliki masa kadaluarsa. Contoh scope :

- `https://www.googleapis.com/auth/gmail.readonly` : scope untuk membaca message dari Gmail
- `https://www.googleapis.com/auth/gmail.modify` : scope untuk mengubah label pada thread atau message
- `https://www.googleapis.com/auth/gmail.compose` : scope untuk mengirim message mewakili pengguna

• Penggunaan pada umumnya

- Mengirim message
 - i. Membuat konten email
 - ii. Membuat string yang dikodekan berdasarkan base64url dari konten
 - iii. Membuat resource message dan memasukkan string tersebut ke properti `raw`
 - iv. Memanggil `message.send` untuk mengirim message
- Mengambil email yang diterima

Mengambil email yang diterima membutuhkan ID email. Mengambil email yang diterima dapat dilakukan dengan metode `get` dari resource `User.messages`. Saat mengambil message, format dari respon dapat diatur. Format `FULL` mengembalikan seluruh informasi dari message. Format `MINIMAL` hanya mengembalikan metadata seperti label. Format `RAW` mengembalikan properti `raw` saja. Secara otomatis, format dari respon memakai format `FULL`.
- Perubahan di history

Perubahan message direpresentasikan oleh `History objects`. Properti `start_history_id` memperbolehkan *developer* mengatur dari titik mana perubahan ingin dikembalikan. Beberapa perubahan dapat mempengaruhi lebih dari satu message, sehingga history yang merepresentasikan perubahan tersebut akan berisi beberapa message.
- Manajemen Label

Label yang diberikan ke sebuah thread juga diberikan ke semua message di dalam thread. Jika sebuah label dihapus, label tersebut akan dihapus dari semua thread dan message yang memiliki label tersebut. Properti `messageListVisibility` digunakan untuk menentukan apakah message dengan label tersebut ada di message list. Properti `labelListVisibility` digunakan untuk menentukan apakah ada label tersebut di daftar label. Untuk mengubah label, gunakan `messages.modify` dan `threads.modify`.

(c) PHP IMAP

PHP IMAP adalah layanan PHP untuk mengakses email melalui protokol IMAP. Berikut adalah *function* dasar dari imap :

- `imap_8bit` : melakukan konversi 8bit string ke quoted-printable string
- `imap_alerts` : mengembalikan semua pesan peringatan dari IMAP yang telah terjadi
- `imap_base64` : decode BASE64 encoded text
- `imap_binary` : melakukan konversi 8bit string ke base64 string
- `imap_body` : membaca message body
- `imap_bodystruct` : membaca struktur dari bagian body tertentu dari message tertentu
- `imap_check` : mengecek mailbox (kotak surat)
- `imap_close` : menutup IMAP stream
- `imap_errors` : mengembalikan semua error dari IMAP yang telah terjadi

- `imap_fetch_overview` : membaca informasi pada header dari message yang diberikan
- `imap_fetchbody` : mengambil bagian tertentu dari message body
- `imap_fetchheader` : mengembalikan header dari message
- `imap_fetchmime` : mengambil MIME headers dari bagian tertentu dari message
- `imap_fetchstructure` : membaca struktur dari message tertentu
- `imap_headerinfo` : membaca header dari message
- `imap_open` : membuka IMAP stream ke mailbox
- `imap_qprint` : melakukan konversi dari quoted-printable string ke 8 bit string
- `imap_sort` : mendapatkan dan menyortir message
- `imap_utf8` : melakukan konversi dari MIME-encoded text ke UTF-8

(d) LINE

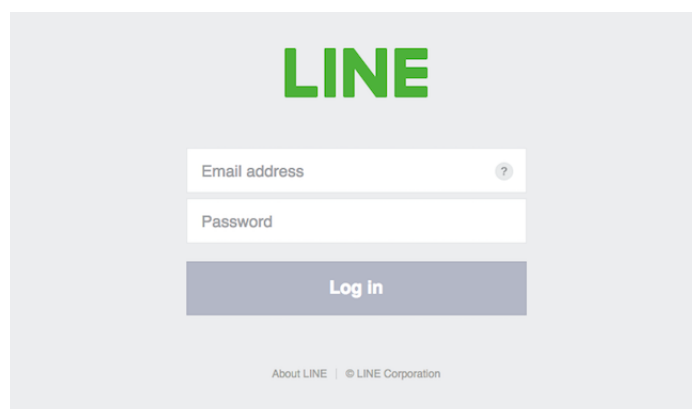
Line Menyediakan Messaging API untuk membangun messaging bot. Messaging API memungkinkan data dioper antara server dari aplikasi bot dengan LINE Platform. Ketika pengguna Line mengirimkan pesan ke bot, sebuah webhook akan terpicu dan LINE Platform akan mengirimkan permintaan ke URL webhook bot. Server akan mengirim permintaan ke LINE Platform untuk merespon pengguna. Permintaan akan dikirimkan dalam format JSON.

Untuk menggunakan Messaging API, *developer* memerlukan akun LINE@. Messaging API juga dapat digunakan menggunakan akun resmi/*official accounts*. Akun resmi mendapatkan fitur tambahan untuk pengguna enterprise.

Membuat Channel

Untuk memulai membangun bot dengan Messaging API, *developer* perlu membuat channel terlebih dahulu. Channel adalah penyambung antara LINE platform dan aplikasi yang dibuat *developer*. Berikut langkah-langkah untuk membuat channel :

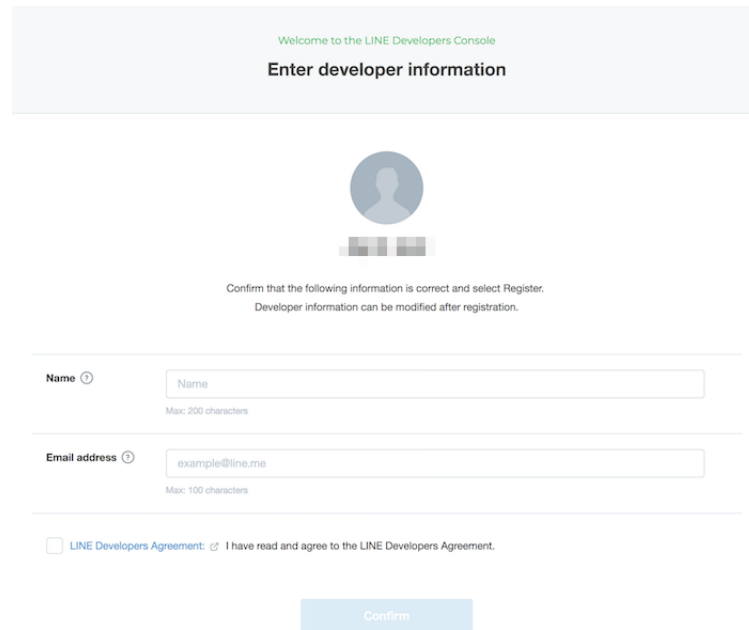
- Langkah ke-1 : Masuk ke LINE Developers console



Gambar 8: Tampilan LINE developer console saat login

Developer perlu masuk ke LINE Developers console (<https://developers.line.me/en/>) dengan alamat email dan password dari akun LINE *developer* (Gambar 8). Jika *developer* belum memiliki akun LINE, *developer* perlu mengunduh aplikasi LINE untuk mendaftar akun LINE.

- Langkah ke-2 : Mendaftar sebagai developer (*developer*)



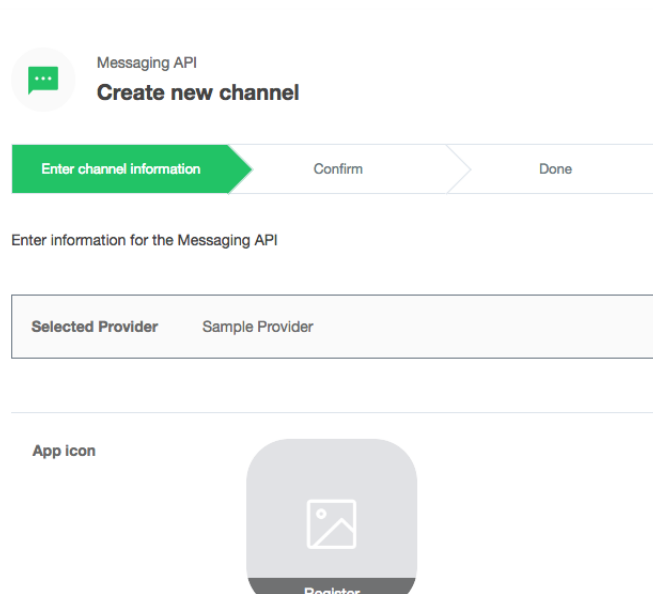
Gambar 9: Tampilan LINE developer console saat register developer

Apabila *developer* baru pertama kali masuk ke LINE Developers console, *developer* perlu membuat akun developer (Gambar 9). *Developer* hanya perlu mencantumkan nama dan alamat email untuk mendaftar.

iii. Langkah ke-3 : Membuat provider baru

Provider adalah individu atau perusahaan yang menyediakan aplikasi yang akan dibuat. *Developer* perlu mencantumkan nama provider untuk membuat provider baru. *Developer* dapat menuliskan nama *developer* sendiri atau nama perusahaan *developer*.

iv. Langkah ke-4 : Membuat channel



Gambar 10: Tampilan LINE developer console saat membuat channel

Developer perlu memasukkan informasi yang dibutuhkan untuk membuat channel :

- Ikon aplikasi

Dokumen gambar untuk ikon aplikasi harus dibawah 3MB dengan ekstensi JPEG/PNG/GIF/BMP.

- Nama aplikasi

Nama aplikasi tidak boleh lebih dari 20 karakter. Kata "LINE" tidak dapat digunakan sebagai nama aplikasi, walaupun kapitalisasinya tidak sama. Setelah dikonfirmasi, nama aplikasi tidak dapat diubah untuk tujuh hari ke depan.

- Deskripsi aplikasi

Deskripsi aplikasi tidak boleh lebih dari 500 karakter.

- Plan

Terdapat dua pilihan, Developer Trial dan Free. Plan Developer Trial memungkinkan *developer* untuk membuat bot yang dapat mengirimkan push message dan memiliki 50 teman. Apabila *developer* memilih plan ini, maka *developer* tidak dapat melakukan upgrade atau membeli ID premium. Plan Free memungkinkan *developer* untuk membuat bot dengan jumlah teman tak terbatas, namun *developer* tidak dapat mengirimkan push message. *Developer* dapat melakukan upgrade kapan saja dengan plan ini.

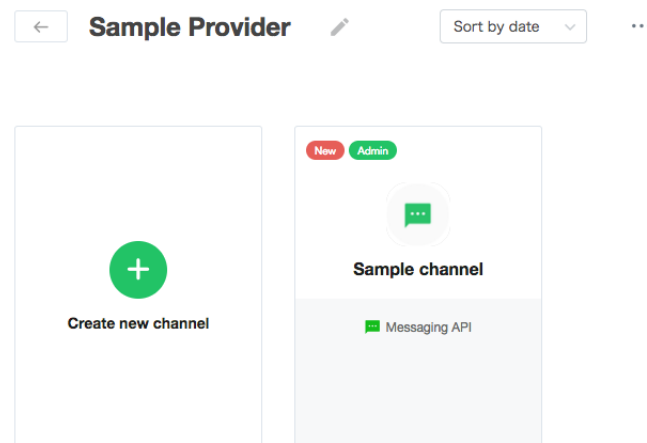
- Kategori dan Subkategori

Developer dapat memilih kategori dan subkategori yang cocok dengan aplikasi yang sedang dikembangkan.

- Alamat email

Alamat email yang dicantumkan adalah alamat email yang akan menerima notifikasi dan pengumuman penting dari LINE. Maksimal karakter pada alamat email adalah 100 karakter.

v. Konfirmasi



Gambar 11: Tampilan LINE developer console saat konfirmasi pembuatan channel

Konfirmasi channel yang baru saja dibuat.

Membuat bot

Setelah membangun channel, *developer* perlu menyiapkan server untuk menjadi host dari bot. *Developer* dapat menggunakan layanan cloud platform, seperti Heroku. Setelah itu, *developer* dapat mulai mengatur bot pada console.

Aplikasi bot membutuhkan channel access token untuk membuat API call dan webhook URL untuk menerima webhook payload dari LINE Platform. Channel access token adalah long-lived token (token yang tidak memiliki kadaluarsa) yang harus diatur di dalam authorization header ketika membuat API call. *Developer* dapat menerbitkan lagi channel access token kapanpun

melalui console. Untuk menerbitkan channel access token, klik Issue pada "Channel settings" di halaman console. Sedangkan webhook URL adalah titik akhir dari server aplikasi bot dimana webhook payload dikirimkan.

Untuk mengatur webhook URL, *developer* dapat memasukkannya ke halaman Channel settings pada console. Webhooks harus diaktifkan terlebih dahulu dengan menekan tombol enable webhooks. Untuk memeriksa apakah webhook URL dapat menerima event webhook, tekan tombol Verify dan pastikan hasilnya "Success". Webhook URL harus menggunakan HTTPS dan memiliki sertifikat SSL yang diterbitkan oleh certificate authority (CA) yang terotorisasi.

Setelah token dan webhook URL berhasil diset, tambahkan bot sebagai teman melalui akun LINE. *Developer* dapat melakukannya dengan scan kode QR pada Channel Settings.

Menkonfigurasi Keamanan

Developer dapat mengkonfigurasi keamanan tapi tidak wajib dilakukan. Untuk meningkatkan keamanan, *developer* dapat mengatur server yang dapat memanggil API pada LINE Platform pada Security settings. *Developer* dapat mendaftarkan alamat IP secara individual atau jika *developer* memiliki server yang banyak *developer* dapat menggunakan notasi classless inter-domain routing (CIDR) untuk mendaftarkan alamat jaringan.

Alur kerja Messaging API

Ketika user berinteraksi dengan bot seperti mengirimkan pesan atau menambah bot sebagai teman, LINE Platform mengirimkan HTTP POST request yang berisi webhook event object ke bot server yang disebutkan di kolom "Webhook URL" pada console. Request header berisi signature.

Untuk mengecek apakah server dapat menerima webhook event, blok bot pada LINE dan cek server logs untuk menkonfirmasi bahwa server dapat menerima unfollow event dari LINE Platform.

Untuk memastikan request yang dikirim berasal dari LINE Platform, bot server harus memvalidasi X-Line-Signature pada request header. Caranya dengan : 1. Menggunakan channel secret sebagai secret key, generate Base64-encoded digest dari request body menggunakan algoritma HMAC-SHA256 2. Menkonfirmasi signature X-Line-Signature dalam request header cocok dengan digest.

Webhook Event Object

i. Khusus untuk one-on-one chat

- Message Event
Menunjukkan bahwa ada user yang mengirim pesan. Event ini dapat dibalas.
- Follow Event
Menunjukkan bahwa akun bot ditambahkan sebagai teman (atau unblocked). Event ini dapat dibalas.
- Unfollow Event
Menunjukkan bahwa akun bot diblok
- Postback event
Menunjukkan user melakukan aksi postback. Event ini dapat dibalas.
- Beacon event
Menunjukkan bahwa user telah masuk atau keluar dari jangkauan LINE Beacon. Event ini dapat dibalas.
- Account link event
Menunjukkan bahwa user telah melink akun LINE dengan akun layanan *developer*.

ii. Group chats

- Message event
Menunjukkan bahwa ada user yang mengirim pesan. Event ini dapat dibalas.

- Join event
Menunjukkan bot telah bergabung ke sebuah group chat
- Leave event
Menunjukkan bot telah keluar dari sebuah group chat
- Postback event
Menunjukkan user melakukan aksi postback. Event ini dapat dibalas.

Operasi pada bot

Developer dapat melakukan operasi berikut lewat bot :

i. Mengirim reply message

Reply message adalah pesan yang dikirim sebagai respons dari user-generated event. User-generated event adalah event yang muncul karena user berinteraksi dengan bot, misalnya mengirim pesan. *Developer* hanya dapat membalas webhook events yang memiliki reply token. Untuk membalas pesan, kirim HTTP POST request ke `/bot/message/reply`. Sertakan channel access token di dalam authorization header dan reply token di request body. *Developer* dapat mengirimkan sampai 5 message object per request.

ii. Mengirim push message

Untuk mengirim push message, *developer* harus memerhatikan plan yang dipakai. Apabila *developer* memakai plan Free maka *developer* tidak dapat melakukan operasi ini. Push message adalah pesan yang dapat bot kirimkan ke user kapanpun. Push message tidak membutuhkan reply token seperti saat mengirim reply message. Ketika mengirim push message, sebutkan user ID di dalam property to. ID penerima dapat ditemukan dari webhook event object. Apabila penerima hanya satu, kirimkan request ke `/bot/message/push`. Sedangkan apabila penerima ada beberapa, kirimkan ke `/bot/message/multicast`. *Developer* dapat mengirimkan sampai 5 message object per request.

iii. Mendapatkan konten yang dikirim oleh user

Untuk mengambil gambar, video, atau audio yang dikirim user, kirimkan HTTP GET request ke `/bot/message/messageId/content`. Konten yang dikirim oleh user otomatis dihapus dalam jangka waktu tertentu.

iv. Mendapatkan informasi user profile

Untuk mendapatkan informasi user profile dari user yang menambahkan bot atau mengirim pesan ke bot, kirimkan HTTP GET request ke `/bot/profile/userId`. Request ini akan mengembalikan display name, user ID, profile image URL, dan status message (jika tersedia) dari user.

LINE@ Manager

LINE@ Manager adalah alat untuk mengatur akun LINE@ (LINE bot). *Developer* dapat meningkatkan user experience dengan mengatur halaman akun, membuat Timeline post, dan menggunakan fitur lain yang disediakan LINE@ Manager. Berikut adalah hal-hal yang bisa dilakukan :

i. Mengubah tampilan halaman akun

Developer dapat mengubah gambar cover, logo, tombol, dan informasi yang disediakan

ii. Mengatur greeting message

Jika *developer* mengaktifkan Greeting message pada Channel settings, maka *developer* dapat mengatur greeting message yang akan dikirim ke user saat pertama kali menambahkan bot sebagai teman. *Developer* dapat melakukannya juga dengan program melalui follow webhook event.

iii. Mengatur auto reply message

Jika *developer* mengaktifkan "Auto reply message" pada Channel settings, maka *developer* dapat mengatur pesan balasan otomatis setiap user mengirimkan pesan ke bot.

2. Memodifikasi BlueTape sehingga dapat berjalan di Heroku menggunakan PostgreSQL

Status : Ada sejak rencana kerja skripsi.

Hasil : BlueTape sudah dapat berjalan di Heroku. Situs BlueTape untuk skripsi ini diberi nama shadowtape dan dapat diakses melalui situs <https://shadowtape.herokuapp.com>. Basis data BlueTape versi skripsi ini telah dikonversi dari berbasis MySQL ke PostgreSQL. Proses modifikasi BlueTape dilakukan dengan :

- Menambah dokumen Procfile yang isinya : "web:vendor/bin/heroku-php-apache2 www/" pada direktori root
- Menambahkan 1 baris kode berikut pada dokumen config.php pada direktori www/application/config :

```
$ config['sess_save_path'] = sys_get_temp_dir();
```

Kode tersebut untuk mengatasi error yang timbul karena session tidak berhasil menulis ke *filesystem*.

- Mengganti fungsi replace pada dokumen migration dengan insert dan update. Contoh :

```
$this->db->replace('Bluetape_Userinfo', array(
    'email' => $email,
    'name' => $name,
    'lastUpdate' => strftime('%Y-%m-%d %H:%M:%S')
));
```

(line 77-81 Auth_Model.php) menjadi :

```
$PKconfirmation = $this->db->get_where('Bluetape_Userinfo', array(
    'email' => $email
));
if($PKconfirmation == null ){
    $this->db->insert('Bluetape_Userinfo', array(
        'email' => $email,
        'name' => $name,
        'lastUpdate' => strftime('%Y-%m-%d %H:%M:%S')
    ));
}
else{
    $this->db->where('email', $email);
    $this->db->update('Bluetape_Userinfo', array(
        'name' => $name,
        'lastUpdate' => strftime('%Y-%m-%d %H:%M:%S')
    ));
}
```

- Mengubah tipe data datetime menjadi timestamp
- Membungkus nama tabel atau kolom yang aturan penamaannya menggunakan *camel case* dengan tanda kutip dua

3. Memodifikasi BlueTape sehingga dapat menangkap email yang masuk ke email khusus

Status : Ada sejak rencana kerja skripsi.

Hasil : BlueTape dapat menangkap email yang masuk ke email khusus. Namun, masih terdapat error apabila email yang teridentifikasi sebagai email pengumuman memiliki lampiran. Modifikasi yang dilakukan untuk BlueTape adalah membuat email baru khusus untuk menangkap pengumuman, menambah model, view, dan controller dari modul Pengumuman, menambahkan dokumen migration untuk membuat tabel Pengumuman, dan menambah Cron. Cara kerja penangkap email khusus adalah sebagai berikut :

- Email masuk ke email khusus untuk menangkap pengumuman, yaitu `shadowbluetape@gmail.com`
- Menjalankan secara manual Cron dengan mengunjungi : `http://shadowtape.herokuapp.com/Cron/daily`
- Cron akan mengaktifkan pengecekan email baru yang belum terbaca
- Apabila ada email baru, maka akan dilakukan pengecekan status pengirim sah atau tidak
- Apabila sah, informasi email akan dimasukkan ke tabel **Pengumuman**
- Email yang baru masuk ke tabel dapat langsung dilihat melalui menu Pengumuman di BlueTape

4. Memodifikasi BlueTape sehingga dapat melakukan push notification ke akun LINE@

Status : Ada sejak rencana kerja skripsi.

Hasil : BlueTape belum bisa melakukan push notification ke akun LINE@.

5. Melakukan pengujian

Status : Ada sejak rencana kerja skripsi.

Hasil : Belum dilakukan. Langkah ini akan dilakukan setelah fitur selesai dibuat.

6. Menulis dokumen skripsi

Status : Ada sejak rencana kerja skripsi.

Hasil : Bab 1 telah selesai ditulis. Bab 1 berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika penulisan. Bab 2 telah selesai ditulis. Bab 2 berisi dasar teori tentang BlueTape, Heroku, Gmail, PHP IMAP, dan LINE. Bab 3 telah ditulis sebagian. Bagian yang telah ditulis di bab 3 adalah analisis BlueTape, dan analisis Heroku.

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Skripsi 1 ini adalah sebagai berikut:

1. Melakukan studi literatur tentang Heroku, Gmail, PHP IMAP, dan LINE
2. Memodifikasi BlueTape sehingga dapat menangkap email yang masuk ke email khusus
3. Menulis bab1, bab2, dan sebagian bab3 pada dokumen skripsi

7 Kendala yang Dihadapi

Kendala - kendala yang dihadapi selama mengerjakan skripsi :

- Pembagian waktu untuk magang di perpustakaan, kerja di DNArtworks, kerja praktek dengan Catholicer, mengerjakan tugas-tugas dari kuliah biasa, dan mengerjakan skripsi.
- Kendala pada laptop yang tidak dapat menanggung beban berat, sehingga harus pintar-pintar mengelola apa yang harus dibuka dan sabar apabila respon lama.
- Selama mengerjakan skripsi, daya tahan tubuh menurun sehingga mudah terserang penyakit.

Bandung, 15/11/2018

Ellena Angelica

Menyetujui,

Nama: Pascal Alfadian Nugroho
Pembimbing Tunggal