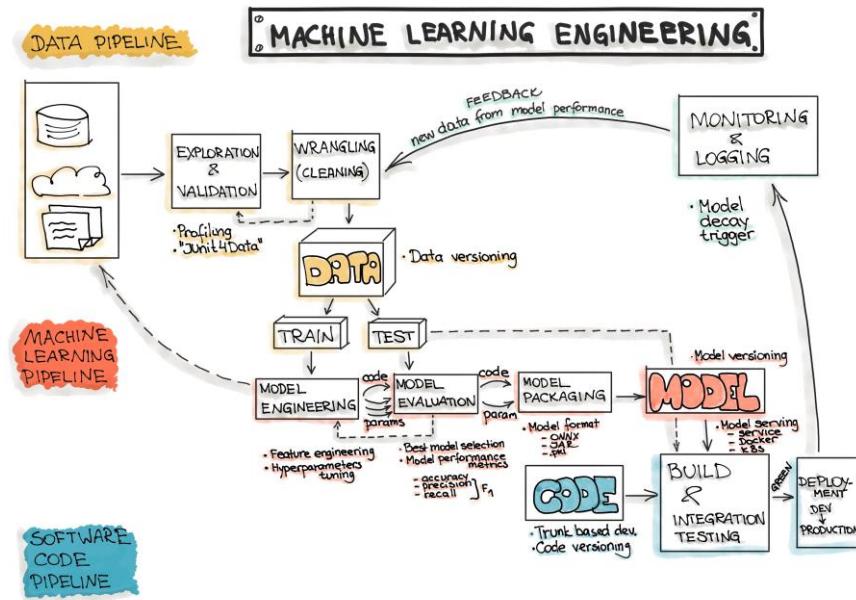


**MELAKUKAN  
DEPLOYMENT  
MODEL  
MACHINE LEARNING**

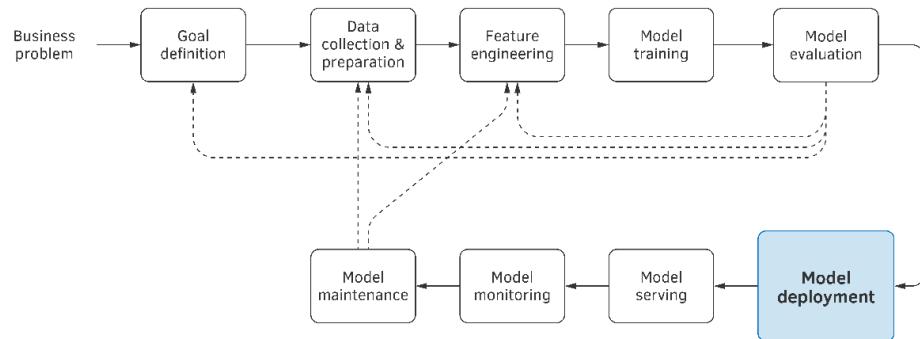
## A. Konsep dasar Deployment Model

*Deployment model* adalah suatu proses untuk membuat model (model AI) tersebut tersedia pada lingkungan produksi, dimana model tersebut dapat memberikan prediksi ke sistem perangkat lunak yang lain. Deployment model merupakan tahapan terakhir pada *machine learning lifecycle* dan merupakan tahapan yang paling menantang. Hal ini dikarenakan model dari hasil pengujian di lab akan diimplementasikan ke aplikasi real dengan data real yang ada di lapangan beserta implementasi di dalam infrastrukturnya.

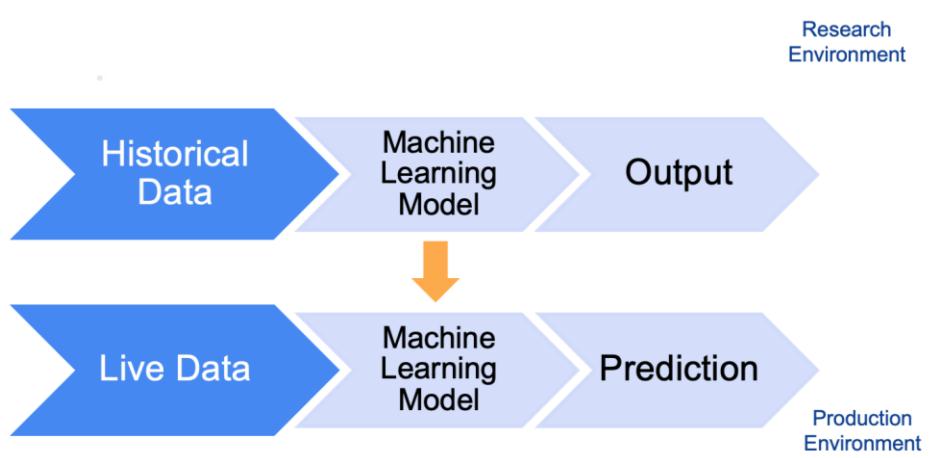


Gambar 1. Ilustrasi machine learning lifecycle

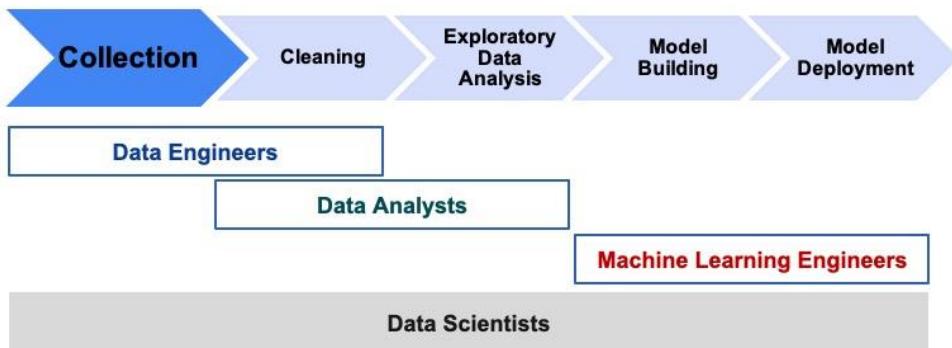
Deployment model sangat penting dikarenakan, untuk menggunakan model ini, penerapan secara efektif model tersebut ke dalam lingkungan produksi sangat diperlukan. Hal ini ditujukan agar model tersebut dapat memberikan prediksi ke sistem perangkat lunak yang lain. Selain itu deployment model diperlukan untuk mengekstraksi prediksi dengan handal dan membagikannya dalam sistem yang lain sangat penting dilakukan untuk memaksimalkan nilai model machine learning yang telah dibuat.



Gambar 2. Ilustrasi siklus projek machine learning



Gambar 3. Ilustrasi model machine learning yang dihasilkan dari research environment



Gambar 4. Siklus pada Data Scientists

Dalam implementasinya, *deployment model* memiliki beberapa tantangan. Adapun tantangan pada implementasi *deployment model* dapat dikategorikan menjadi dua bagian utama, yaitu tantangan pada perangkat lunak biasa dan tantangan spesifik pada *machine learning*. Tantangan pada perangkat lunak biasa, antara lain: *reliability*, *reusability*, *maintainability*, dan *flexibility*. Sementara itu, tantangan spesifik pada *machine learning* adalah *reproducibility*. Reproducibility

Lebih lanjut, dalam implementasinya di lapangan terdapat juga beberapa hal yang perlu diperhatikan saat mengimplementasikan *deployment model*, antara lain:

1. Koordinasi yang baik antara Data Scientists, IT Teams, Software Developers, dan profesional bisnis. Dalam melakukan koordinasi, perlu dipastikan beberapa hal seperti:
  - a. Memastikan bahwa model dapat bekerja secara *reliable*.
  - b. Memastikan bahwa model dapat memberikan keluaran yang sesuai.
2. Potensi perbedaan antara bahasa pemrograman yang digunakan dalam mengembangkan model dengan bahasa yang digunakan pada sistem produksi. Hal ini menjadi sangat penting untuk diperhatikan karena apabila terdapat ketidakcocokan maka perlu dilakukan coding ulang yang berakibat akan memperpanjang durasi pengerjaan proyek dan mengurangi produktifitas.

## B. Deployment Model Berbasis Pola

Dalam *deployment model*, suatu model dapat dideploy berdasarkan beberapa pola, yaitu secara statistic (*static deployment*), secara dinamis pada perangkat pengguna (*Dynamic deployment on user's devices*), dan secara dinamis pada server (*Dynamic deployment on a server*)

### 1. Static Deployment

*Static Deployment* mirip dengan *deployment* perangkat lunak seperti biasa. File binary yang dapat diinstall dari seluruh perangkat lunak. Model dikemas sebagai resources yang tersedia saat runtime seperti .dll files (windows), \*.so (linux), Java dan .Net.

Keuntungan dari static deployment sendiri, antara lain:

- Eksekusi lebih cepat karena perangkat lunak memiliki akses langsung ke model.
- Efisien waktu dan privacy. Hal ini dikarenakan data pengguna tidak harus diupload di server.
- Model dapat dipanggil saat pengguna *offline*.
- Pengoperasian model menjadi tanggung jawab pengguna, bukan vendor perangkat lunak.

### 2. Dynamic Deployment

#### • *Dynamic Deployment on user's devices*

*Dynamic deployment* pada perangkat user mirip dengan *static deployment*, perbedaannya adalah model bukan merupakan bagian dari *binary code* aplikasi. Pembaruan model dapat dilakukan tanpa harus memperbarui seluruh aplikasi yang berjalan pada perangkat pengguna. *Dynamic deployment*

pada perangkat user dapat dilakukan dengan *deploying model parameters*, *deploying a serialized object*, dan *deploying ke web browser*

**a) Deploying model parameters**

Pada skenario ini, file model hanya berisi parameter-parameter yang telah dipelajari. Perangkat pengguna juga telah menginstall *runtime environment* untuk model tersebut. Selain itu, juga dapat menggunakan beberapa versi ringan dari *machine learning package* sehingga dapat berjalan pada *mobile devices*, seperti:

- TensorFlow,
- Apple's Core ML
- Scikit-learn, Keras, XGBoost



Gambar 5. Machine learning package yang biasa digunakan pada mobile devices

**b) Deploying of serialized object**

Pada skenario ini, file model adalah object serial yang akan di-deserialisasi oleh aplikasi. Keuntungannya adalah tidak memerlukan *runtime environment* pada perangkat pengguna. Namun juga memiliki kelemahan, seperti update system akan cukup berat dan hal ini merupakan masalah jika perangkat lunak kita memiliki jutaan pelanggan.

**c) Deploying to web browser**

Pada skenario ini, akses ke browser banyak digunakan oleh aplikasi desktop maupun seluler. *Deploying to web browser* berarti model dapat dilatih dan berjalan pada browser. Contoh: framework TensorFlow.js. Skenario lain adalah model TensorFlow ditraining

menggunakan python kemudian dideploy dan dijalankan di browser dengan runtime environment JavaScript. Dalam hal ini, GPU (Graphic Processing Unit) dapat digunakan juga oleh TensorFlow.js

Kelebihan dari *dynamic deployment on user devices*, antara lain:

- Proses model lebih cepat karena komputasi sebagian di perangkat pengguna.
- Jika diimplementasikan ke browser, pengorganisasian infrastruktur hanya perlu menyajikan halaman web yang menyertakan parameter model.
- Model sangat mudah tersedia untuk analisis pihak ketiga.

Sedangkan, kekurangan *dynamic deployment on user devices*:

- Biaya bandwidth meningkat (jika berbasis browser)
- Updating model (jika serial)
- Performansi model susah dimonitor

- **Dynamic Deployment on a server**

Dynamic deployment pada server merupakan salah satu solusi untuk mengatasi kekurangan pada dynamic deployment pada perangkat user. Model ditempatkan pada server, tersedia untuk antarmuka REST API atau gRPC Google. Dynamic deployment pada server dapat dilakukan dengan Deployment pada virtual machine, Deployment dalam container, Serverless deployment, dan Model streaming

- a) **Deployment pada virtual machine**

- Pada domain web di cloud, prediksi disajikan sebagai respons HTTP.
    - Hal ini dapat diilustrasikan sebagai berikut :  
Pengguna → Layanan website (pada Virtual Machine/VM) → Koneksi ke machine learning → Mengubah output ke dalam bentuk JavaScript Object Notation (JSON) atau XML.
    - VM berjalan secara paralel untuk mengatasi beban komputasi yang tinggi.
    - Setiap instance berisi codes untuk menjalankan ekstraktor fitur dan model
    - Layanan web yang memiliki akses code tersebut
    - Dalam python, layanan web REST API biasanya diimplementasikan menggunakan Flask atau FastAPI
    - Tensorflow → TensorFlow Serving (layanan gRPC bawaan).

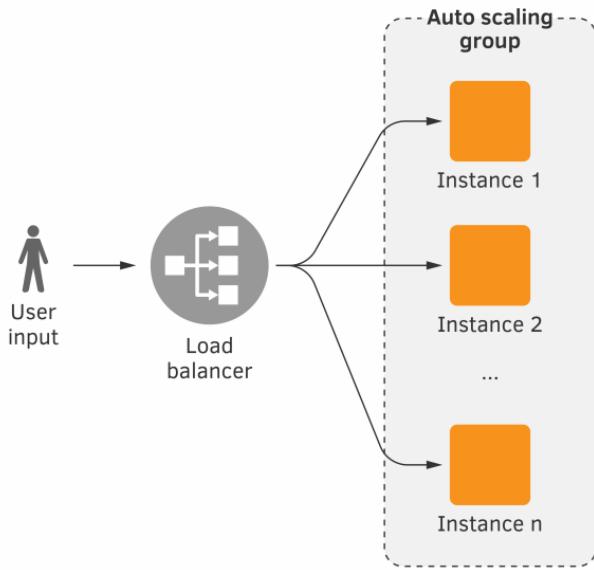
Adapun Keuntungan implementasi pada VM dantara lain:

- Arsitektur sistem perangkat lunak secara konseptual sederhana.

Sedangkan, kekurangan implementasi pada VM, antara lain:

- Pemeliharaan server (fisik maupun virtual).
  - Latensi jaringan

Biaya relative tinggi dibanding dengan menggunakan container atau serverless.



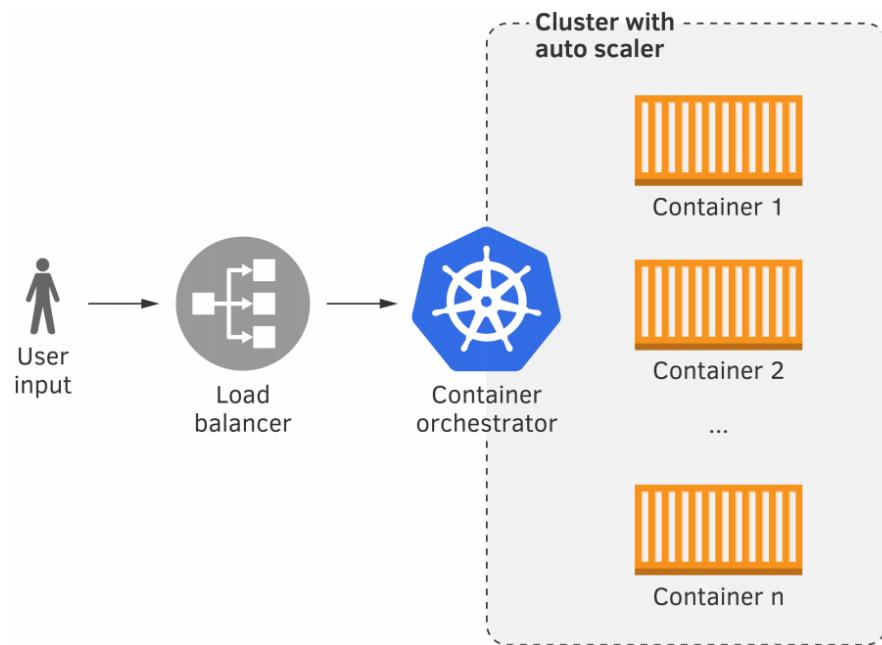
Gambar 6. Deployment model untuk web services pada virtual environment

### b) Deployment pada container

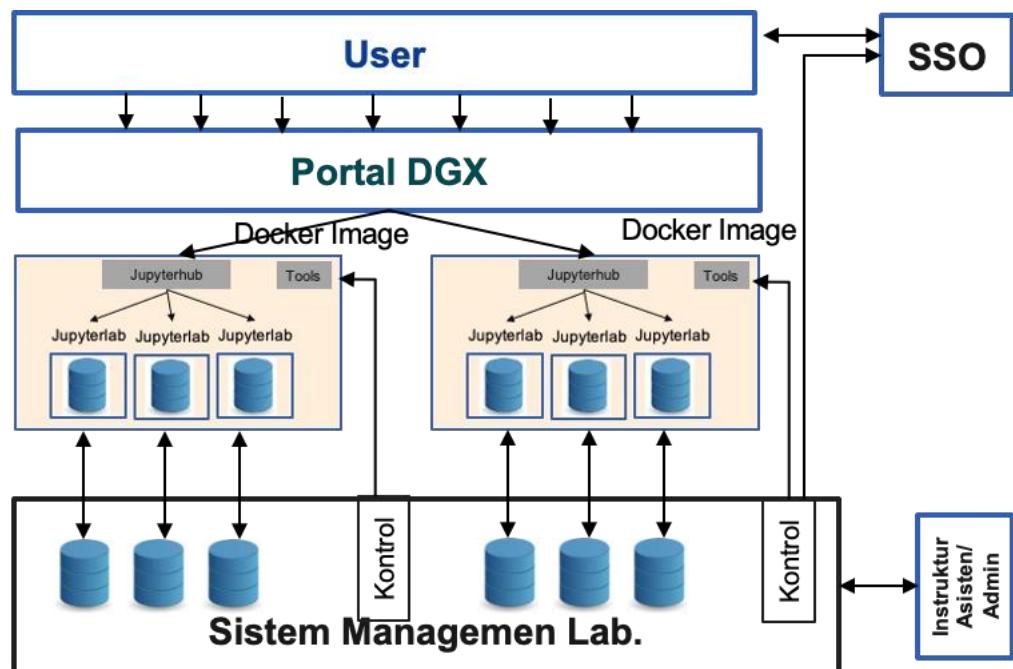
Container mirip dengan VM yang memiliki runtime yang terisolasi dengan sistem file, CPU, Memory, dan ruang prosesnya sendiri. Container merupakan suatu alternatif modern dibanding dengan deployment pada VM. Semua container berjalan pada VM atau fisik yang sama dan berbagi sistem operasi. VM Menjalankan sistem operasi sendiri.

Adapun keuntungan deployment pada container, adalah lebih hemat resources dan fleksibel dibanding menggunakan VM. Sedangkan kelemahan deployment pada container yaitu deployment pada container sering dianggap kompleks dan membutuhkan expert/tenaga ahli.

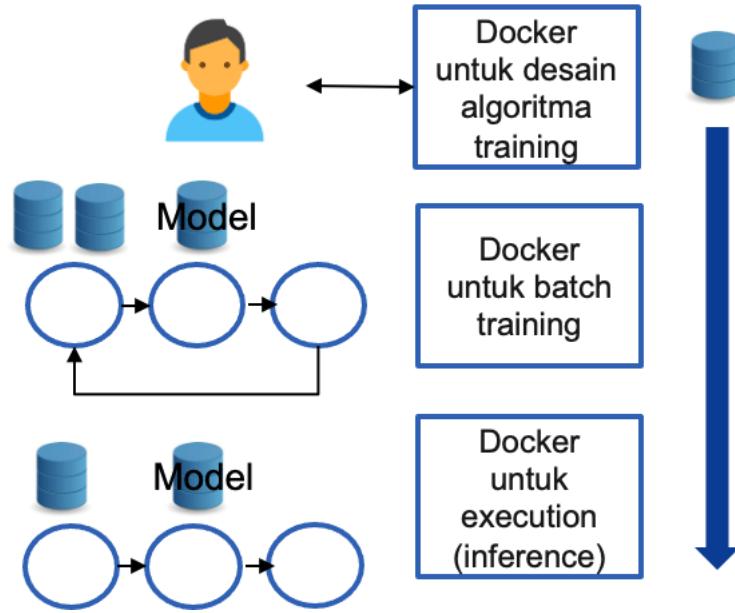
Sistem machine learning dan web services diinstall pada container (Docker Container). Container orchestrator digunakan untuk menjalankan container pada sekelompok server fisik/virtual. Contoh container orchestrator adalah Kubernetes, AWS fargate, Google Kubernetes Enginee.



Gambar 7. Deployment model untuk web services dalam container yang berjalan pada suatu cluster



Gambar 8. Contoh arsitektur penerapan Deployment Model menggunakan container



Gambar 9. Aliran dalam Deployment Model dari desain algoritma hingga model untuk inference

- Docker adalah aplikasi open source untuk menyatukan file-file yang dibutuhkan suatu perangkat lunak sehingga menjadi satu kesatuan yang lengkap dan berfungsi.
- Data pengaturan dan file pendukung disebut sebagai image. Kumpulan image digabung menjadi suatu wadah yang disebut Container.
- Fitur Docker:
  - Docker Engine, digunakan untuk membangun Docker Images dan membuat Container Docker.
  - Docker Hub, registry yang digunakan untuk berbagai macam Docker Image.
  - Docker Compose, digunakan untuk mendefinisikan aplikasi menggunakan banyak Container Docker.
  - Docker Mac, Docker Linux, Docker Windows

### c) Serverless Deployment

Serverless deployment merupakan deployment yang memanfaatkan severless computing dari cloud services providers, seperti:

- Amazon (lambda functions pada AWS),
- Google (Google cloud platform) dan
- Microsoft (functions pada Microsoft Azure).

Serverless deployment sendiri terdiri dari:

- Arsip ZIP yang berisi codes untuk menjalankan machine learning (model, feature extractor, dan scoring code).

- File arsip ZIP berisi: nama tertentu yang berisi fungsi tertentu, atau class-method dengan spesifik signature.  
File arsip ZIP diupload pada cloud platform dan dilakukan registrasi dengan nama yang unik.

- Cloud services:

- Menyediakan API untuk mensubmit masukan ke Serverless Function.
- Menangani deploying codes dan model agar memadai pada sumber daya yang dimiliki.
- Mengeksekusi codes dan mengarahkan keluaran kembali ke client.
- Limitasi pada waktu eksekusi fungsi, ukuran file ZIP, dan jumlah RAM
  - Harus menyertakan Python Library agar model dapat dieksekusi dengan benar: Numpy, SCipy, Scikit-Learn.

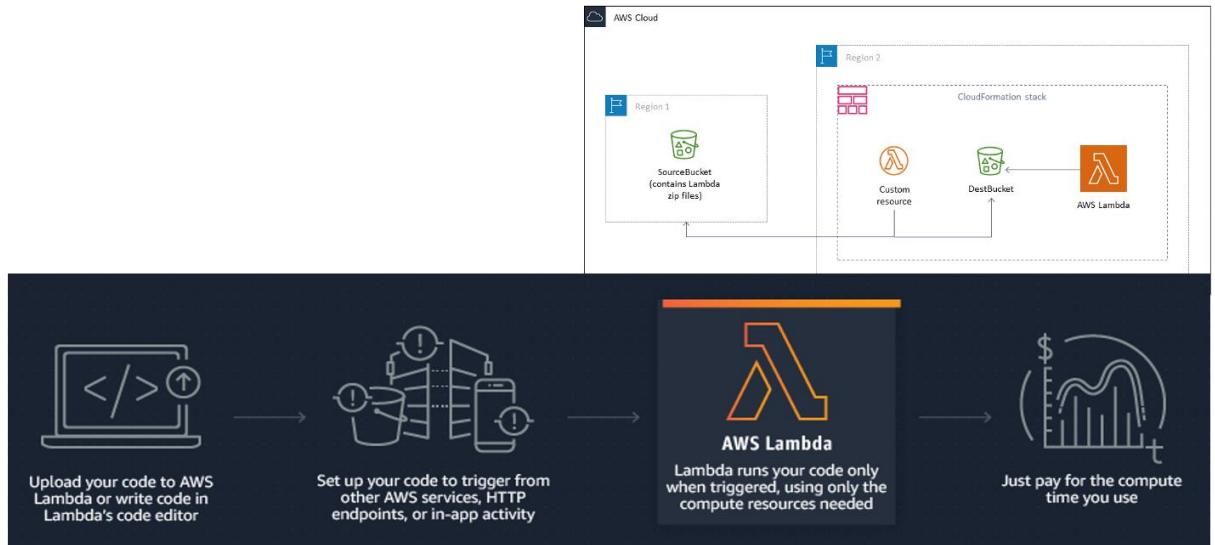
Bergantung pada platform cloud: Java, Go, PowerShell, Node.js, C#, Ruby

Adapun kelebihan dari serverless deployment, adalah:

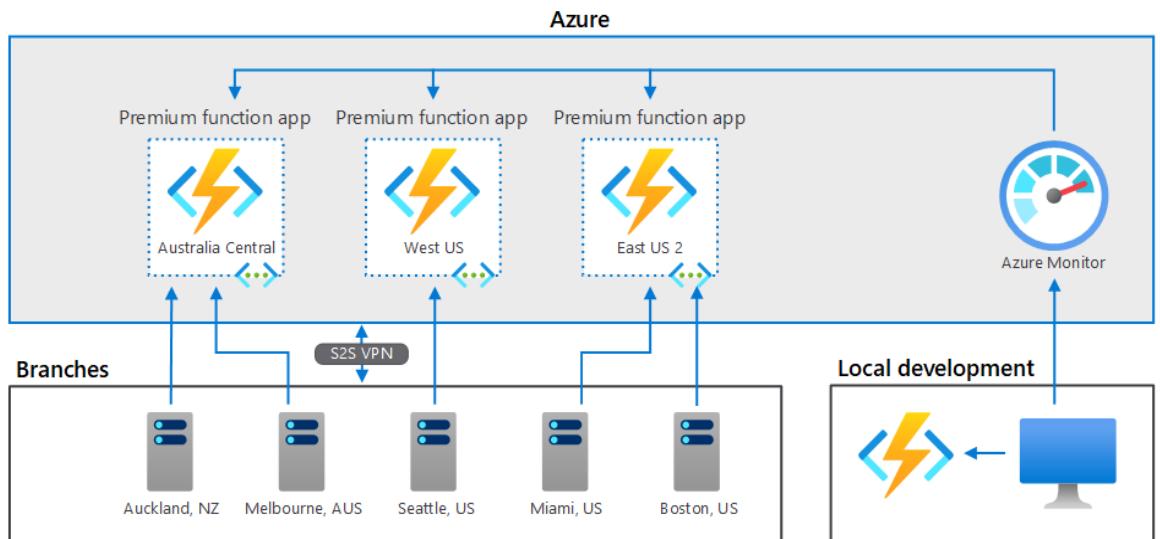
- Tidak perlu menyediakan server maupun VM.
- Tidak perlu install depedensi, maintenance atau update system.
- Bersifat scalable.
- Hemat biaya: hanya membayar waktu komputasi.
- Mendukung operasi sinkron dan asinkron
- Rollback yang mudah untuk kembali ke versi sebelumnya.

Sedangkan, kekurangan serverless deployment:

- Limitasi pada waktu eksekusi fungsi, ukuran file ZIP, dan jumlah RAM
- Tidak tersedianya akses GPU: terbatas jika ingin mendeploy Deep Model.



Gambar 10. Lambda functions pada AWS



Gambar 11. Function pada Azure (Microsoft)

Google Cloud Platform Icons

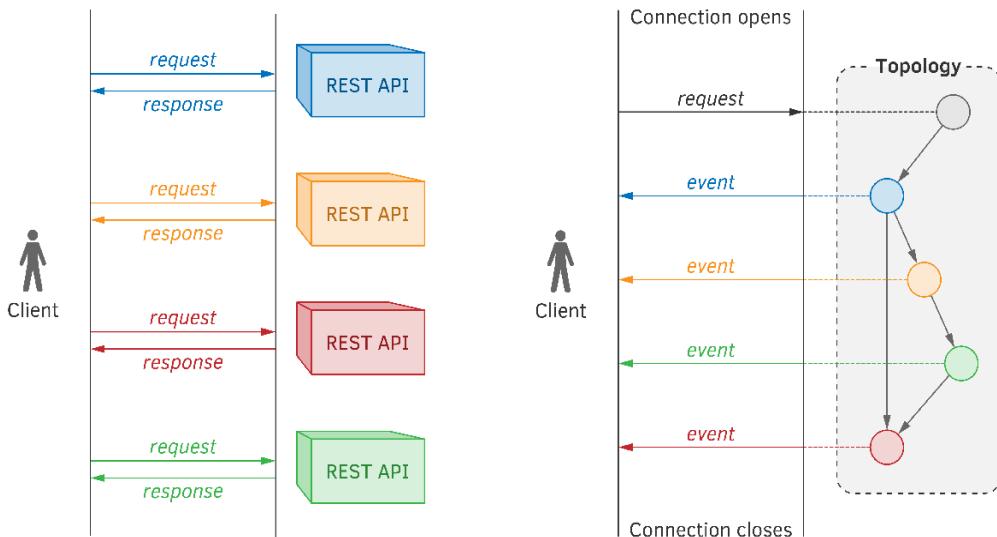


Gambar 12. Google cloud platform

#### d) Model Streaming

Model Steaming merupakan kebalikan dari REST API. Semua model dan codes didaftarkan pada mesin pemrosesan aliran (SPE).

- Apache Storm, Apache Spark, Apache Flink
- Aplikasi berbasis stream-processing library (SPL): Apache Samza, Apache Kafka Streams, Akka Streams.

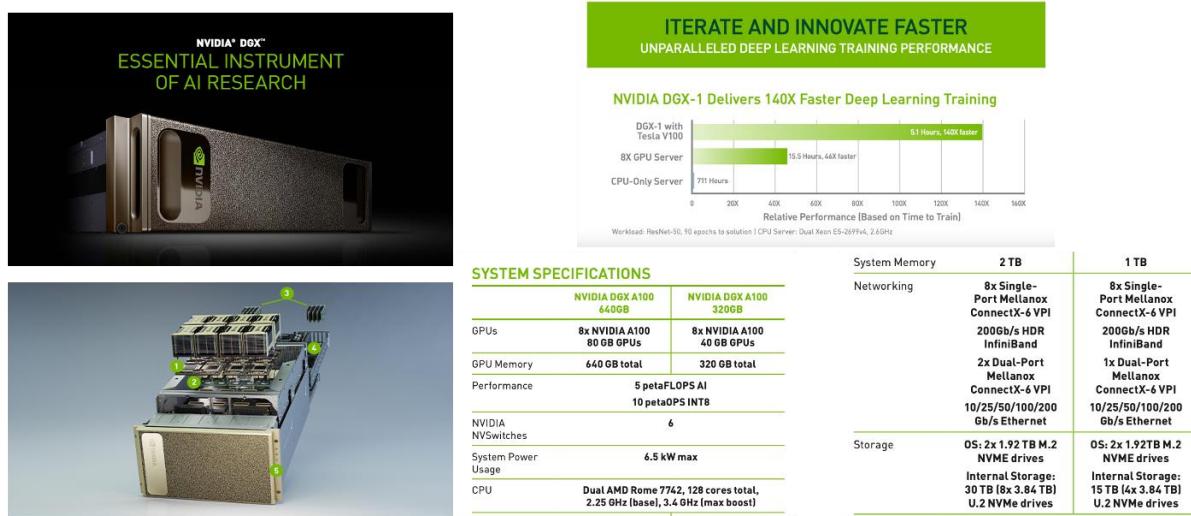


Gambar 13. Ilustrasi perbedaan antara REST API dengan Streaming

**REST API:** Memproses elemen data, client menggunakan REST API mengirimkan permintaan satu-per-satu dan menerima respon secara sinkronus

**STREAMING:** Memproses elemen data, client menggunakan STREAMING dengan membuka koneksi, mengirimkan permintaan, dan mendapatkan update event ketika hal tersebut terjadi

### C. AI Infrastructure dan platform



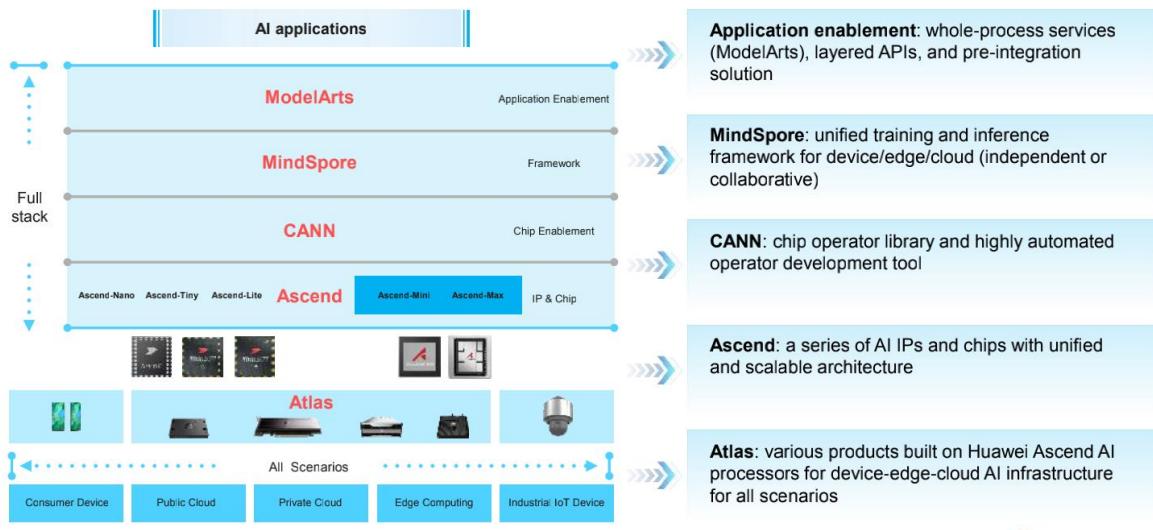
Gambar 14. Contoh peralatan yang digunakan sebagai arsitektur AI dari NVIDIA

### Atlas AI Computing Platform Portfolio

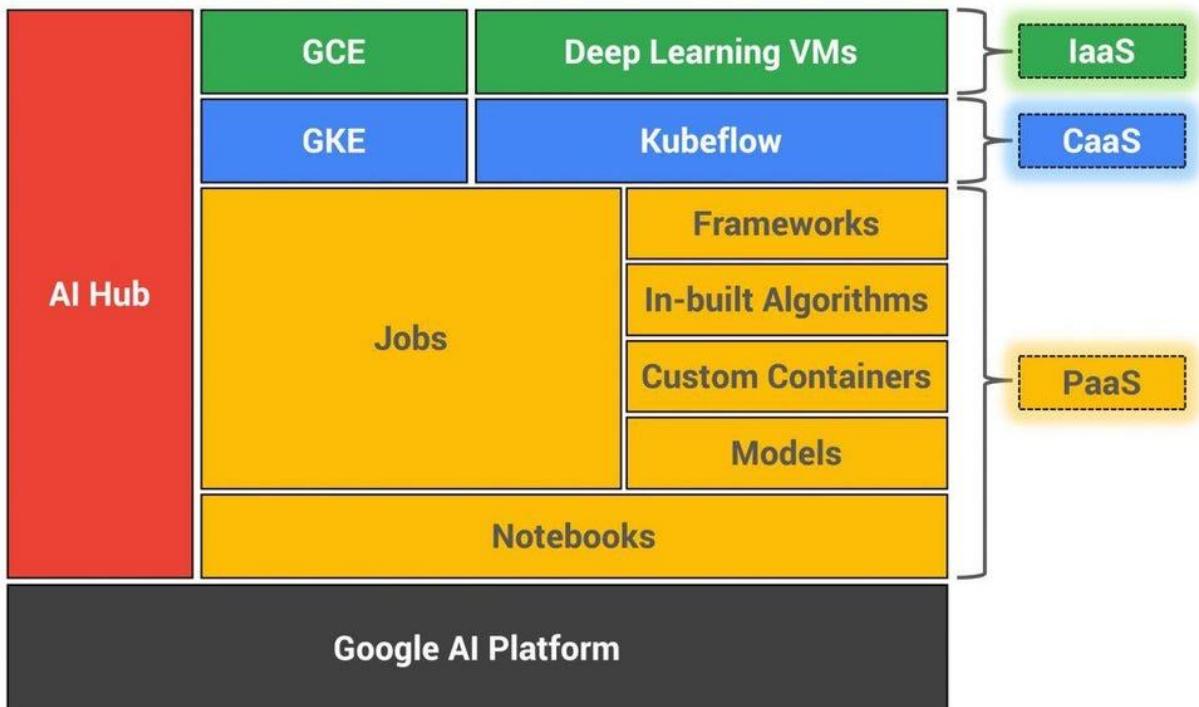


Gambar 15. Contoh peralatan yang digunakan sebagai arsitektur AI dari Huawei

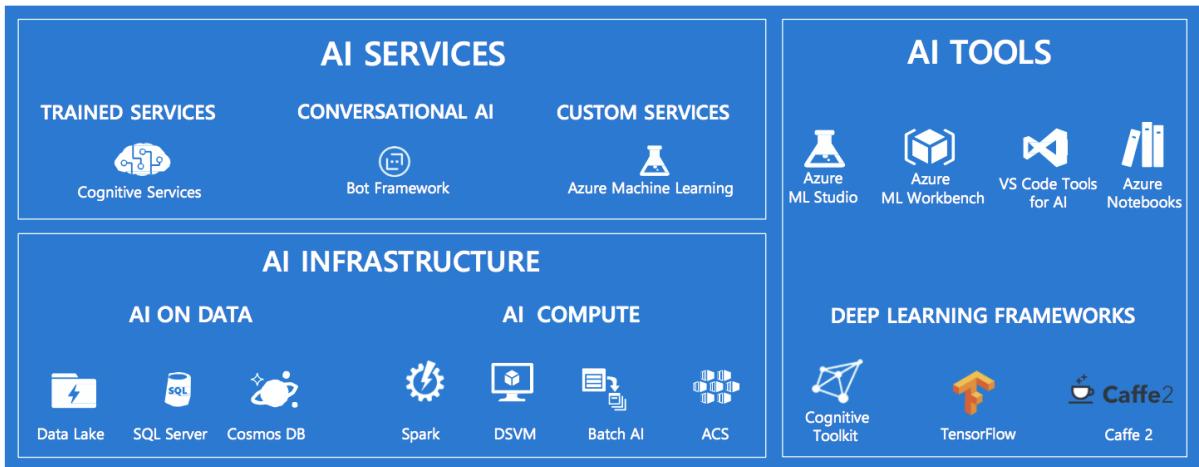
## Huawei's Full-Stack, All-Scenario AI Solution



Gambar 16. Huawes's Full-Stack dan beberapa scenario untuk solusi AI



Gambar 17. Struktur Google AI platform



Gambar 18. AI services dan tools pada Microsoft



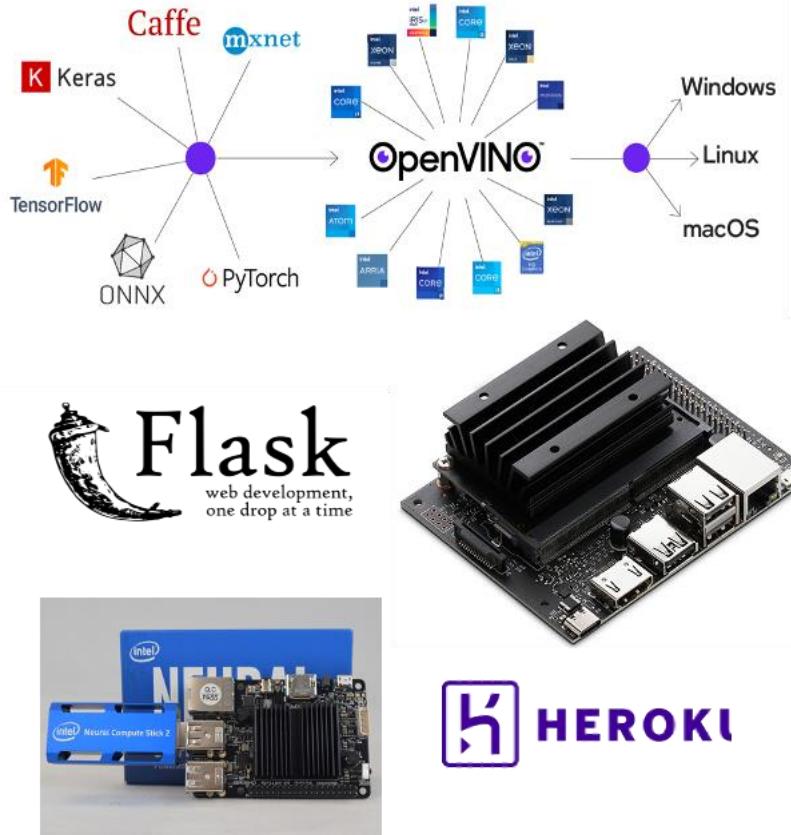
Gambar 19. Contoh aplikasi yang menerapkan Deployment Model

## D. Deployment Strategy

### • Deployment strategy

- Computational resources: melakukan prediksi (inference) lebih baik dibandingkan melakukan training.
- Inferensi pada perangkat (berbasis Apps):
  - Performansi
  - Data dengan volume yang besar → bandwidth resources
  - User tidak memiliki akses internet D
- Dapat menggunakan OPENVINO, Jetson NANO, dll (berbasis Desktop) → Robotics, Surveillance System, dll
- Dapat menggunakan microplatform FLASK dan HEROKU (berbasis Web)

- Deployment strategy secara tipikal dapat dibagi menjadi:
  - Single Deployment
  - Silent Deployment
  - Canary Deployment



Gambar 20. Beberapa contoh tools yang dapat digunakan dalam Deployment Strategy

**Single deployment** merupakan strategi yang paling sederhana. Pada Model dibuat serial menjadi file, file lama bisa direplace dengan yang baru, termasuk ekstraktor fitur jika diperlukan. Keunggulan dari strategi ini adalah sederhana, sementara itu kelemahan adalah jika ada bug, maka semua pengguna akan terpengaruh.

Berikut adalah contoh penerapan single deployment

- **Deployment di cloud environment:**
  - VM atau container baru harus disiapkan
  - VM image atau container lama direplace
  - Autoscaler memulai dengan yang baru
- **Deployment di server:**
  - Upload new model file pada server
  - Replace yang lama dengan yang baru
  - Lakukan restart web service

**Silent deployment** merupakan strategi dengan menjalankan versi model yang baru dan lama secara paralel. Keunggulan dari deployment ini adalah menyediakan waktu yang cukup untuk memastikan model baru dapat digunakan sesuai kriteria. Sementara itu kelemahannya adalah dapat menghabiskan banyak resource karena menjalankan dua model secara bersama-sama.

**Canary deployment** merupakan strategi deployment dengan cara mendorong versi dan kode model baru ke sebagian kecil pengguna, dengan tetap menjalankan versi lama untuk sebagian besar pengguna. Keunggulan dari deployment ini adalah memungkinkan untuk memvalidasi model baru dan efek prediksinya, selain itu, deployment tidak mempengaruhi banyak user jika ada bug. Kelemahan pada canary deployment adalah lebih rumit dan kompleks.

Dalam Deployment Strategy, terdapat istilah **reproducibility**, dimana ini adalah akuntabilitas yang diperlukan dalam bisnis untuk lebih memahami dan mempercayai penerapan machine learning pada kehidupan sehari-hari. Sementara itu, reproducibility pada machine learning yaitu suatu kemampuan untuk menduplikasi model secara tepat sehingga ketika model dilewati dengan data input yang sama, model yang direproduksi akan mengembalikan output yang sama. Tantangan yang ditemui pada proses Reproducibility ketika Deployment Model antara lain sebagai berikut

- Feature tidak terdapat pada Live Environment
- Perbedaan bahasa pemrograman
- Perbedaan perangkat lunak
- Kondisi real tidak cocok dengan data dan kondisi pada saat training model

Berdasarkan permasalahan tersebut, maka solusi yang dapat dilakukan pada proses Reproducibility ketika Deployment Model adalah

- Versi perangkat lunak harus sama persis dan aplikasi harus mencantumkan semua dependensi library pihak ketiga beserta versinya.
- Menggunakan container dan perangkat lunak untuk melacak spesifikasi.
- Research, develop dan deploy menggunakan bahasa yang sama (contoh: Python).
- Sebelum membangun model, harus memahami terlebih dahulu bagaimana model akan diintegrasikan dengan sistem lain.

## E. Menyimpan model

```
H=model.fit(trainX, trainY,validation_data=(testX, testY), batch_size=b, epochs=e, shuffle=True )

regressor.fit(X,y)

from keras.models import load_model
model.save('model.h5')

import pickle
```

```
pickle.dump(regressor, open('model.pkl','wb'))
```

## F. Persiapan Deployment

Berikut adalah beberapa hal yang harus dipersiapkan untuk men-deploy model *Machine Learning* yang sudah dibuat:

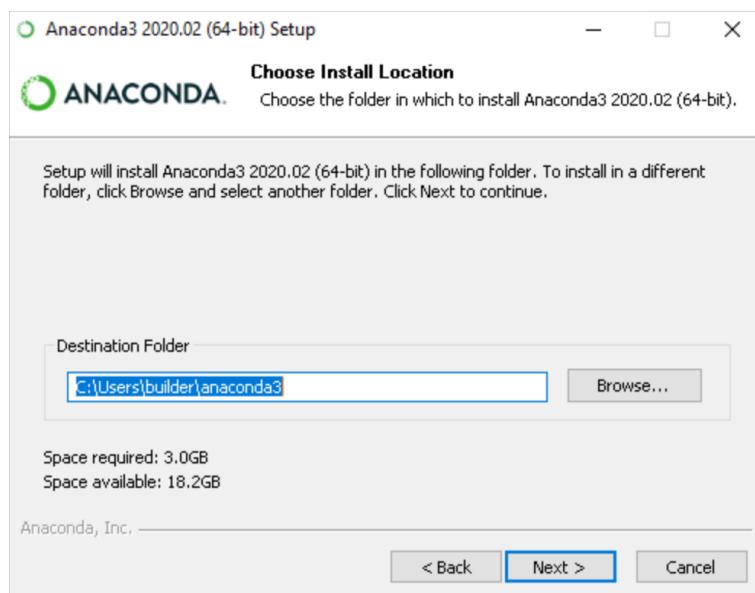
### 1. Anaconda

**Anaconda** berfungsi membuat *virtual environment* untuk aplikasi yang dibuat menggunakan *Python*. *Virtual environment* ini akan menampung semua *dependencies/library* yang dibutuhkan pada aplikasi *Python* dan model *Machine Learning* yang telah dibuat.

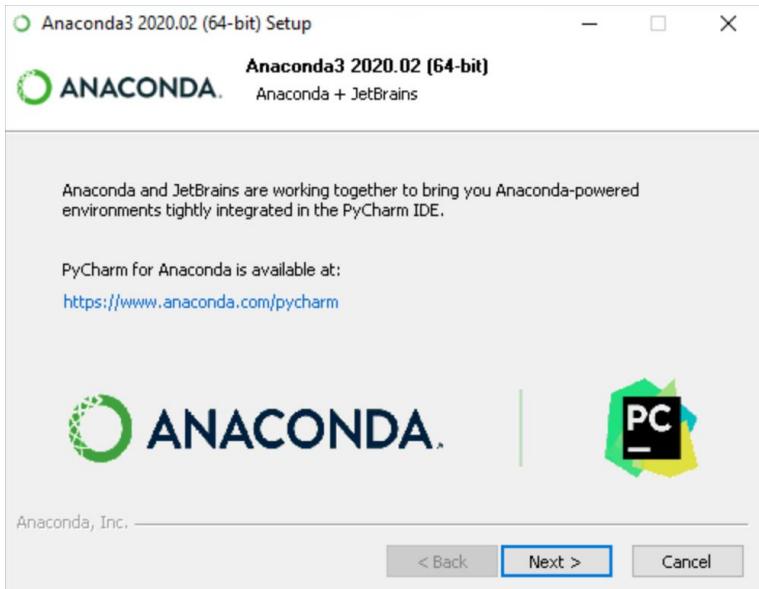


### Install Anaconda untuk Laptop dan PC (Windows)

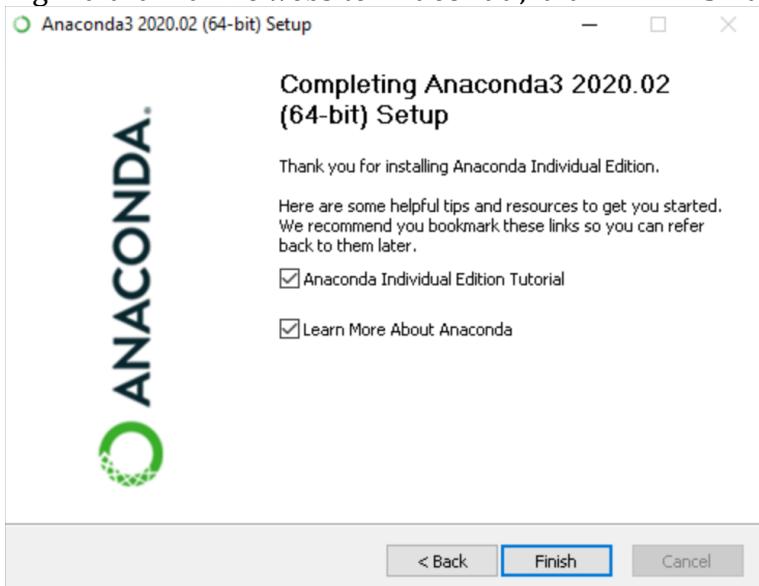
- Download installer Anaconda* [di sini](#). Sesuaikan bit dari sistem operasi perangkat Anda (32bit atau 64bit)
- Buka file *installer* yang sudah didownload, klik **Next** pada halaman awal.
- Baca licensing terms dan klik "*I Agree*".
- Pilih "*Just for me*" pada pilihan *install for*, lalu klik **Next**.
- Pilih folder tujuan *install Anaconda*, lalu klik **Next**.



- Centang "*Register Anaconda3 as my default Python 3.7*", lalu klik **Install**.
- Setelah *install* selesai, Anda akan diberikan saran untuk menginstall *PyCharm* melalui *link*, klik **Next** untuk tidak menginstall *PyCharm*.



- h. *Install* selesai, matikan centang pada 2 pilihan yang diberikan jika Anda tidak ingin diarahkan ke website **Anaconda**, lalu klik **Finish** untuk menutup *installer*.



2. *Visual Studio Code (VSCode)*

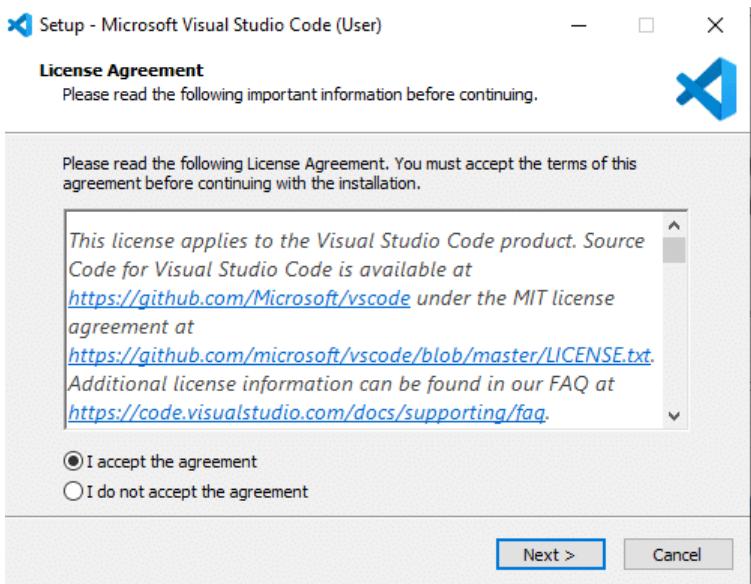
**VSCode** merupakan *text editor* yang dapat digunakan untuk beragam bahasa pemrograman, termasuk *Java*, *JavaScript*, *Go*, *Node.js*, *Python* dan *C++*. Pada modul ini, **VSCode** akan digunakan untuk membuka *source code* dari aplikasi *Python* dan model *Machine Learning* yang akan di-deploy.



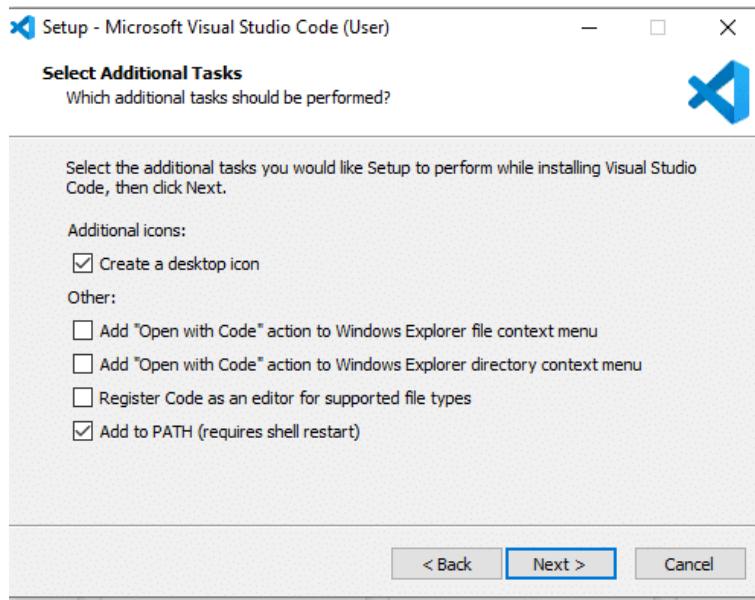
# Visual Studio Code

## **Install VSCode untuk Laptop dan PC**

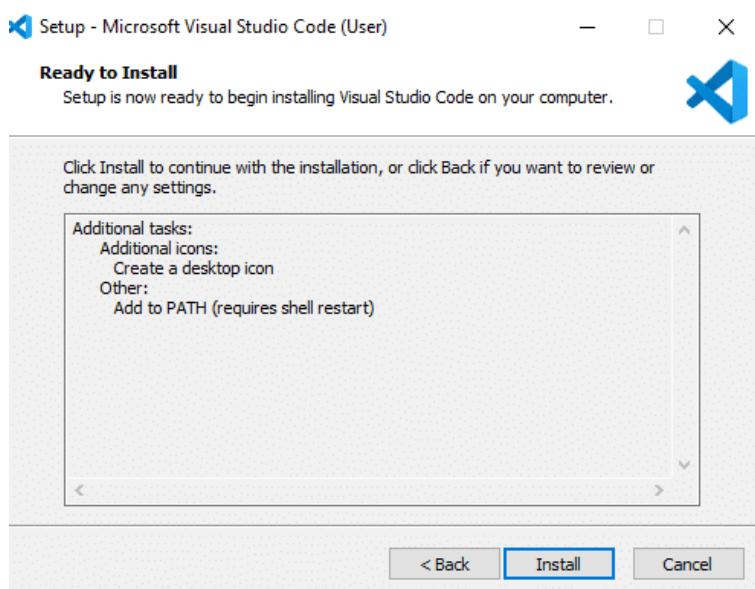
- a. Download installer Visual Studio Code [di sini](#). Sesuaikan bit dari sistem operasi perangkat Anda (32bit atau 64bit).
- b. Buka file installer yang telah selesai didownload.
- c. Baca License Agreement, pilih "I accept the agreement", lalu klik **Next**.



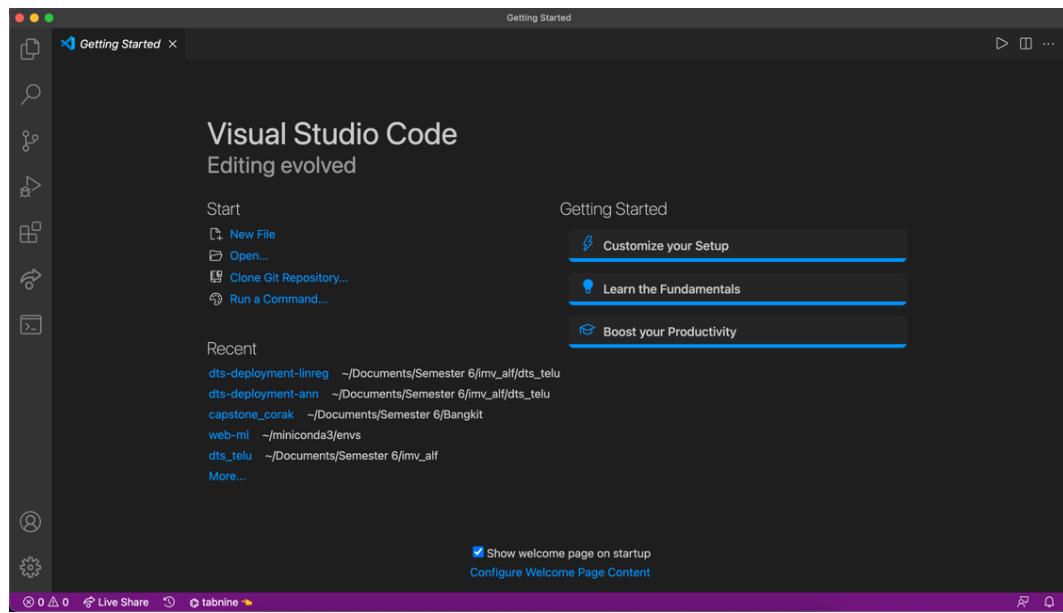
- d. Centang "Create a desktop icon" dan "Add to PATH", lalu klik **Next**.



- e. Klik tombol **Install** dan tunggu sampai proses *install* selesai.



- f. Klik tombol **Finish** dan Anda akan disajikan tampilan awal dari **VSCode**.

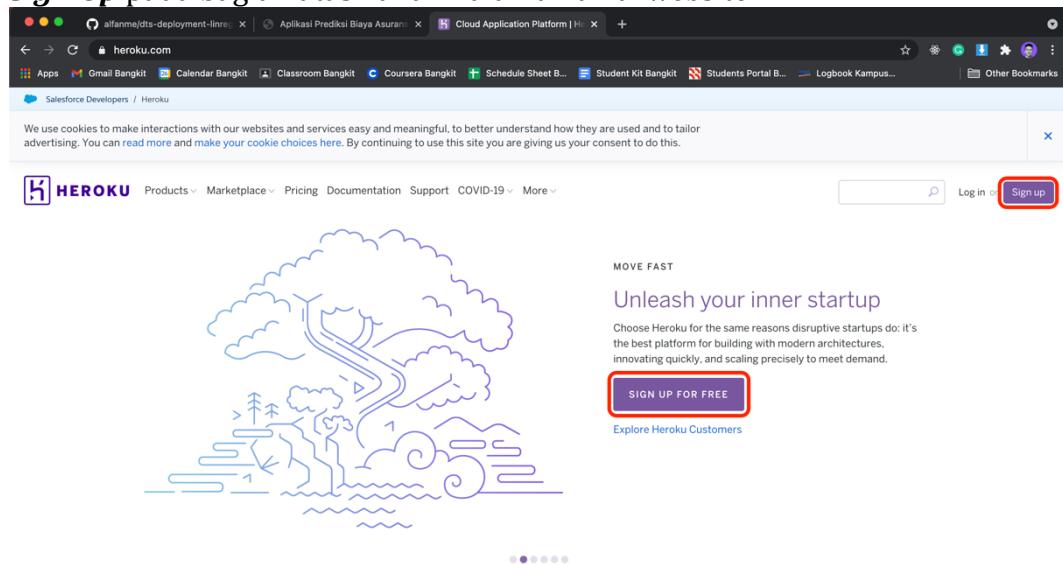


### 3. Heroku

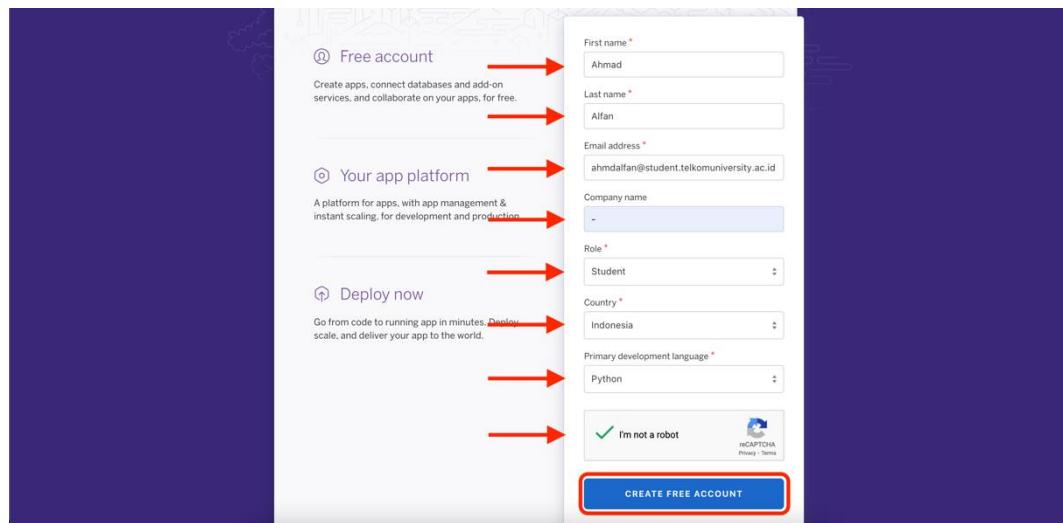
**Heroku** merupakan platform cloud yang menyediakan layanan *server* yang bisa dimanfaatkan untuk *deployment* berbagai aplikasi. **Heroku** mendukung banyak bahasa pemrograman seperti *Java*, *Node.js*, *Scala*, *Clojure*, *Python*, *PHP*, and *Go*. Pada modul ini, Anda akan mempelajari penggunaan **Heroku** untuk *deployment* model *Machine Learning* dengan bahasa *Python*.

#### Pendaftaran akun **Heroku**

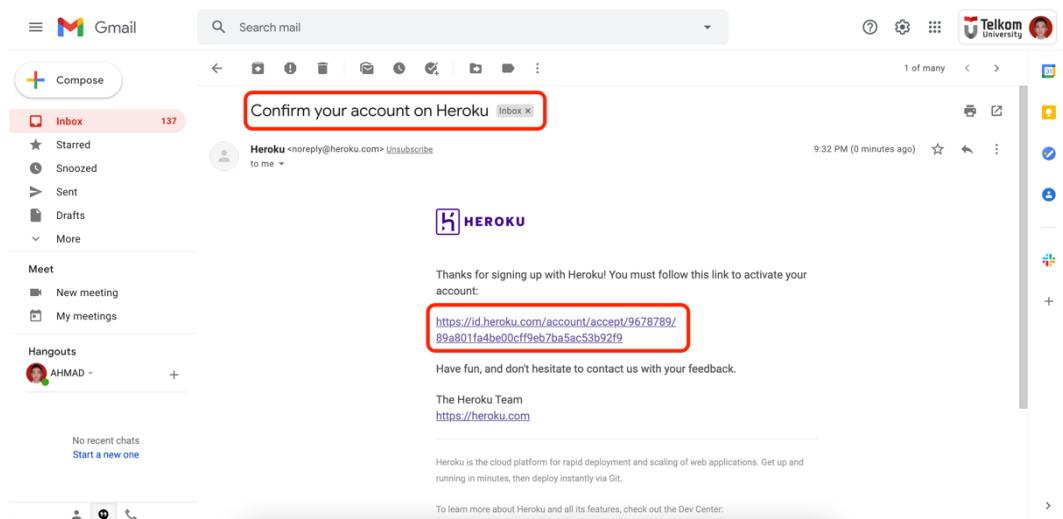
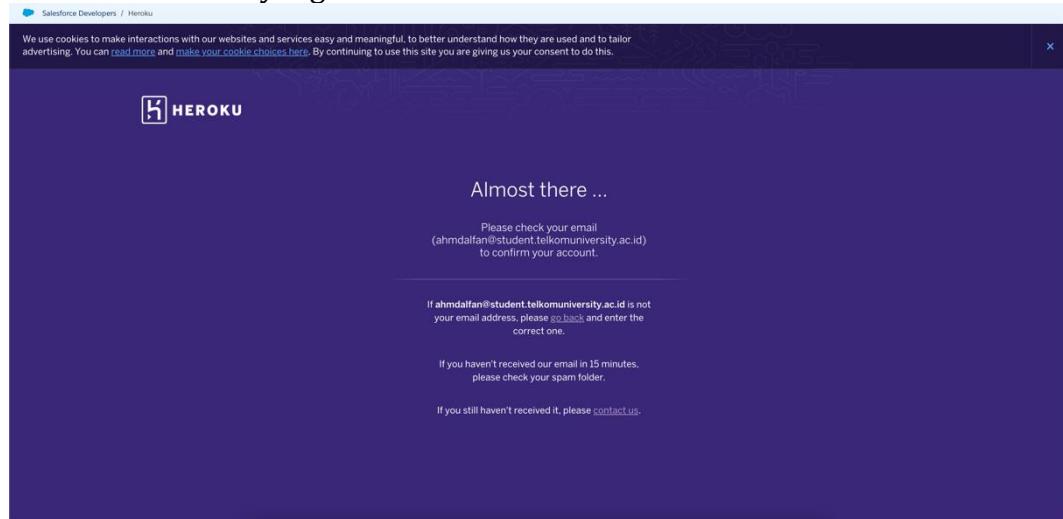
- Buka website resmi **Heroku** melalui URL [ini](https://heroku.com). Klik tombol **Sign Up for Free** atau **Sign Up** pada bagian atas kanan halaman awal website.



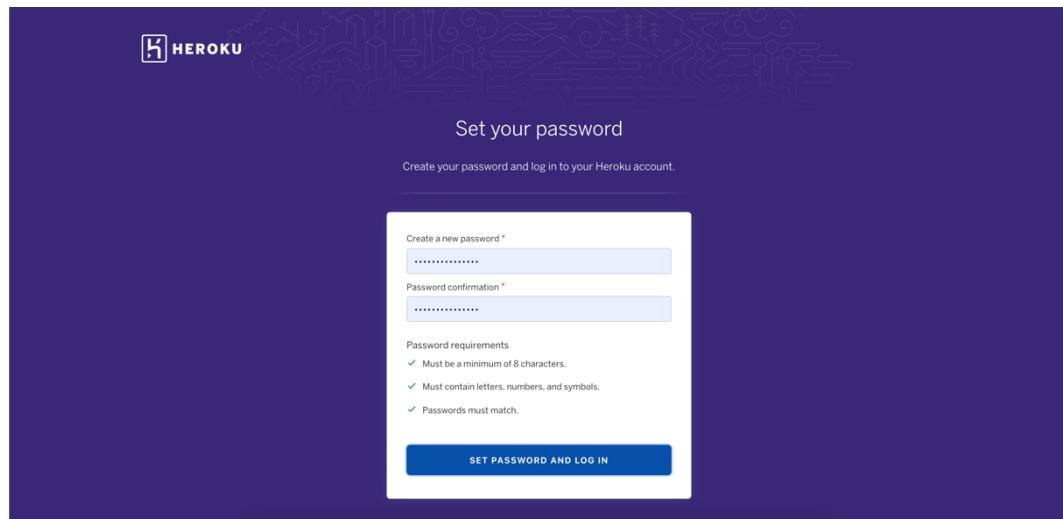
- Isi data pribadi secara lengkap, verifikasi "I'm not a robot" lalu tekan tombol **Create Free Account**.



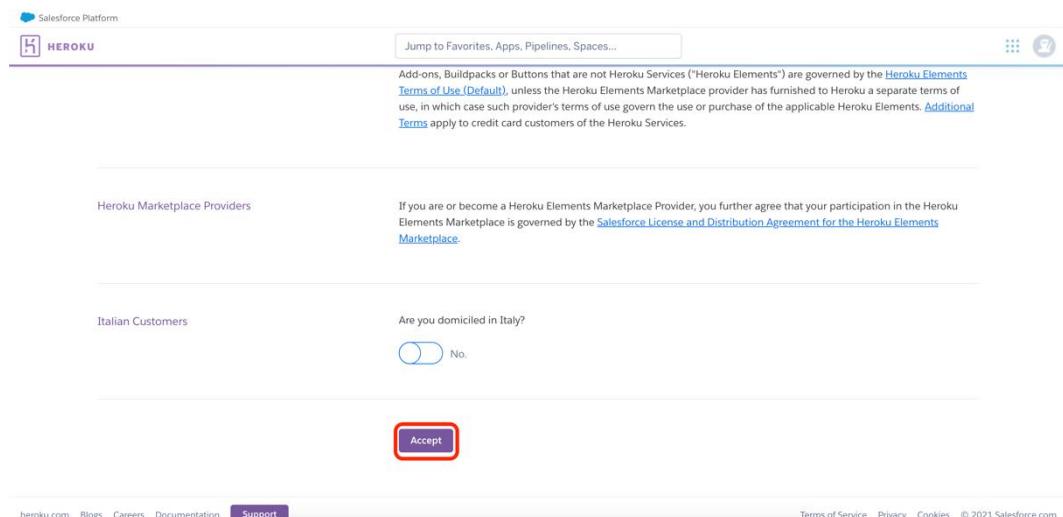
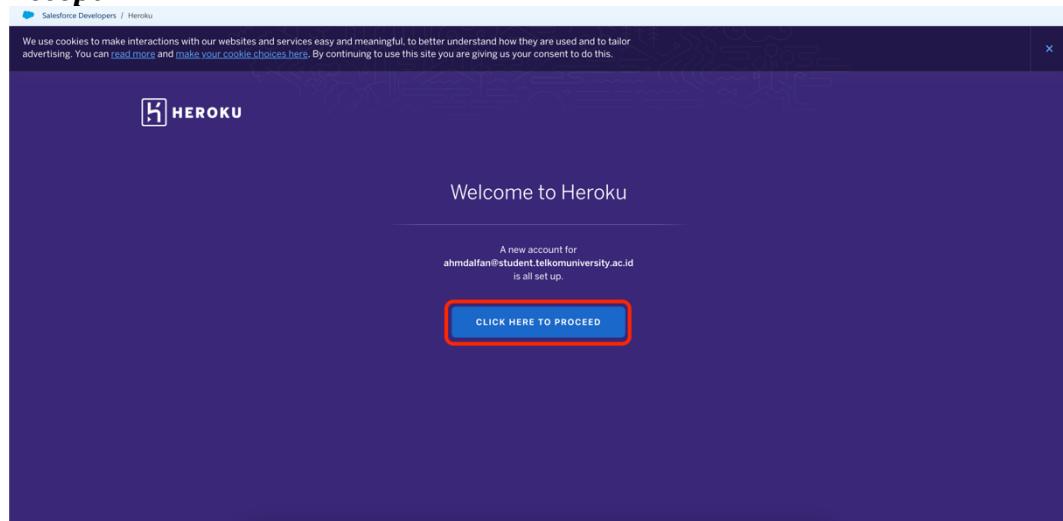
- c. Sekarang Anda diminta untuk konfirmasi akun melalui *email*. Buka *email* Anda lalu klik *email* terbaru dari **Heroku** dengan judul “*Confirm your account on Heroku*”. Klik *link* yang diberikan.



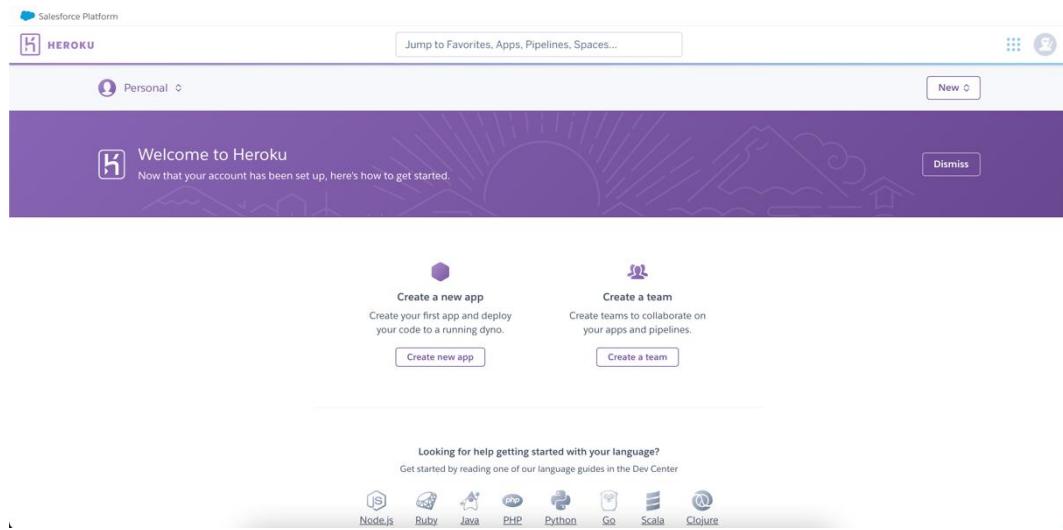
- d. Anda akan diarahkan ke halaman pengisian *password*. Masukkan *password* yang Anda inginkan lalu klik ***Set Password and Login***.



- e. Klik **Here to Proceed**. Setelah di arahkan ke halaman *License & Agreement*, klik **Accept**.



- f. Selesai. Anda akan diarahkan ke halaman awal *dashboard Heroku*.



### **Install Heroku CLI**

- Untuk melakukan *deployment*, Anda harus men-download dan menginstall **Heroku CLI** melalui [link berikut](#). Pilih bit *installer* sesuai PC/Laptop Anda (32 bit atau 64 bit).

#### Download and install

The Heroku CLI requires Git, the popular version control system. If you don't already have Git installed, complete the following before installing the CLI:

- Git installation
- First-time Git setup

Currently, the Windows installers may display a warning titled "Windows protected your PC". To run the installation when this warning is shown, click "More info", verify the publisher as "salesforce.com, inc", then click the "Run anyway" button.

- Periksa apakah **Heroku CLI** sudah ter-install dengan mengetikkan perintah `heroku --version` pada *terminal VSCode* lalu tekan **Enter**. Jika muncul informasi versi dari **Heroku CLI**, maka Anda dapat melanjutkan ke langkah berikutnya.

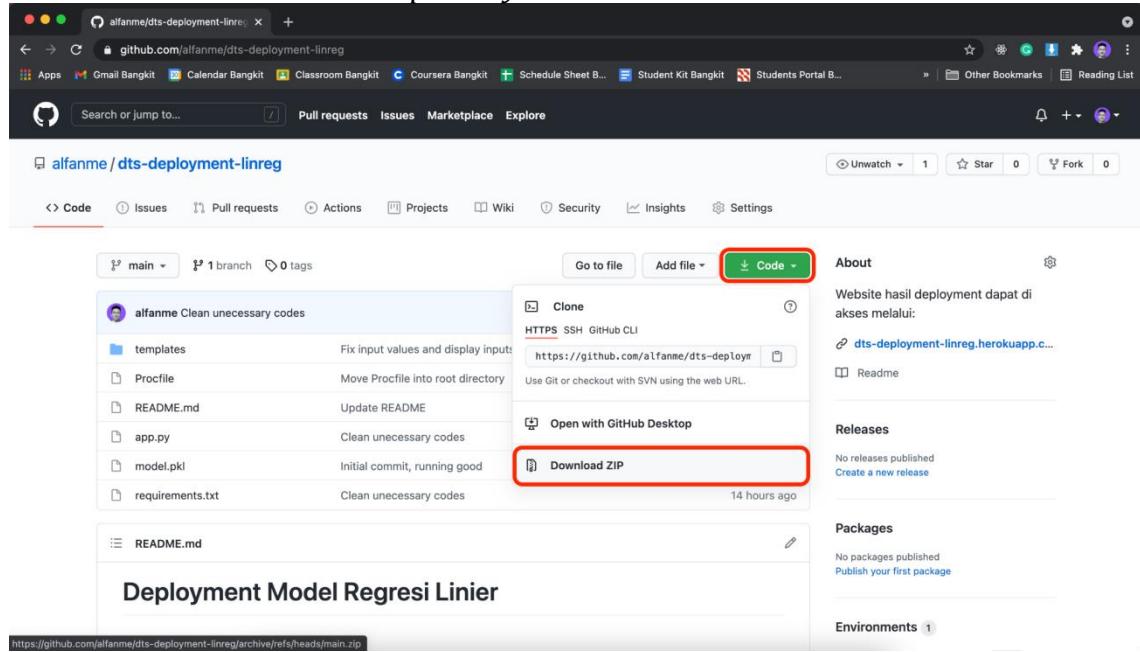
```
TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE
1: zsh + - ×
(deploy-model-linreg) ➜ dts-deployment-linreg-main heroku --version
heroku/7.53.0 darwin-x64 node-v15.6.0 ←
```

- Selamat, Anda berhasil menginstall **Heroku CLI**.

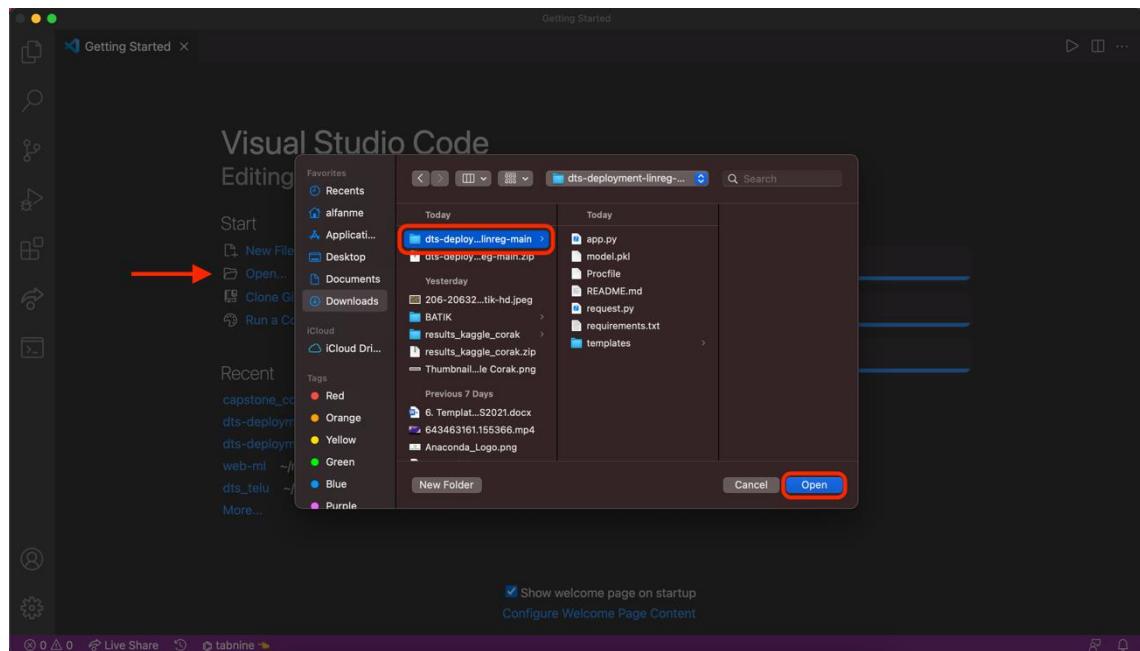
## G. Deployment Model (Regresi Linier): Langkah-Langkah Deployment Model Linear Regression

Setelah semua persiapan dilakukan, sekarang saatnya memulai proses *deployment* model, berikut langkah-langkahnya:

1. Download source code dari repository Github berikut



2. Buka dan extract file ZIP yang sudah di-download.
3. Buka folder source code yang sudah di-extract menggunakan **VSCode** dengan cara klik **Open...** pada halaman awal **VSCode**, pilih folder, lalu tekan tombol **Open**.



4. Buka terminal pada **VSCode** menggunakan **Ctrl+J**.
5. Pada terminal, ketikan `conda create --name deploy-model-linreg python=3.9` lalu tekan **Enter**. Anda bebas memberi nama apa saja selain "deploy-model-linreg", ketikan 'y' lalu **Enter** jika diberikan pertanyaan, tunggu proses pembuatan *virtual environment* selesai.

```
→ dts-deployment-linreg-main conda create --name deploy-model-linreg python=3.9
Collecting package metadata (current_repotdata.json): done
Solving environment: done
```

```
wheel          pkgs/main/noarch::wheel-0.36.2-pyhd3eb1b0_0
xz            pkgs/main/osx-64::xz-5.2.5-h1de35cc_0
zlib          pkgs/main/osx-64::zlib-1.2.11-h1de35cc_3

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
```

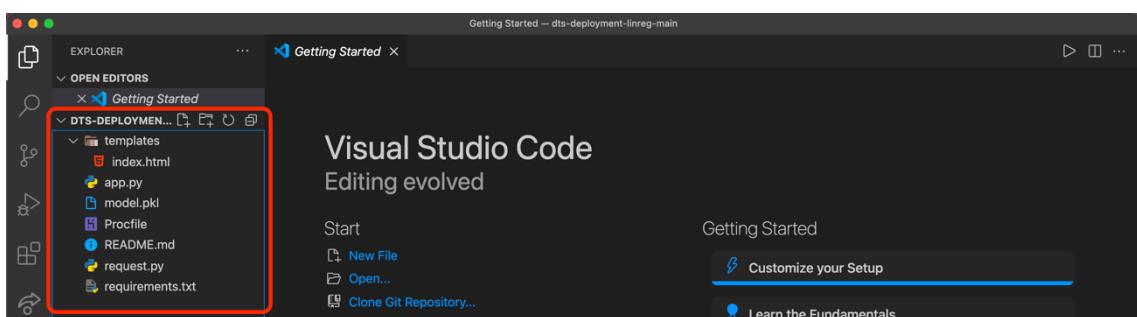
6. Aktifkan *virtual environment* menggunakan perintah `conda activate deploy-model-linreg` lalu tekan **Enter**, tunggu sampai *virtual environment* aktif, sekarang pada *terminal* akan terlihat nama *virtual environment* yang aktif yaitu (*deploy-model-linreg*).

```
→ dts-deployment-linreg-main conda activate deploy-model-linreg
(deploy-model-linreg) → dts-deployment-linreg-main
```

7. *Install* semua *dependencies/library* yang dibutuhkan dengan perintah `pip install -r requirements.txt`, tunggu sampai selesai.

```
(deploy-model-linreg) → dts-deployment-linreg-main pip install -r requirements.txt
Collecting flask
  Using cached Flask-2.0.1-py3-none-any.whl (94 kB)
Collecting gunicorn
```

8. Setelah semua *install* selesai, Anda bisa menyembunyikan *terminal* dengan menekan **Ctrl+J** atau klik icon 'x' pada sisi kanan atas *terminal*.
9. Pada sisi kiri **VSCode**, terdapat struktur direktori dari *folder* yang dibuka sebelumnya.



Ada satu buah *folder* “*templates*” dan beberapa *file* penting dengan kegunaan sebagai berikut.

- *templates/*
  - *index.html* → Berisi template *website*.

Berikut potongan kode yang perlu diperhatikan pada file *index.html*.

```
<form  
    action="{{ url_for('predict') }}"  
    method="post"  
    class="flex flex-col"  
>
```

Perintah `action="{{ url_for('predict') }}"` berfungsi memanggil *route/endpoint* dari *API Flask* yaitu `/predict`. Sedangkan `method="post"` menentukan jenis *HTTP request* yang dibuat. Post artinya *form* ini akan mengirimkan *payload* data ke *API* melalui *route* `/predict`.

```
<input  
    type="text"  
    name="Usia"  
    placeholder="Usia"  
    required="required"  
    class="p-4 bg-gray-100 rounded-md"  
>
```

*Tag input* ini berfungsi untuk menampilkan *text box* dan menerima pengisian data **usia**.

```
<select  
    name="Jenis Kelamin"  
    id="Jenis Kelamin"  
    class="p-4 bg-gray-100 rounded-md"  
>  
    <option value="Laki-laki">Laki-  
laki</option>  
    <option  
    value="Perempuan">Perempuan</option>  
</select>
```

*Tag select* ini berfungsi untuk menerima input berupa pilihan/*drop down*. Ada 2 buah opsi yang dapat dipilih yaitu Laki-laki atau Perempuan dengan

`value="Laki-laki"` dan `value="Perempuan"` yang akan menjadi nilai dari data **jenis kelamin**.

```
<select  
    name="Perokok"  
    id="Perokok"  
    class="p-4 bg-gray-100 rounded-md"  
>  
    <option value="Ya">Ya</option>  
    <option value="Tidak">Tidak</option>  
</select>
```

Tag *select* ini juga berfungsi untuk menerima input berupa pilihan/*drop down*. Ada 2 buah opsi yang dapat dipilih yaitu Ya atau Tidak dengan `value="Ya"` dan `value="Tidak"` yang akan menjadi nilai dari data **status perokok**.

```
<button  
    type="submit"  
    class="  
        flex  
        justify-center  
        ...  
    "  
>  
PREDIKSI SEKARANG  
...  
</button>
```

Tag ini berfungsi sebagai tombol/*button* dengan `type="submit"` yang berfungsi mengeksekusi *submission* dari *form* data **usia**, **jenis kelamin**, dan **status perokok**.

```
<h2 class="text-5xl font-bold">USD {{ insurance_cost }}</h2>
```

Pada tag *Heading2* atau *h2* inilah hasil prediksi ditampilkan melalui variable `{{ insurance_cost }}` yang dikirimkan oleh *endpoint/predict* dari *API Flask*.

```
{% if (insurance_cost) != 0 %}  
    <div class="mt-8">  
        <p>Usia: {{ age }} tahun</p>  
        <p>Jenis kelamin: {{ sex }}</p>
```

```
<p>Status perokok: {{ smoker }}</p>
</div>
{% endif %}
```

Ketika biaya asuransi berhasil diprediksi, maka data **usia**, **jenis kelamin**, dan **status perokok** hasil input pengguna akan ditampilkan juga.

- app.py → Berisi konfigurasi *route* untuk API.

Berikut potongan kode yang harus diperhatikan pada file app.py.

```
from flask import Flask, request, render_template
import pickle

app = Flask(__name__)

.
.
.

if __name__ == '__main__':
    app.run(debug=True)
```

Potongan kode di atas adalah bagian awal dan akhir file. Bagian ini sangat penting untuk memulai API Flask. Pada bagian awal terdapat perintah *import* untuk memanggil *library* yang dibutuhkan. Kemudian panggil kelas Flask dengan argumen `_name_` dan simpan di dalam variabel `app`. Di bagian akhir terdapat kondisi, jika `_name_` bernilai '`__main__`', maka jalankan API melalui perintah `app.run(debug=True)` yang sekaligus mengaktifkan mode *debug* untuk mempermudah proses pembuatan API. Jika terdapat *error* maka akan ditampilkan pada laman web yang terbuka di *browser*.

```
model_file = open('model.pkl', 'rb')
model = pickle.load(model_file, encoding='bytes')
```

Potongan kode di atas berfungsi membuka file model *Linear Regression* yang tersimpan dalam format **pickle** (.pkl) dan menyimpannya ke *variable* **model** untuk digunakan saat prediksi.

```
@app.route('/')
def index():
    return render_template('index.html', insurance_cost=0)
```

Setelah bagian inti yang berada di awal dan akhir file, selanjutnya adalah bagian di mana *API endpoints/routes* didefinisikan. Untuk *route* pertama yakni '/' berfungsi sebagai *home* atau *index* dari *API Flask* yang dibuat.

Jika pengguna mengakses route '/' contohnya **127.0.0.1/** maka API akan *render template* laman web pada file index.html yang sudah dijelaskan sebelumnya. Selain itu, *route* ini akan mengirimkan *variable insurance\_cost* dengan nilai awal 0 (sebelum dilakukannya prediksi).

```
@app.route('/predict', methods=['POST'])
def predict():
    ...
    Predict the insurance cost based on user inputs
    and render the result to the html page
    ...
```

*Endpoint/route* selanjutnya adalah */predict* yang menerima **HTTP request** berjenis **POST**. Pada route inilah data **usia**, **jenis kelamin**, dan **status perokok** akan diolah untuk memberikan prediksi nilai **insurance\_cost**.

```
age, sex, smoker = [x for x in request.form.values()]
```

Selanjutnya, semua *values* dari *form* akan diambil dan dimasukkan ke dalam *variable age, sex, dan smoker*.

```
data = []

data.append(int(age))
if sex == 'Laki-laki':
    data.extend([0, 1])
else:
    data.extend([1, 0])

if smoker == 'Ya':
    data.extend([0, 1])
else:
    data.extend([1, 0])
```

Setelah mendapatkan data. Kemudian data kategori yaitu **jenis kelamin** dan **status perokok** perlu dijadikan angka atau dilakukan *one hot encoding*. Untuk jenis kelamin akan dipecah menjadi dua kategori yaitu **[Perempuan, Laki-laki]**, bernilai 0 jika tidak dan bernilai 1 jika iya. Begitupun dengan **status perokok** yang menjadi **[Tidak, Ya]**.

Semua nilai akan digabungkan menjadi *list* [**Usia**, **Perempuan**, **Laki-laki**, **Tidak**, **Ya**] dan disimpan pada variabel **data**. Contoh hasilnya adalah [20, 0, 1, 1, 0].

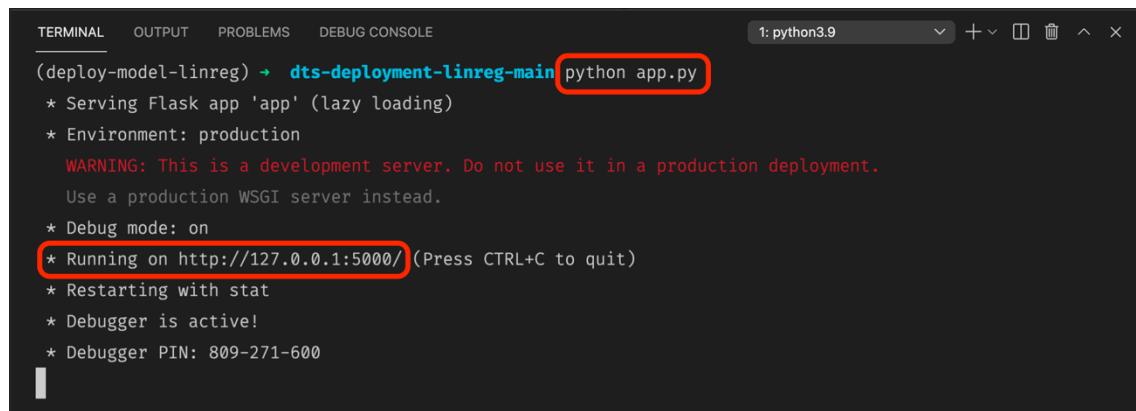
```
prediction = model.predict([data])
output = round(prediction[0], 2)

return render_template('index.html',
insurance_cost=output, age=age, sex=sex, smoker=smoker)
```

Pada bagian akhir fungsi predict() terdapat proses memprediksi biaya asuransi melalui perintah `model.predict([data])` yang hasilnya akan disimpan ke dalam *variable prediction*. Selanjutnya, nilai hasil prediksi dibulatkan menjadi 2 angka di belakang koma melalui perintah `round(prediction[0], 2)`. Terakhir, fungsi predict() akan me-*return* render dari template website beserta nilai **insurance\_cost** yang disimpan pada *variable output*. Selain itu, data input **age**, **sex**, dan **smoker** dikirimkan kembali untuk ditampilkan pada halaman *template*.

- `model.pkl` → Model Regresi Linier yang sudah di-*training* dan disimpan.
- `Procfile` → Berisi konfigurasi *Dyno formation* untuk **Heroku**.
- `requirements.txt` → Berisi daftar dependency/package *Python* yang diperlukan untuk menjalankan *API* dan model Regresi Linier.

10. Sebelum melakukan *deployment* ke platform **Heroku**, Anda dapat mencoba menjalankan *server websitenya* secara *local* pada PC/laptop dengan cara membuka *terminal* (Crtl+J), ketikan perintah `python app.py` lalu tekan **Enter**. Anda akan diberikan *URL* berupa `http://localhost:5000/` atau `http://127.0.0.1:5000/`.



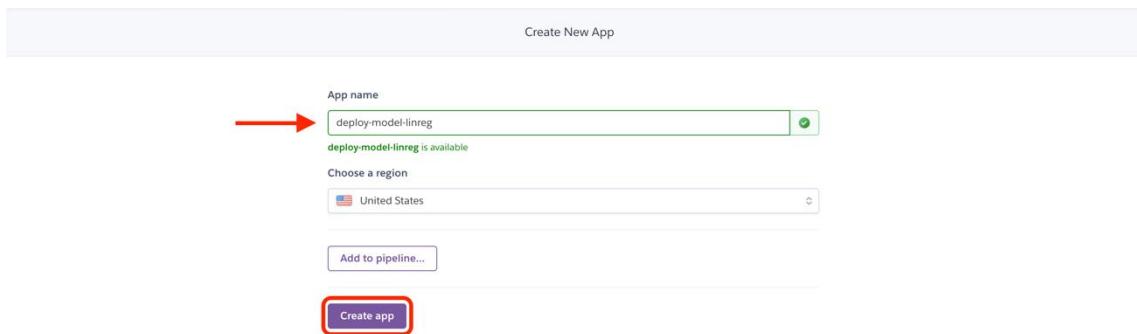
```
TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE
1: python3.9 + ×
(deploy-model-linreg) ➜ dts-deployment-linreg-main python app.py
* Serving Flask app 'app' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 809-271-600
```

11. Buka *URL* tersebut menggunakan *browser*, maka *websitenya* akan terbuka. Lakukan pengetesan dengan cara memasukkan data **Usia**, **Jenis Kelamin**, dan **Status Perokok**. Klik tombol **Prediksi Sekarang** dan lihat hasil prediksi biaya asuransinya seperti berikut.



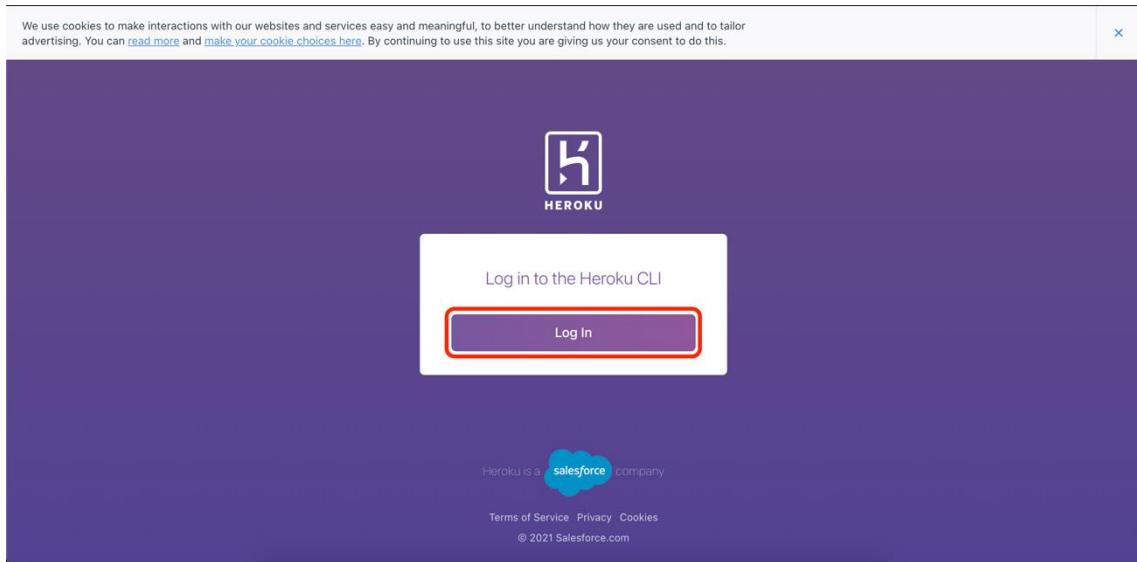
# DIGITAL TALENT SCHOLARSHIP



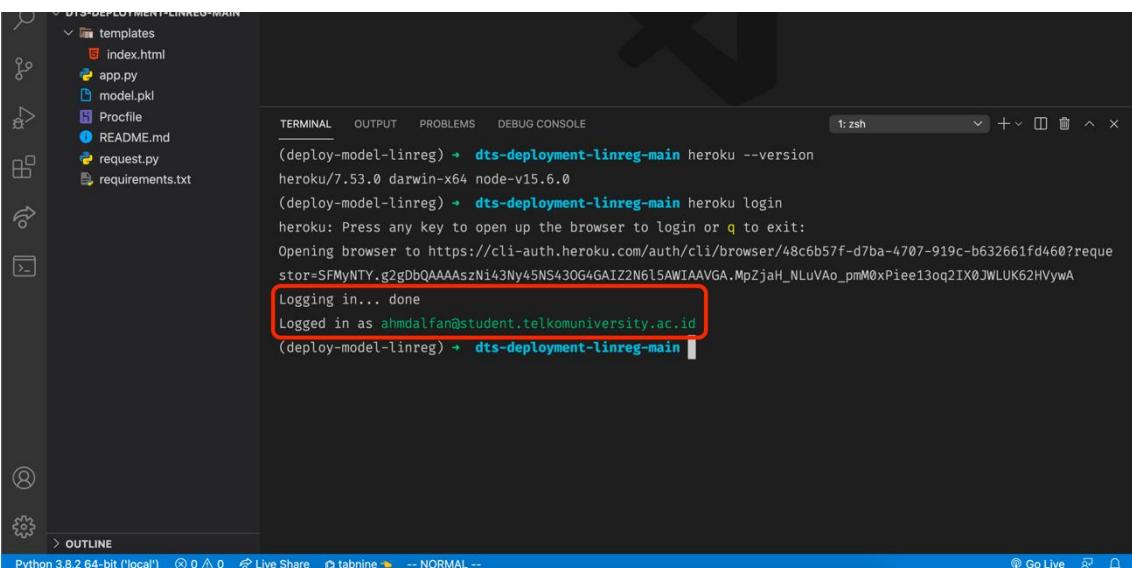
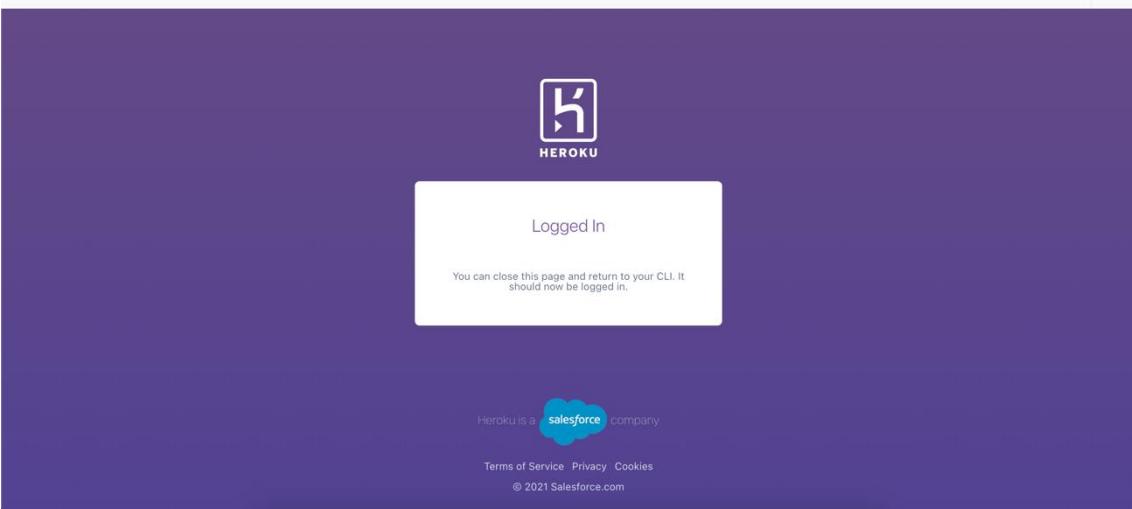


15. Login pada **Heroku CLI** dengan mengetikkan perintah `heroku login`, tekan **Enter**, lalu tekan sembarang kunci/tombol *keyboard* untuk membuka *Browser*. Anda akan diarahkan untuk login melalui *Browser*. Setelah selesai, buka kembali ke **VSCODE**.

```
(deploy-model-linreg) ➜ dts-deployment-linreg-main heroku login  
heroku: Press any key to open up the browser to login or q to exit: ←  
Opening browser to https://cli-auth.heroku.com/auth/cli/browser/48c6b57f-d7ba-4707-919c-b632661fd460?reque  
stor=SFMyNTY.g2gDbQAAAAszNi43Ny45NS430G4GAIZ2Nl5AWIAAVGA.MpZjaH_NLuVAo_pmM0xPiee13oq2IX0JWLUK62HVywA
```



We use cookies to make interactions with our websites and services easy and meaningful, to better understand how they are used and to tailor advertising. You can [read more](#) and [make your cookie choices here](#). By continuing to use this site you are giving us your consent to do this.



16. Buat git *repository* dan upload semua *file* yang ada ke **Heroku** dengan menggunakan beberapa perintah berikut secara berurutan satu persatu.

- git init

```
(deploy-model-linreg) ✘ dts-deployment-linreg-main git init
Initialized empty Git repository in /Users/alfanme/Downloads/dts-deployment-linreg-main/.git/
```

- heroku git:remote -a deploy-model-linreg

```
(deploy-model-linreg) ✘ dts-deployment-linreg-main git:(master) ✘ heroku git:remote -a deploy-model-linreg
set git remote heroku to https://git.heroku.com/deploy-model-linreg.git
```

- git add .

```
(deploy-model-linreg) ✘ dts-deployment-linreg-main git:(master) ✘ git add .
```

- git commit -am "deployment pertama"

```
(deploy-model-linreg) → dts-deployment-linreg-main git:(master) ✘ git commit -am "deployment pertama"
[master (root-commit) 15235aa] deployment pertama
 7 files changed, 270 insertions(+)
 create mode 100644 Procfile
 create mode 100644 README.md
 create mode 100644 app.py
 create mode 100644 model.pkl
 create mode 100644 request.py
 create mode 100644 requirements.txt
 create mode 100644 templates/index.html
```

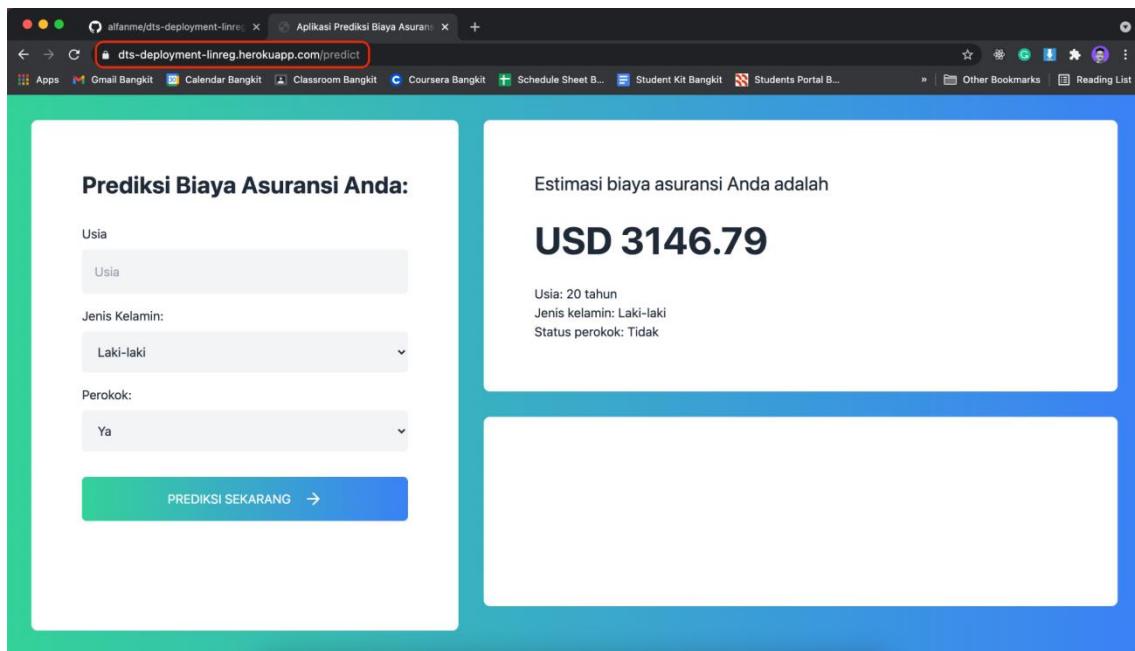
- git push heroku master

```
(deploy-model-linreg) → dts-deployment-linreg-main git:(master) git push heroku master
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 4.11 KiB | 701.00 KiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
remote:
```

17. Setelah selesai, Anda akan diberikan *URL* untuk mengakses *website* yang sudah *deploy* pada **Heroku**. Sekarang Anda dapat membagikan *URL* tersebut untuk diakses oleh banyak orang.

```
remote:      Installing collected packages: click, itsdangerous, Werkzeug, MarkupSafe, Jinja2, flask, guni
tl, joblib, scipy, scikit-learn, idna, chardet, certifi, urllib3, requests
remote:      Successfully installed Jinja2-3.0.1 MarkupSafe-2.0.1 Werkzeug-2.0.1 certifi-2020.12.5 chardet
-2.0.1 gunicorn-20.1.0 idna-2.10 itsdangerous-2.0.1 joblib-1.0.1 numpy-1.20.3 requests-2.25.1 scikit-learn-0
poolctl-2.1.0 urllib3-1.26.4
remote: -----> Discovering process types
remote:      Procfile declares types -> web
remote:
remote: -----> Compressing...
remote:      Done: 124.6M
remote: -----> Launching...
remote:      Released v3
remote:      https://deploy-model-linreg.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/deploy-model-linreg.git
 * [new branch]      master -> master
(deploy-model-linreg) → dts-deployment-linreg-main git:(master)
```

18. Selamat, Anda telah berhasil melakukan *deployment* model *Linear Regression* dalam bentuk *website* menggunakan **Heroku**!



## H. Deployment Model (ANN): Langkah-Langkah *Deployment Model ANN Image Classification (Cat vs Dog)*

Sampai di sini, Anda telah berhasil melakukan *deployment* untuk model *Linear Regression*. Untuk men-deploy model *ANN Image Classification*, tidak ada perbedaan langkah-langkah kecuali pada struktur *directory file* dan kode program. Berikut beberapa perbedaan yang harus diperhatikan.

1. Semua *file deployment* model *ANN* dapat di-download pada *repository Github* [berikut](#). Pada bagian halaman *Github* juga terdapat deskripsi mengenai *folder*, *file*, dan kode program.

<https://github.com/alfanme/dts-deployment-ann>

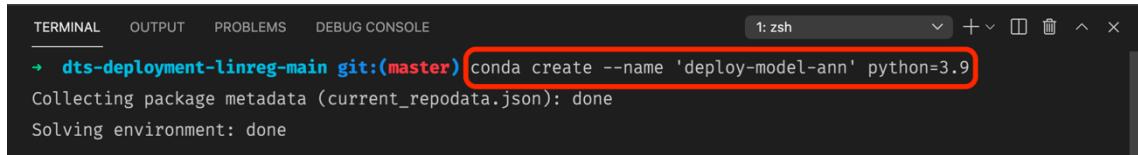
Code dropdown menu:

- Clone
- HTTPS SSH GitHub CLI
- <https://github.com/alfanme/dts-deployment-ann>
- Use Git or checkout with SVN using the web URL...
- Open with GitHub Desktop
- Download ZIP

Description:

Website hasil deployment dapat diakses melalui:  
<https://dts-deployment-ann.herokuapp.com/>

2. Pastikan Anda menggunakan *virtual environment* **Anaconda** yang berbeda. Nama *virtual environment* sebelumnya adalah “deploy-model-linreg”, Anda dapat membuat *virtual environment* baru dengan nama “deploy-model-ANN” atau nama lainnya sesuai keinginan.



```
TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE
1: zsh
→ dts-deployment-linreg-main git:(master) conda create --name 'deploy-model-ann' python=3.9
Collecting package metadata (current_repodata.json): done
Solving environment: done
```



```
TERMINAL OUTPUT PROBLEMS DEBUG CONSOLE
1: zsh
→ dts-deployment-linreg-main git:(master) conda activate deploy-model-ann
(deploy-model-ann) → dts-deployment-linreg-main git:(master)
```

3. Struktur *directory* program adalah sebagai berikut:

- static/
  - uploads/ → Berisi gambar yang diunggah untuk diprediksi.
- templates/
  - index.html → Berisi template *website*.

Perhatikan beberapa potongan kode pada file index.html berikut.

```
{% if uploaded_image %}
    
{% else %}
    <div
        class="
            flex
            justify-center
            items-center
            my-8
            w-48
            h-48
            border-8 border-gray-200
            rounded-md
        "
    >
        <p class="text-center">
            Masukkan gambar kucing atau anjing
        </p>
    </div>
{% endif %}
```

```
</div>  
{% endif %}
```

Pada bagian ini terdapat percabangan *if-else* untuk mengatur tampilan laman web. Jika gambar yang di-*upload* ada, maka tampilkan gambar menggunakan tag **img**. Jika belum ada gambar yang di-*upload*, maka tampilkan tulisan “Masukkan gambar kucing atau anjing”.

```
{% if prediction %}  
    <p class="mb-8">Hasil Prediksi:  
    {{prediction}}</p>  
{% endif %}
```

Kemudian terdapat juga percabangan *if* untuk menampilkan hasil prediksi apabila hasil prediksi sudah ada, lalu ditampilkan ke laman web. Contoh: “Hasil Perdiksi: Cat (Kucing)”.

```
<form  
    method="post"  
    enctype="multipart/form-data"  
    class="flex flex-col"  
>  
    <input  
        type="file"  
        name="image"  
        class=""  
            mb-4  
            p-4  
            border-8 border-gray-200  
            rounded-md  
            bg-gray-100  
        "  
    />  
    <button  
        type="submit"  
        class="p-4 bg-indigo-500 text-white  
rounded-md"  
    >  
        Prediksi Gambar  
    </button>  
</form>
```

Bagian ini merupakan *form* yang berfungsi untuk *upload* gambar. Tag *input* berfungsi membuka *file explorer* untuk memilih gambar yang ingin di *upload*. Ketika gambar selesai dipilih, selanjutnya *button Prediksi Gambar* berfungsi mengirimkan gambar ke *API Flask* untuk diolah dan diprediksi.

- app.py → Berisi konfigurasi *route* dan proses prediksi model untuk *API*.

```
from flask import Flask, render_template, request,
send_from_directory
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array,
load_img
from tensorflow import expand_dims
import numpy as np
import os

app = Flask(__name__)

.

.

.

if __name__ == '__main__':
    app.run(debug=True)
```

Potongan kode di atas adalah bagian awal dan akhir file. Bagian ini sangat penting untuk memulai API Flask. Pada bagian awal terdapat perintah *import* untuk memanggil *library* yang dibutuhkan. Kemudian panggil kelas Flask dengan argumen *\_name\_* dan simpan di dalam variabel *app*. Di bagian akhir terdapat kondisi, jika *\_name\_* bernilai '*\_main\_*', maka jalankan API melalui perintah *app.run(debug=True)* yang sekaligus mengaktifkan mode *debug* untuk mempermudah proses pembuatan API. Jika terdapat *error* maka akan ditampilkan pada laman web yang terbuka di *browser*.

```
app.config['UPLOAD_FOLDER'] = './static/uploads/'
model = load_model('model.h5')

class_dict = {0: 'Cat (Kucing)', 1: 'Dog (Anjing)'}
```

Bagian ini merupakan konfigurasi folder penyimpanan gambar ter-*upload*, membuka file *model.h5* dan menyimpannya ke *variable* *model* untuk prediksi. Kemudian, membuat **class\_dict** yang berisi *encoding* untuk 2 label klasifikasi yakni 0 untuk kucing dan 1 untuk anjing.

```

def predict_label(img_path):
    loaded_img = load_img(img_path, target_size=(256, 256))
    img_array = img_to_array(loaded_img) / 255.0
    img_array = expand_dims(img_array, 0)
    predicted_bit =
    np.round(model.predict(img_array)[0][0]).astype('int')
    return class_dict[predicted_bit]

```

Fungsi **predict\_label** dengan parameter **img\_path** berfungsi untuk memprediksi kucing atau anjing dari gambar yang di-upload. Pertama gambar dibuka dengan perintah **load\_img** sekaligus di-resize menjadi 256x256 pixel. Ubah gambar menjadi array dan dinormalisasi dengan membagi semua pixel dengan nilai 255.0. Sebelum dilakukan prediksi, dimensi gambar diubah terlebih dahulu dengan menambahkan 1 dimensi pada indeks 0 sehingga ukuran akhir array gambar adalah (1, 256, 256, 3). Terakhir dilakukan prediksi dengan menggunakan **model.predict()** dan hasil prediksi dibulatkan menjadi *integer* sehingga nilai nya hanya 0 atau 1. Fungsi selanjutnya mengembalikan (*return*) label kelas dari **class\_dict** berupa “Cat (Kucing)” atau “Dog (Anjing)”

```

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        if request.files:
            image = request.files['image']
            img_path =
os.path.join(app.config['UPLOAD_FOLDER'], image.filename)
            image.save(img_path)
            prediction = predict_label(img_path)
            return render_template('index.html',
uploaded_image=image.filename, prediction=prediction)

    return render_template('index.html')

```

Bagian ini merupakan *endpoint home/index* dari *API Flask* yang dibuat. Route ‘/’ akan mengembalikan render template index.html saja jika laman web dibuka (*request.method == ‘GET’*). Jika *request.method == ‘POST’*, berarti laman web mengirimkan gambar untuk diprediksi. Simpan gambar tersebut ke *folder upload* yang sudah di-*config* beserta nama *file* gambarnya. Lakukan prediksi dengan memanggil fungsi **predict\_label()** yang sudah dijelaskan sebelumnya dengan menggunakan argumen **img\_path** untuk memberikan *path* atau lokasi file *upload* gambar yang tersimpan. Terakhir, *return* render template index.html sekaligus nama *file* gambar dan hasil prediksi untuk ditampilkan kembali ke laman web.

```

@app.route('/display/<filename>')

```

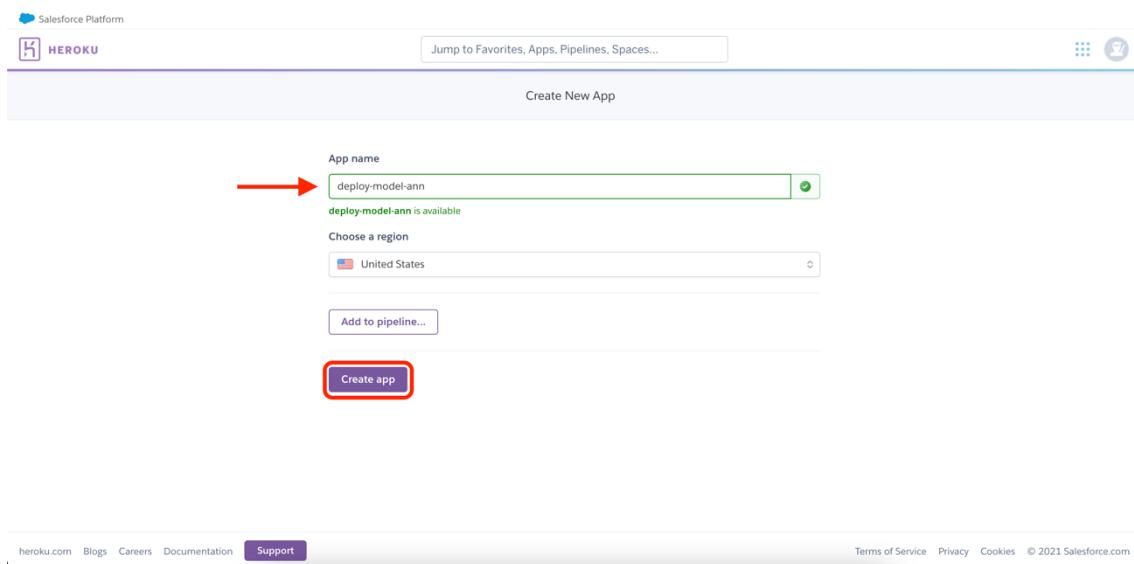
```

def send_uploaded_image(filename=''):
    return send_from_directory(app.config['UPLOAD_FOLDER'],
filename)

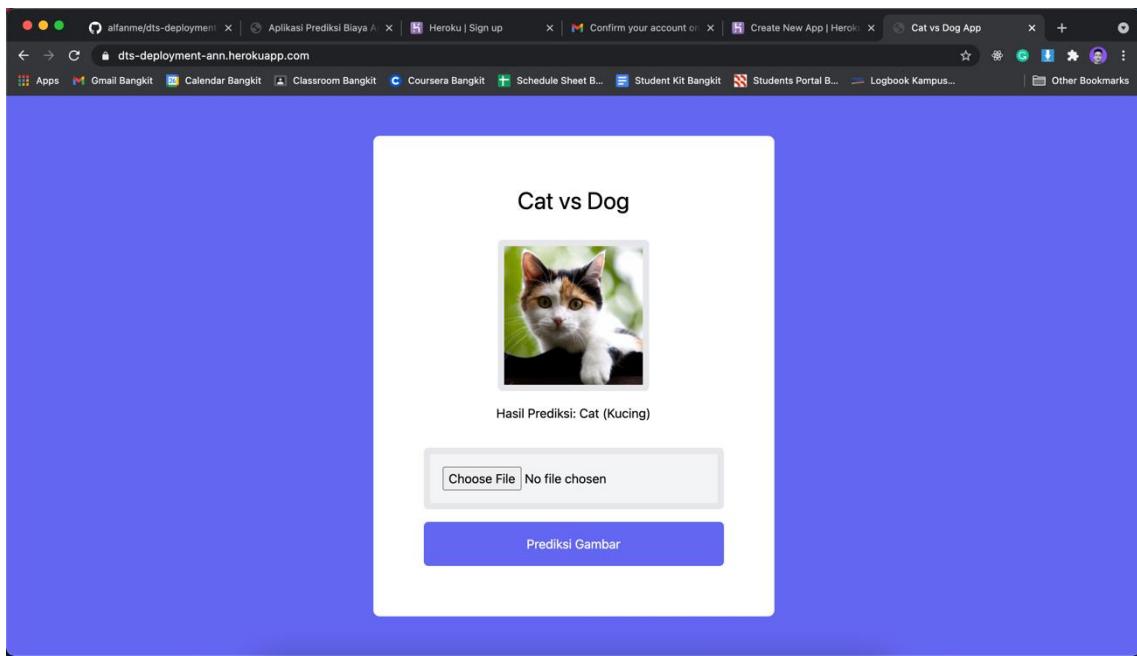
```

Kemudian ada *endpoint atau route* '/display/<filename>' agar laman web dapat membuka *path* gambar yang dikirimkan melalui *link* yang disediakan *endpoint* ini. Pada file index.html, Anda dapat melihat terdapat potongan kode `src="{{ url_for('send_uploaded_image', filename=uploaded_image) }}"` pada bagian inilah **send\_uploaded\_image** atau route '/display/<filename>' dipanggil.

- model.h5 → Model Image Classification *ANN* yang sudah di-training.
  - Procfile → Berisi konfigurasi Dyno formation untuk Heroku.
  - requirements.txt → Berisi daftar dependency/package *Python* yang diperlukan untuk menjalankan *API* dan model Image Classification *ANN*.
4. Pastikan Anda membuat project/app baru pada dashboard **Heroku**. Anda dapat memberi nama app-nya dengan "deploy-model-ANN" atau "cat-vs-dog" atau nama lainnya sesuai keinginan.



5. Jika *website* model *Linear Regression* menerima input data **usia, jenis kelamin, dan status perokok** untuk memprediksi biaya asuransi. Pada *website* model *ANN* ini akan menerima input berupa gambar kucing atau anjing yang di-upload dan memprediksi apakah gambar tersebut merupakan gambar kucing atau anjing seperti berikut ini.



Setelah memahami perbedaan antara *deployment* model *Linear Regression* dan *ANN*, maka Anda dapat memulai langkah-langkah yang sama seperti *deployment Linear Regression*. Selamat mencoba! Jika berhasil, buka *URL website* yang diberikan **Heroku** menggunakan *Browser*. Masukkan gambar kucing atau anjing, lalu lihat hasil prediksinya. Sekarang Anda dapat membagikan *URL* tersebut kepada orang lain.