

Stroke Prediction Project using Machine Learning on Google Cloud Platform

From Development to Production: with Flask, Docker, and Kubernetes

Dr. Aygul Zagidullina

April 4, 2024

Introduction

Using Python to build a predictive model on GCP to determine whether a patient has a high risk of stroke or not

What will we learn?

- What are Cloud Repositories in GCP and how to create them?
- What is Docker, DockerHub? How to create and build Docker images?
- What is Kubernetes and how to use Google Kubernetes Engine?
- How to push Docker images to Google Container Registry and create Kubernetes cluster?
- What is Flask and how to deploy and run Flask application on GKE cluster?

Tech Stack

Language: Python

Packages: Pandas, Numpy, Scikit-Learn

Services: Flask, Docker

Cloud: GCP

- Google Cloud Repositories
- Google Container Registry
- Google Kubernetes Engine



docker



kubernetes



Flask



`Flask` is a lightweight web framework also built with Python. It's designed to make web development quick and straightforward.

We use Flask to build a web service that exposes our machine learning model's predictions as an API. This will enable other applications to interact with our model easily.

`pip install Flask`

<https://pypi.org/project/Flask/>

Docker simplifies the process of packaging an application and its dependencies into a standardized container. **Containers** ensure that our application runs consistently across various environments, making deployment smoother.

We use Docker to encapsulate our Flask application, along with its dependencies, into a portable container.

<https://www.docker.com/get-started/>

Tech Stack Overview

Kubernetes is a container management system that allows to create, scale and monitor containers/containerized applications across a **cluster of machines**. It is an open-source software and was developed by Google.

Features:

- **Load Balancing:** automatically (re-) distributes the load between containers.
- **Self-healing:** it restarts containers that fail, replaces containers, kills containers that don't respond to user-defined checks
- **Scaling:** automatically scale up or down the application (e.g. during peak hours)

<https://kubernetes.io/docs/concepts/overview/>

Tech Stack Overview

Life of an application

How do you package and distribute an application?



How do you scale, run and monitor an application?



Lifecycle of an application deployed through Kubernetes / Docker

We use GCP's resources to set up a Kubernetes cluster using Google Kubernetes Engine (GKE)

Self-Check: Understanding Containerization

- **Container:**

Container is a type of software that packages up an application and all its dependencies so the application runs reliably from one computing environment to another.

- **Docker:**

Docker is a software used for building and managing containers.

- **Kubernetes:**

Kubernetes is an open-source system for managing containerized applications in a clustered environment.

- **Google Kubernetes Engine (GKE):**

Google Kubernetes Engine is an implementation of the open-source Kubernetes framework on Google Cloud Platform.

Data, Model, App, Deployment: Tasks

Tasks:

- Train and develop a machine learning pipeline for deployment.
- Build a web application using a Flask framework. It will use the trained ML pipeline to generate predictions on new data points in real-time.
- Create a new repository on Google Cloud Source Repository
- Activate Cloud Shell, copy the created repository
- Define the variables, create Docker image of the application and push it to .Google Container Registry (GCR)
- Create clusters and deploy the application on Google Kubernetes Engine

Data and Model: Live Demo

Kaggle Stroke Dataset

Attribute Information

- 1) id: unique identifier
- 2) gender: "Male", "Female" or "Other"
- 3) age: age of the patient
- 4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
- 5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- 6) ever_married: "No" or "Yes"
- 7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
- 8) Residence_type: "Rural" or "Urban"
- 9) avg_glucose_level: average glucose level in blood
- 10) bmi: body mass index
- 11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"
- 12) stroke: 1 if the patient had a stroke or 0 if not

```
def create_pipeline():  
    # Create a pipeline with preprocessing and model  
    preprocessor = create_data_preprocessor()  
    pipeline = Pipeline([  
        ('preprocessor', preprocessor),  
        ('model', DecisionTreeClassifier(random_state=42))  
    ])  
    return pipeline
```

Data, Model, App: Live Demo

Flask Docs & Minimal Example

Swagger GitHub repo & Docs

The screenshot shows the Swagger UI for an API. The browser address bar displays `http://127.0.0.1:5000/apidocs/#/default/post_forecast`. The Swagger logo is in the top left, and a search bar with `/apispec_1.json` and an `Explore` button is in the top right. The main heading is **A swagger API** with a version tag `0.0.1`. Below it, it says `/apispec_1.json`, `powered by Flask`, and `Terms of service`. A dropdown menu is set to `default`. Two endpoints are listed: `GET /forecast` (Predict stroke probability for new data) with operation `get_forecast`, and `POST /forecast` (Predict stroke probability for new data) with operation `post_forecast`. The `POST /forecast` endpoint is selected, showing its parameters. A `Cancel` button is in the top right of the parameters section. The parameters table has two columns: `Name` and `Description`.

Name	Description
age * required number (formData)	<input type="text" value="age"/>
hypertension * required number (formData)	<input type="text" value="hypertension"/>
heart_disease * required number (formData)	<input type="text" value="heart_disease"/>
avg_glucose_level * required number (formData)	<input type="text" value="avg_glucose_level"/>
bmi * required number (formData)	<input type="text" value="bmi"/>

Useful commands:

```
python main.py
```

```
python app.py
```

```
docker build -t api .
```

```
docker run -p 5000:5000 api
```

```
docker images
```

```
docker rmi -f XXX
```

```
docker ps
```

Google Cloud CLI Documentation

Install the [gcloud](#) CLI to communicate with Google Cloud Platform through your terminal:

Google Cloud Command Line Interface

Configure Cloud sdk

- Authenticate the `gcloud` CLI with the google account you used for GCP

```
gcloud auth login --no-launch-browser
```

- Login to your Google account on the new tab opened in your web browser
- List your active account and check your email address you used for GCP is present

```
gcloud auth list
```


- Set your current project (replace `PROJECT_ID` with the `ID` of your project, e.g. `DSPROZ`)

```
gcloud config set project PROJECT_ID
```



- List your active account and current project and check your project is present


```
gcloud config list
```

Cloud Source Repositories

 Cloud Source Repositories

stroke_prediction_app_2 ▾

 Search for code or files 

 Your repository is currently empty. Add some code using a selected method and then refresh your browser. Contents added to this repository can take some time to show up in search results. [Learn more.](#)

Select an option to push code to your repository:

☐ Push code from a local Git repository

☒ Clone your repository to a local Git repository

Select your preferred authentication method

SSH authentication Google Cloud SDK Manually generated credentials

1. Install the [Google Cloud SDK](#).
2. Provide your authentication credentials:

```
$ gcloud init
```
3. Clone this repository to a local Git repository:

```
$ gcloud source repos clone stroke_prediction_app_2 --project=dspro2-417019
```

Note: This may display the following message that is safe to ignore:
"Warning: Remote HEAD refers to a non-existent ref, unable to check."
4. Switch to your new local Git repository:

```
$ cd stroke_prediction_app_2
```
5. After you've committed code to your local Git repository, push it to this repository:

```
$ git push -u origin master
```


Cloud Console

 Clone

Google Cloud SDK ▾

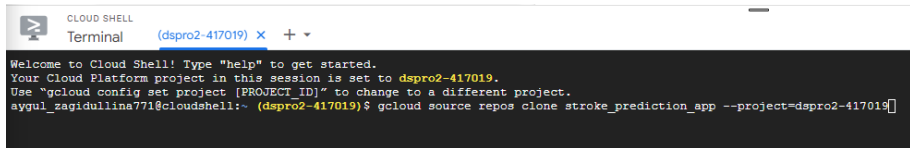
 How to set up

Clone with command line

```
$ gcloud source repos clone stroke_pred_ 
```

Cloud Shell Commands: Docker

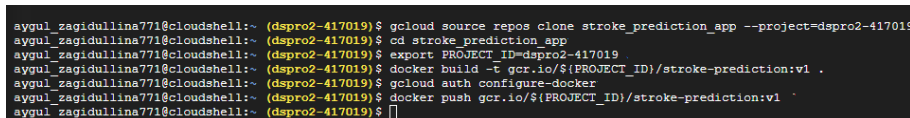
- Copy the repository:



The screenshot shows a Cloud Shell terminal window with the title "Terminal (dspro2-417019)". The terminal text is as follows:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to dspro2-417019.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ gcloud source repos clone stroke_prediction_app --project=dspro2-417019
```

- Define the project ID variable
- Create the Docker image and assign a tag to it
- Provide Google authentication to be able to push Docker image to Google Container Registry
- Docker image to Registry



The screenshot shows the continuation of the Cloud Shell terminal window with the following commands and output:

```
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ gcloud source repos clone stroke_prediction_app --project=dspro2-417019
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ cd stroke_prediction_app
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ export PROJECT_ID=dspro2-417019
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ docker build -t gcr.io/${PROJECT_ID}/stroke-prediction:v1 .
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ gcloud auth configure-docker
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $ docker push gcr.io/${PROJECT_ID}/stroke-prediction:v1
aygul_zagidullina771@cloudshell:~ (dspro2-417019) $
```

Cloud Shell Commands: GKE

- Set up project configurations:

```
aygul_zagidullina771@cloudshell:~ (dspro2-417019)$ gcloud config set project $PROJECT_ID  
aygul_zagidullina771@cloudshell:~ (dspro2-417019)$ gcloud config set compute/zone asia-northeast3-b  
aygul_zagidullina771@cloudshell:~ (dspro2-417019)$ gcloud container clusters create stroke-cluster --num-nodes=2
```

- Deploy application on Kubernetes cluster
- Expose deployed application on the port 5000
- Verify that services are running

```
cloudshell:~ (dspro2-417019)$ gcloud config set project $PROJECT_ID  
cloudshell:~ (dspro2-417019)$ gcloud config set compute/zone asia-northeast3-b  
cloudshell:~ (dspro2-417019)$ gcloud container clusters create stroke-cluster --num-nodes=2
```

- Kubernetes will show [external IP address](#) once the application is deployed. We can use it together with the port to access the Swagger API.

Cloud Shell Commands: Finalize

Don't forget to check which clusters are running and delete them (not to incur any charges)!

```
cloudshell:~ (dspro2-417019)$ gcloud container clusters list
cloudshell:~ (dspro2-417019)$ gcloud container clusters delete stroke-cluster
```