Multi-Container App on AWS

Building Multi Container Application

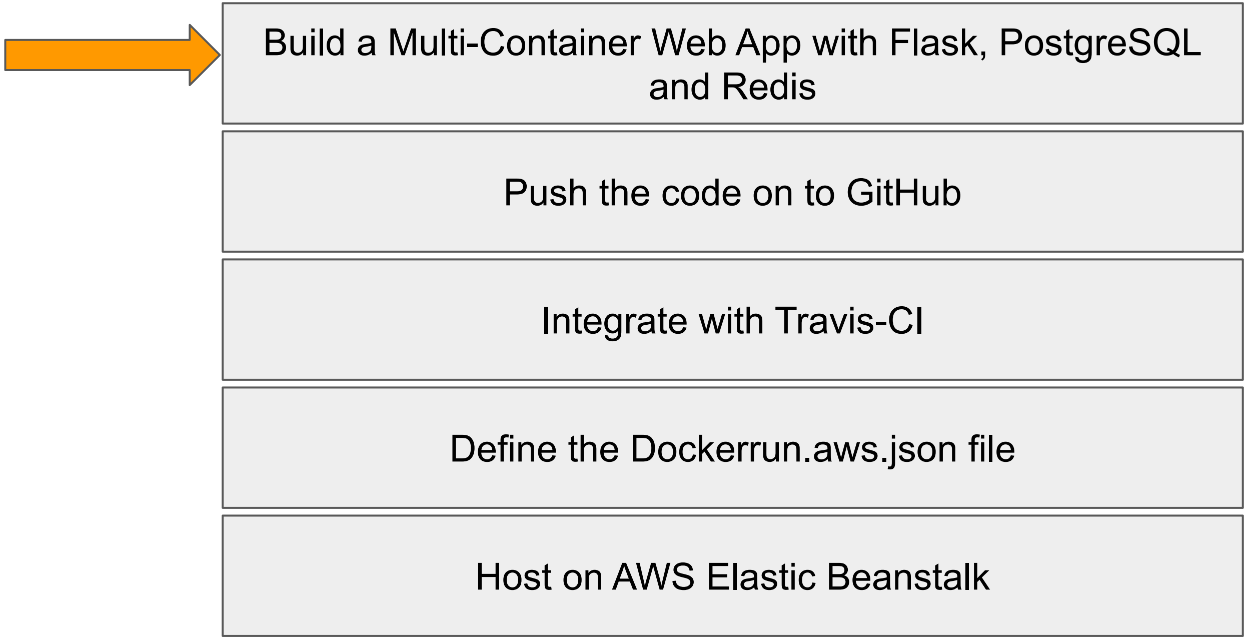
- Previously, we have deployed single container application
- A real-world, web application requires various services and each service runs on a separate container

PostgreSQL
redis
N
uWSGI
Flask

Building Multi Container Application

- Let’s build multi container application
- This application uses
  - PostgreSQL – As Database
  - Redis – Caching
  - Nginx – As reverse proxy web server
  - uWSGI – a WSGI server
  - Flask – As web application framework

Multi-Container App on AWS



Build a Multi-Container Web App with Flask, PostgreSQL and Redis

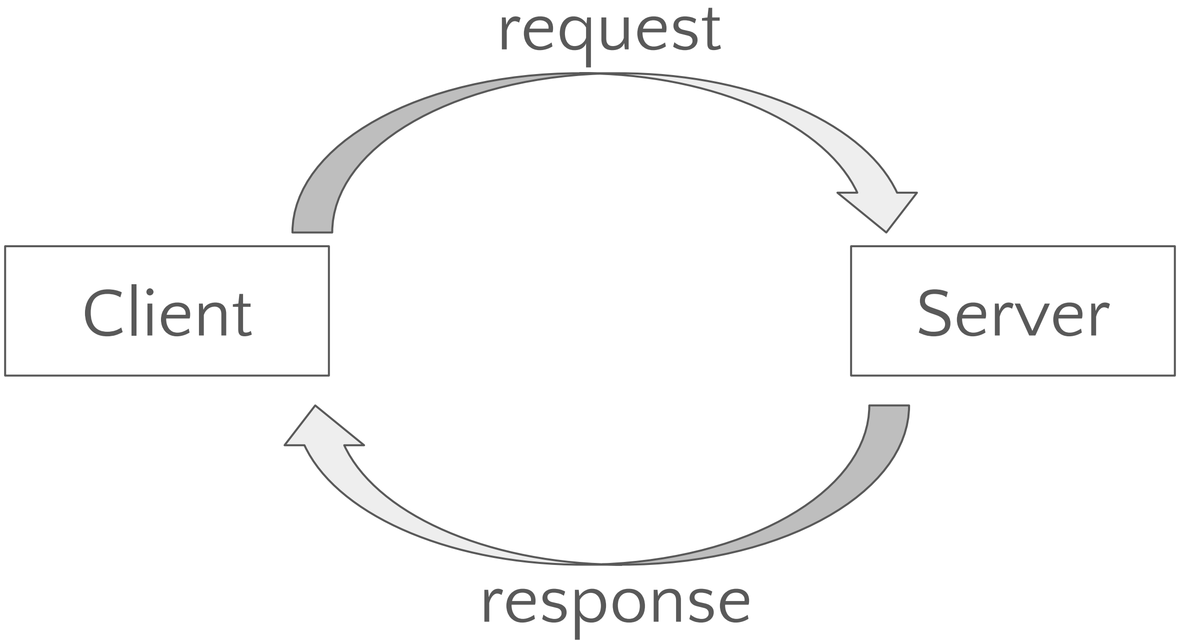
Push the code on to GitHub

Integrate with Travis-CI

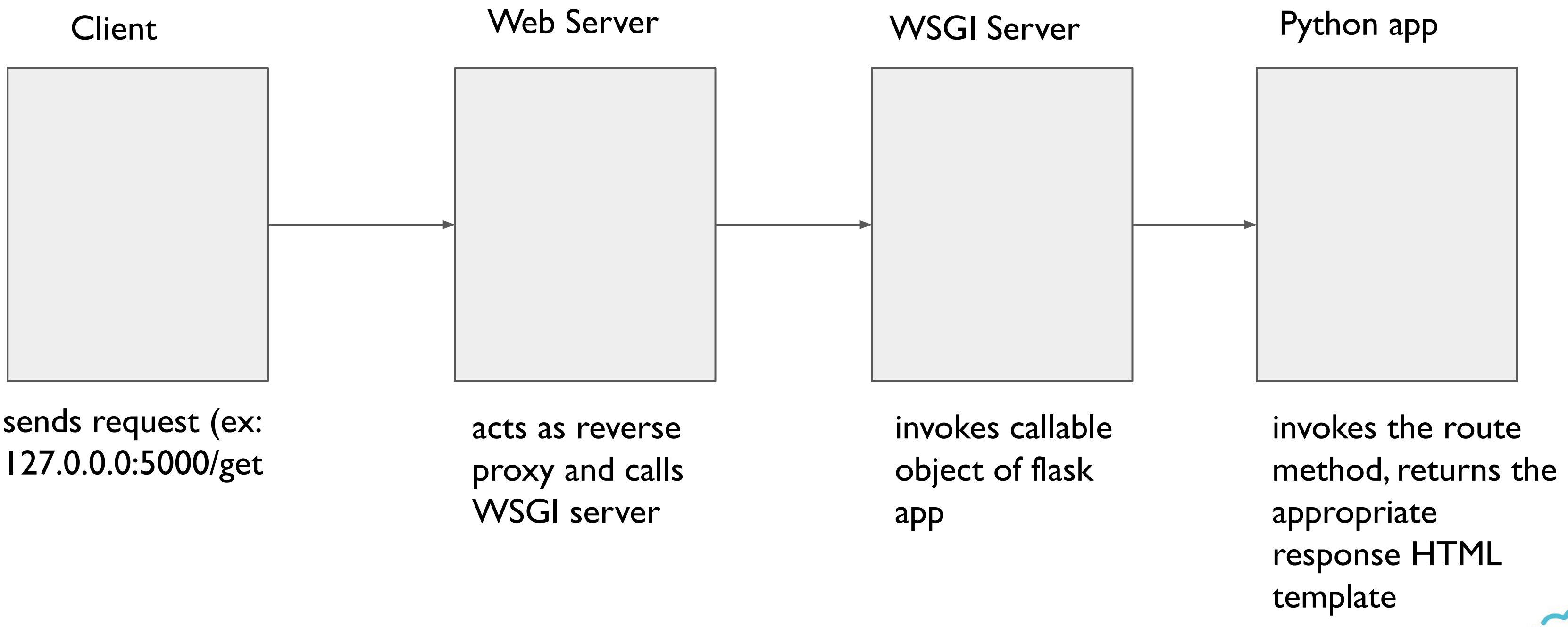
Define the Dockerrun.aws.json file

Host on AWS Elastic Beanstalk

Client-Server Cycle



General Architecture



Client

Web Server

WSGI Server

Python app

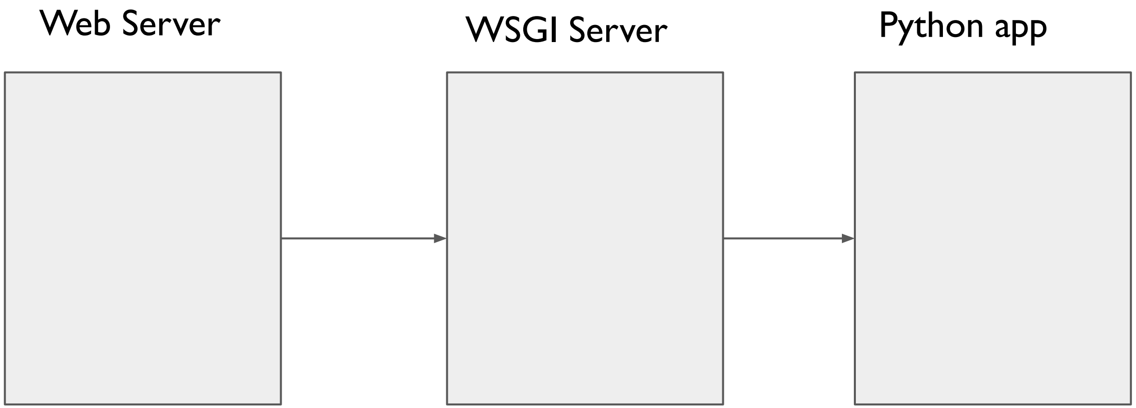
sends request (ex: 127.0.0.0:5000/get

acts as reverse proxy and calls WSGI server

invokes callable object of flask app

invokes the route method, returns the appropriate response HTML template

General Architecture



Web Server

WSGI Server

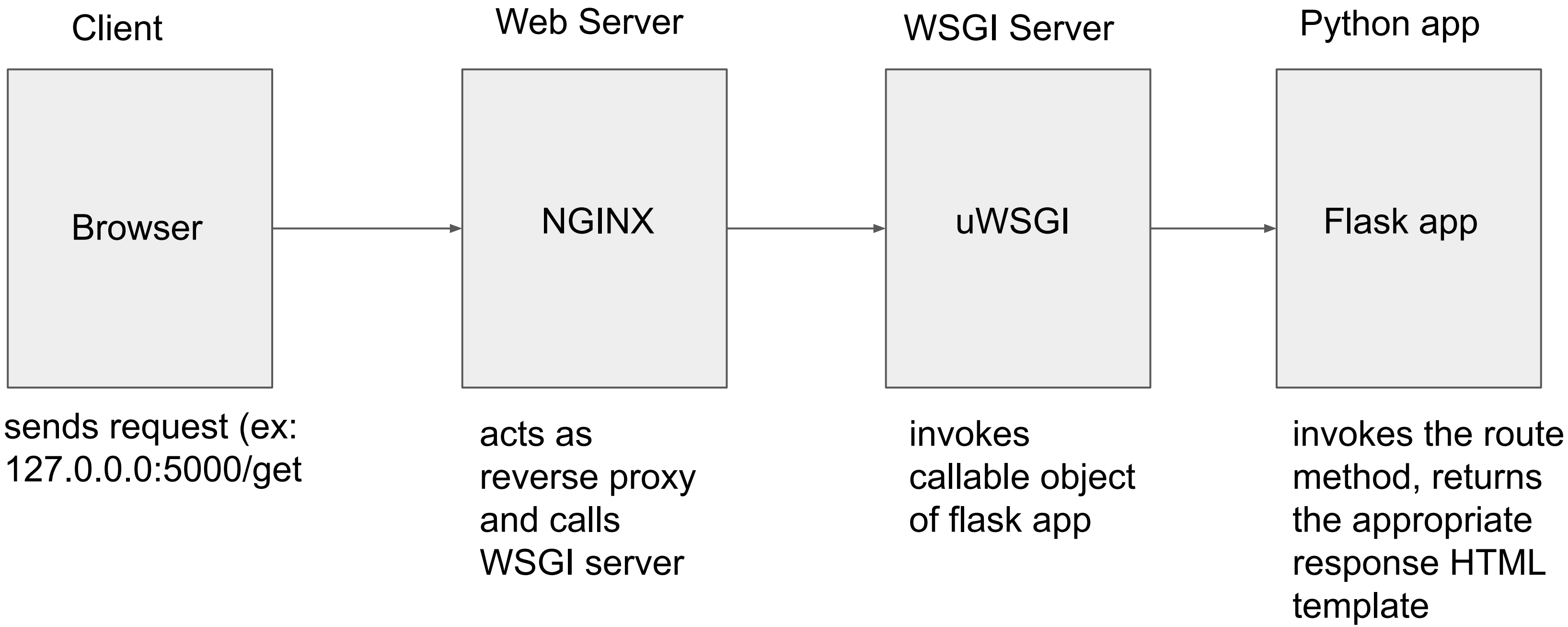
Python app

acts as reverse proxy and calls WSGI server

invokes callable object of flask app

invokes the route method, returns the appropriate response HTML template

General Architecture



Client

Browser

sends request (ex: 127.0.0.0:5000/get

Web Server

NGINX

acts as reverse proxy and calls WSGI server

WSGI Server

uWSGI

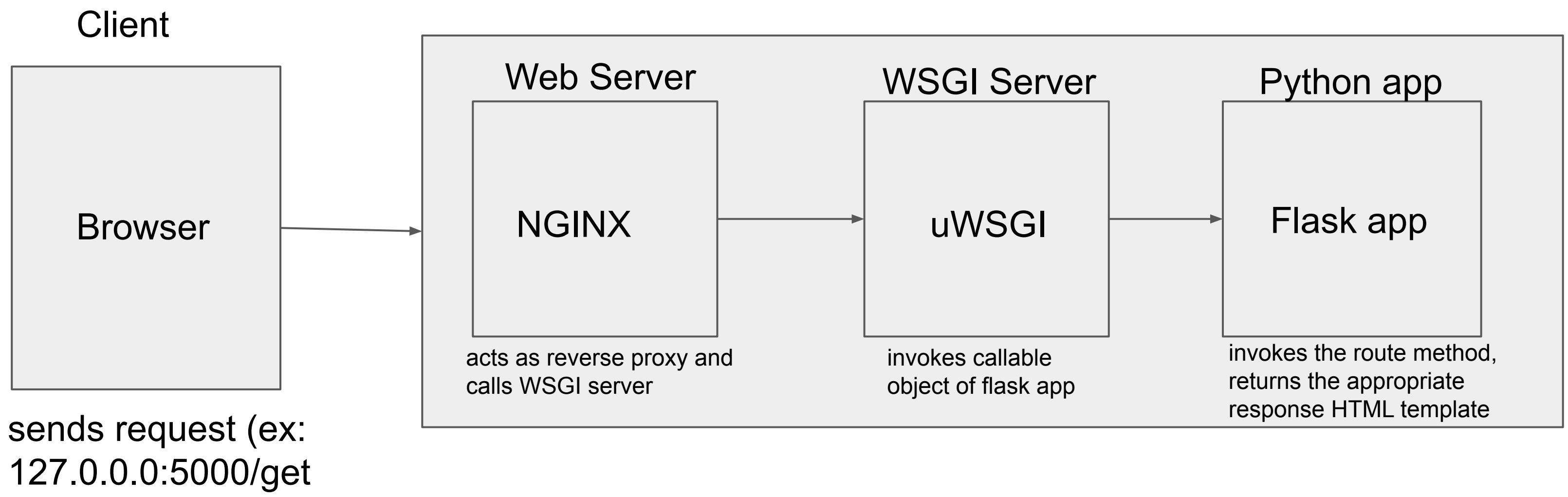
invokes callable object of flask app

Python app

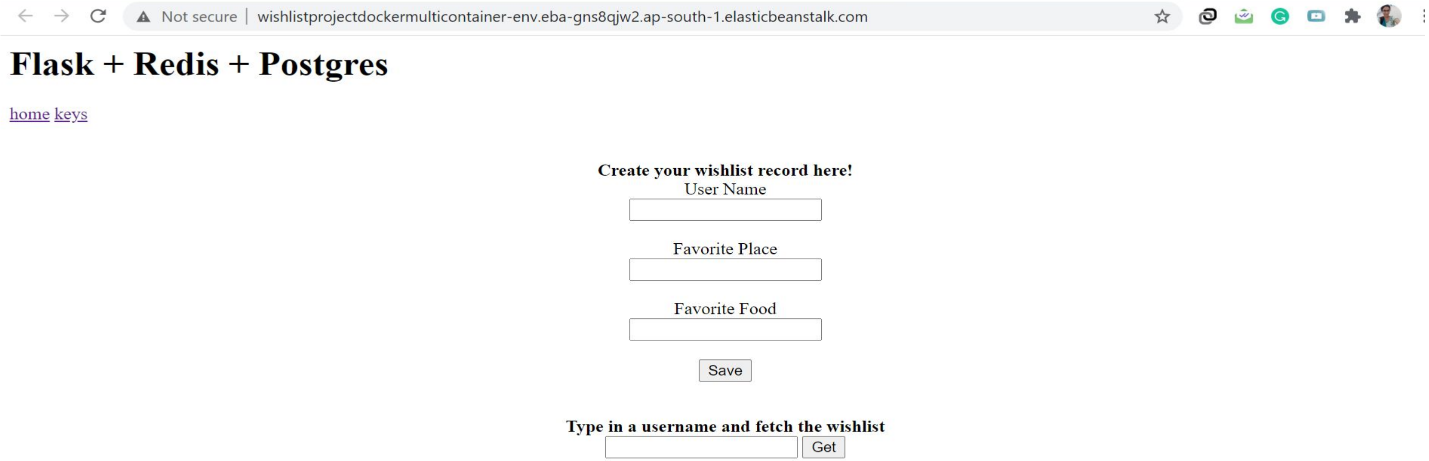
Flask app

invokes the route method, returns the appropriate response HTML template

Architecture in our case



What are we going to do now?



Not secure | wishlistprojectdockermulticontainer-env.eba-gns8qjw2.ap-south-1.elasticbeanstalk.com

Flask + Redis + Postgres

home keys

Create your wishlist record here!

User Name

Favorite Place

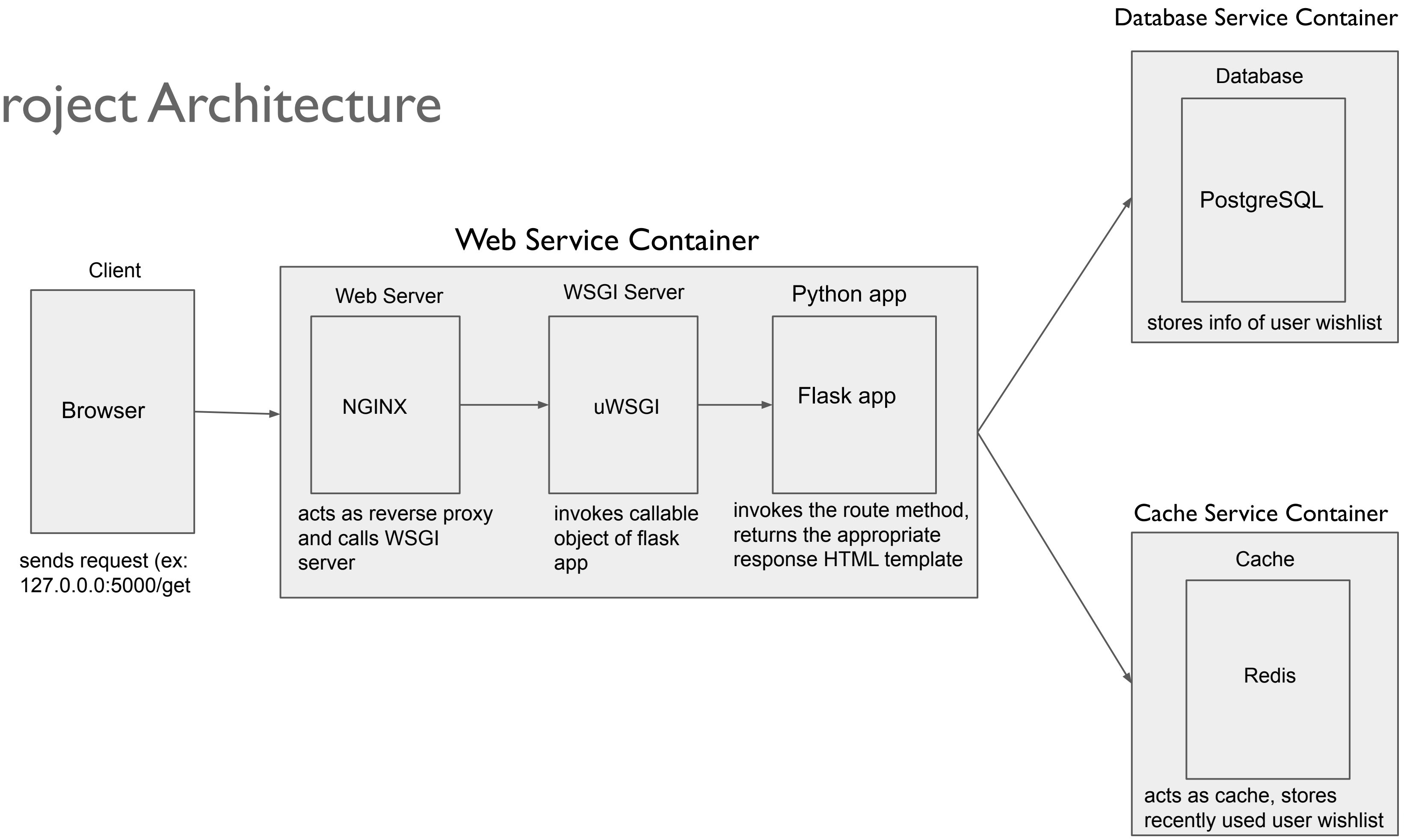
Favorite Food

Type in a username and fetch the wishlist

PostgreSQL: https://www.postgresql.org/

Redis: https://redis.io/

Project Architecture



Client

Browser

sends request (ex:
127.0.0.0:5000/get

Web Service Container

Web Server

NGINX

acts as reverse proxy and calls WSGI server

WSGI Server

uWSGI

invokes callable object of flask app

Python app

Flask app

invokes the route method, returns the appropriate response HTML template

Database Service Container

Database

PostgreSQL

stores info of user wishlist

Cache Service Container

Cache

Redis

acts as cache, stores recently used user wishlist

Demo on building flask app, configuring docker-compose and writing docker file

Getting a basic Flask app running with Nginx-uwsgi

- Let us first focus on building a basic Flask app running with Nginx-uwsgi
- Visit https://hub.docker.com/r/tiangolo/uwsgi-nginx-flask/ for the nginx-uwsgi image for flask apps.

Redis

- A NoSQL cache
- Stores data structures in-memory(ie in RAM)
- Redis keys are always strings
- Redis is a key value store and supports several data structures
- It supports values of many kinds:
    - Strings
    - Sets
    - Lists
    - Hashes, etc.

Redis

For example, setting and getting string values

```
# import redis
import redis

# create redis client
red = redis.Redis(host=’localhost’, port=6379)

# Set a string value ‘Apple’ for the key named 'Fruit'
red.set(‘Fruit’,‘Apple’)

# Get the value for the key and print it
print(red.get(‘Fruit’))

# Change the value of they key
red.set(‘Fruit’,‘Mango’)

# print the new value of Fruit
print(red.get(‘Fruit’))
```

Redis

- Similarly, in our case, we shall use hashes as the values for each username.
- For example:

```
# import redis
import redis

# create redis client
red = redis.Redis(host=’localhost’, port=6379)

#  Add key value pairs to the Redis hash
red.hset(“Alice”, “place”, “Australia”)
red.hset(“Alice”, “food”, “Pasta”)

# Retrieve the value for a specific key
print(red.hget(“Alice”, “place”))
print(red.hget(“Alice”, “food”))
```

Databases with Flask

- RDBMS servers store data as tables
- We query it through SQL
- We use programming languages, which are object oriented, like Python to build web applications.
- So we need to a way to bridge the gap between the SQL programming and Object Oriented Programming
- Object Relational Mapping is something which fills this gap

Object-Relational Mapping

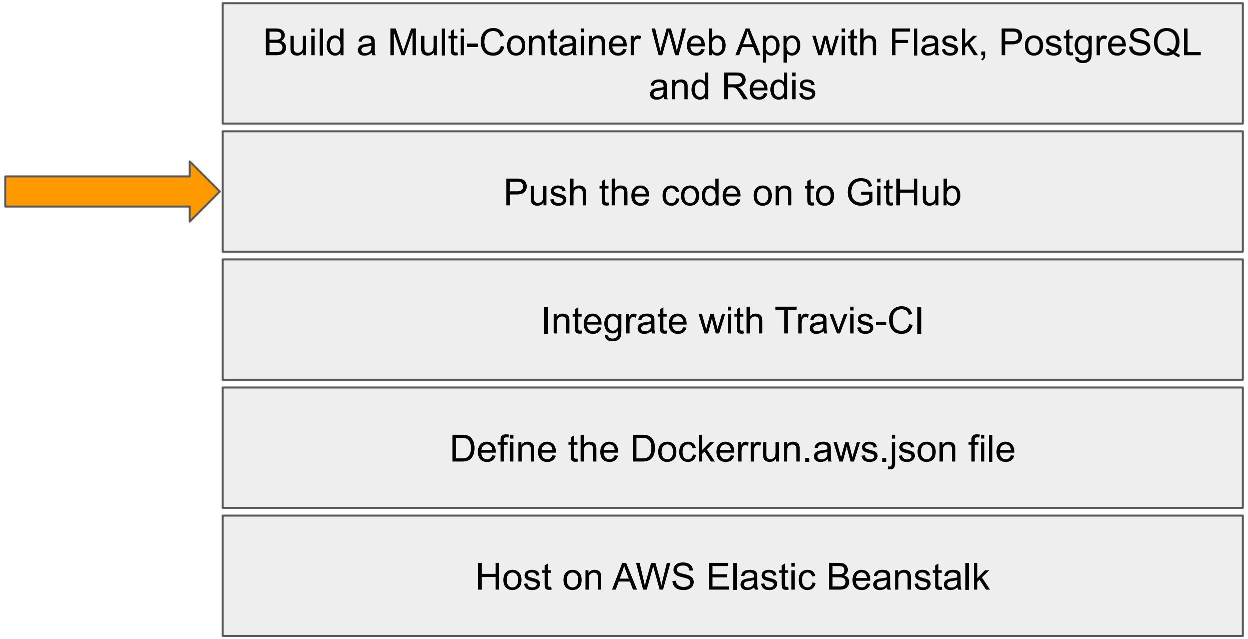
- Object-Relational Mapping provides the flexibility to create/use underlying tables by programming in object-oriented style instead of using SQL

- Using ORM API,
  - We can define database tables using classes
  - We can perform CRUD operations using Object oriented programming

SQLAlchemy

- SQLAlchemy is a Python toolkit and Object Relational Mapper
- Flask-SQLAlchemy is an extension for Flask that adds support for SQLAlchemy to your application.

Multi-Container App on AWS



Build a Multi-Container Web App with Flask, PostgreSQL and Redis

Push the code on to GitHub

Integrate with Travis-CI

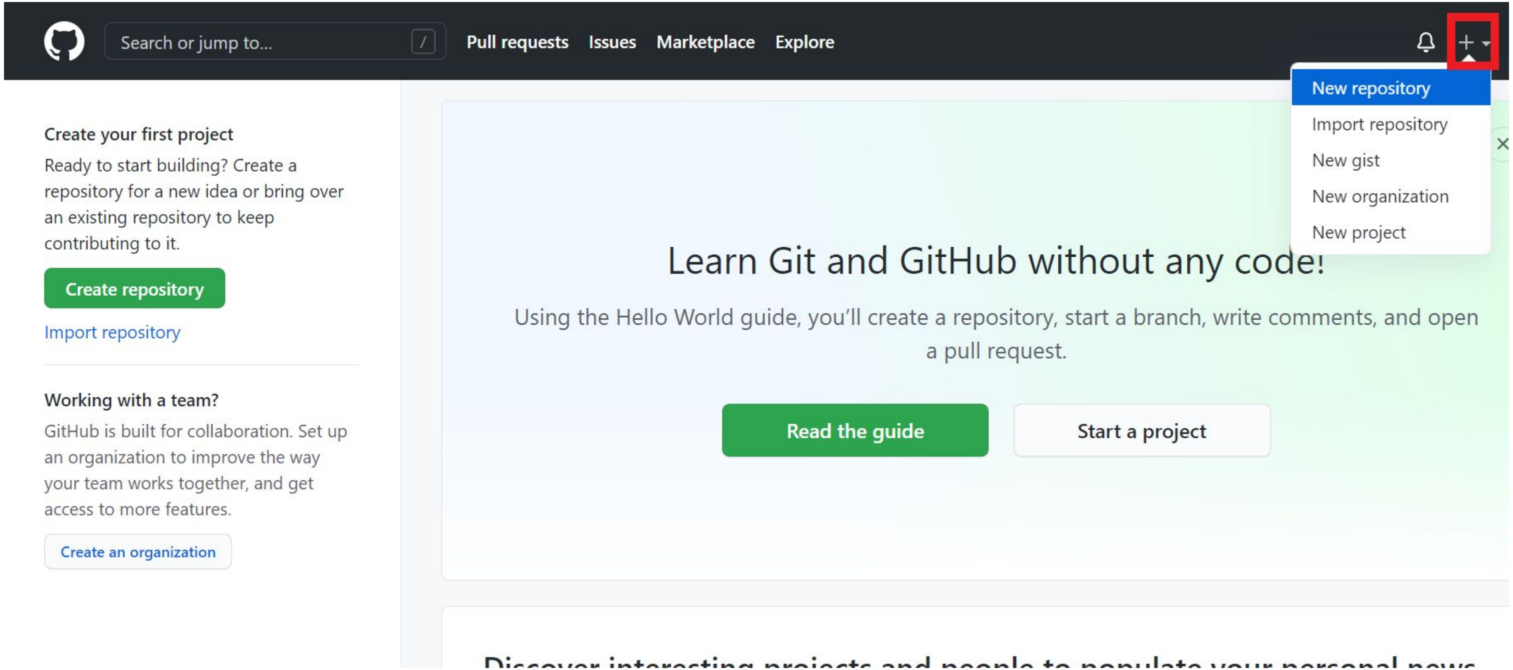
Define the Dockerrun.aws.json file

Host on AWS Elastic Beanstalk

GitHub

- GitHub is a code hosting platform
- For version control and collaboration.
- It lets you and others work together on projects from anywhere.

Create a new repository



Create your first project

Ready to start building? Create a repository for a new idea or bring over an existing repository to keep contributing to it.

Working with a team?

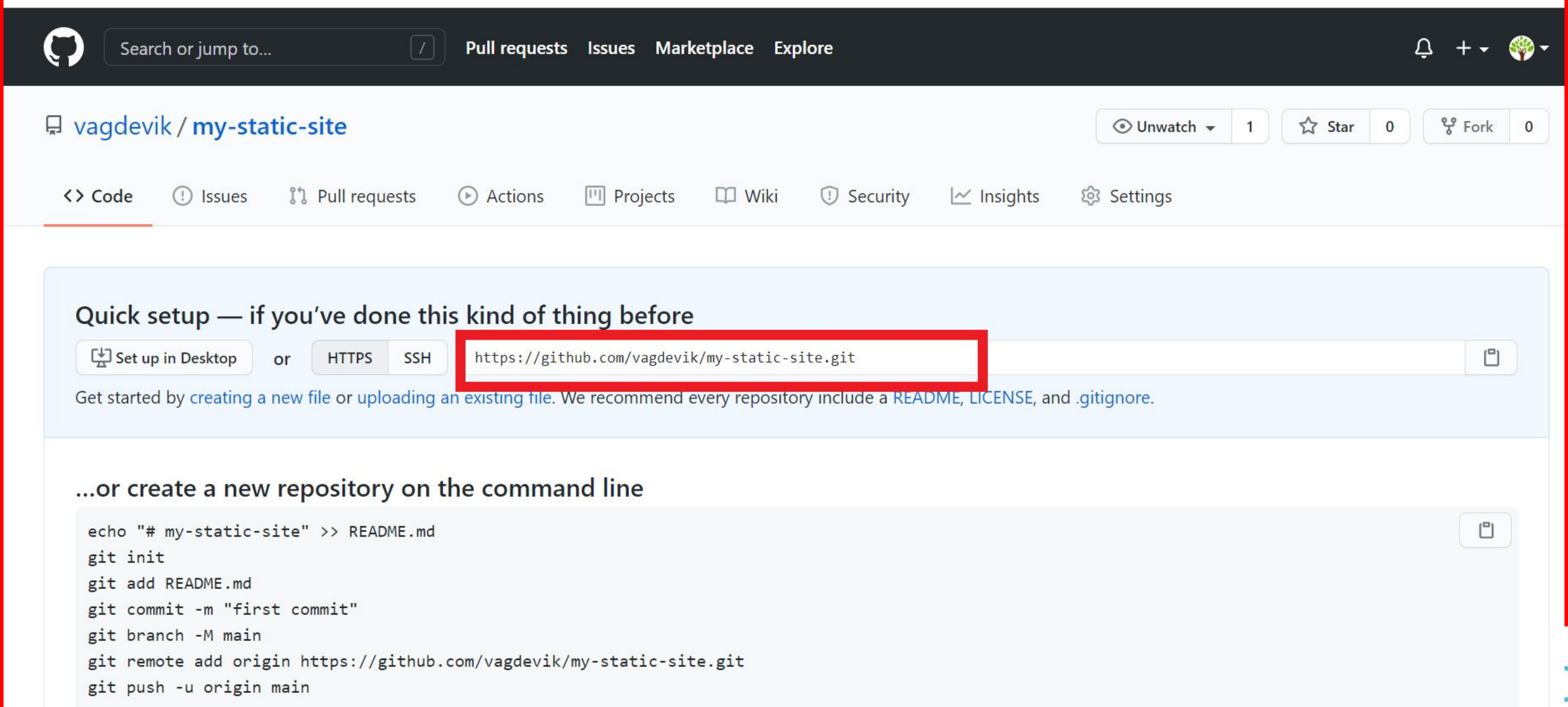
GitHub is built for collaboration. Set up an organization to improve the way your team works together, and get access to more features.

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Discover interesting projects and people to populate your personal news

Push code to the GitHub repo



Quick setup — if you've done this kind of thing before

https://github.com/vagdevik/my-static-site.git

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

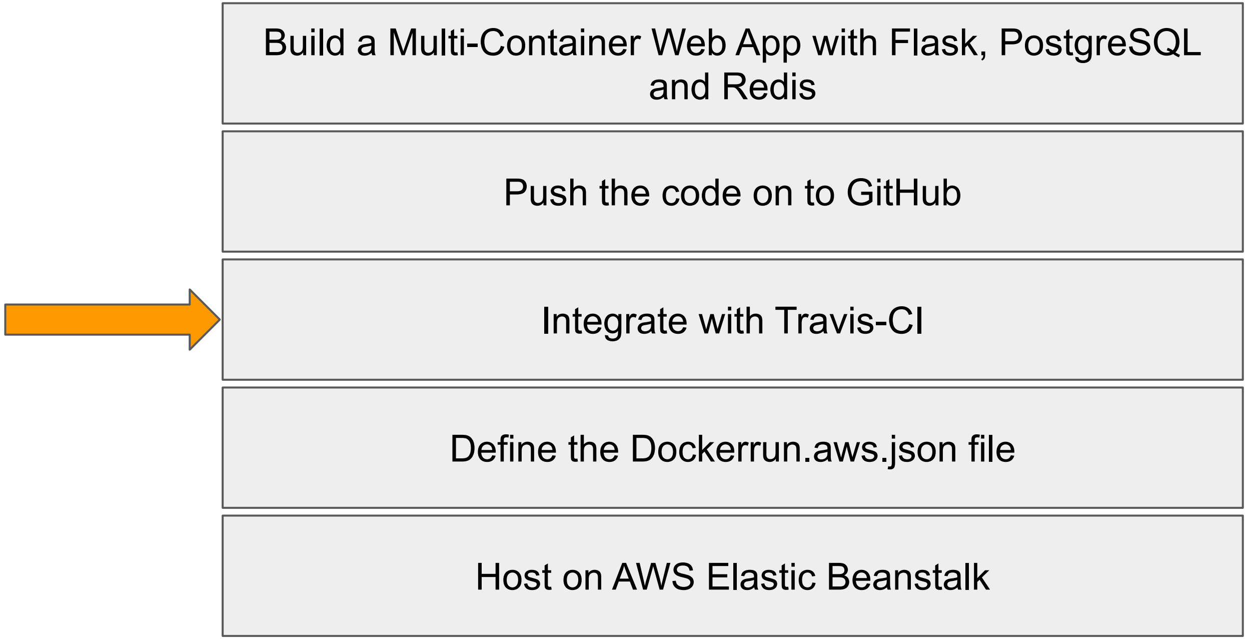
```
echo "# my-static-site" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/vagdevik/my-static-site.git
git push -u origin main
```

Push code to the GitHub repo

Traverse to the project directory and execute the commands:

```
● git init
● git add .
● git commit -m “added dockerized app”
● git push origin master
```

Multi-Container App on AWS



Build a Multi-Container Web App with Flask, PostgreSQL and Redis

Push the code on to GitHub

Integrate with Travis-CI

Define the Dockerrun.aws.json file

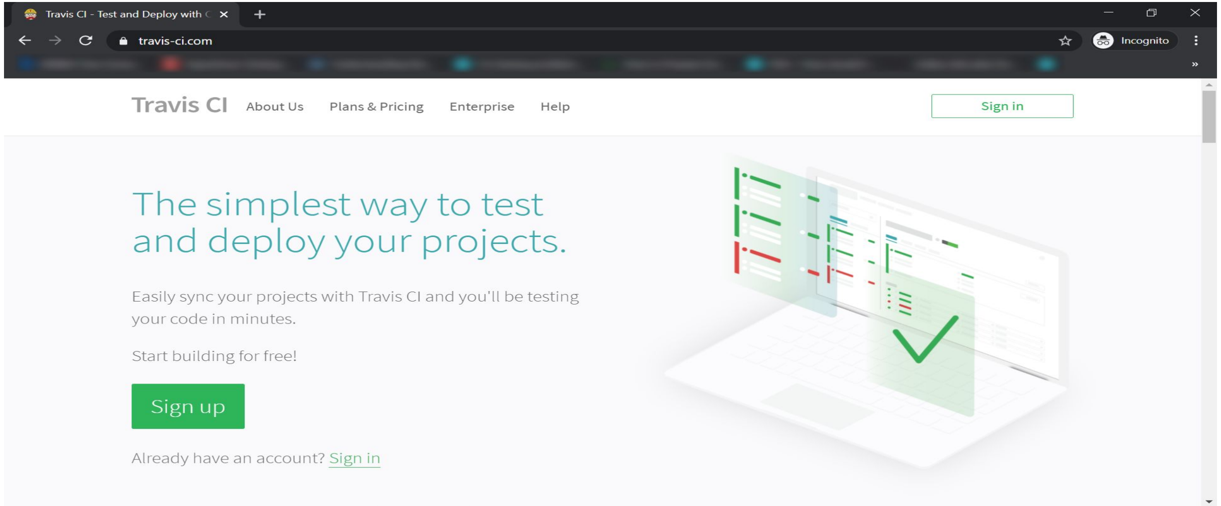
Host on AWS Elastic Beanstalk

Travis CI

- Continuous-Integration tool
- As soon as changes detected:
  - Runs some tests on the new code
  - If tests were successful, integrates those changes on the hosting platforms
- Thus it removes the manual integration step; new features get automatically deployed into production if the code passed test cases of Travis.

Travis Continuous Integration

- Let’s configure Travis CI
- Signup on Travis CI with your GitHub account
  - https://travis-ci.com/



Travis CI - Test and Deploy with C

travis-ci.com

Travis CI About Us Plans & Pricing Enterprise Help Sign in

The simplest way to test and deploy your projects.

Easily sync your projects with Travis CI and you'll be testing your code in minutes.

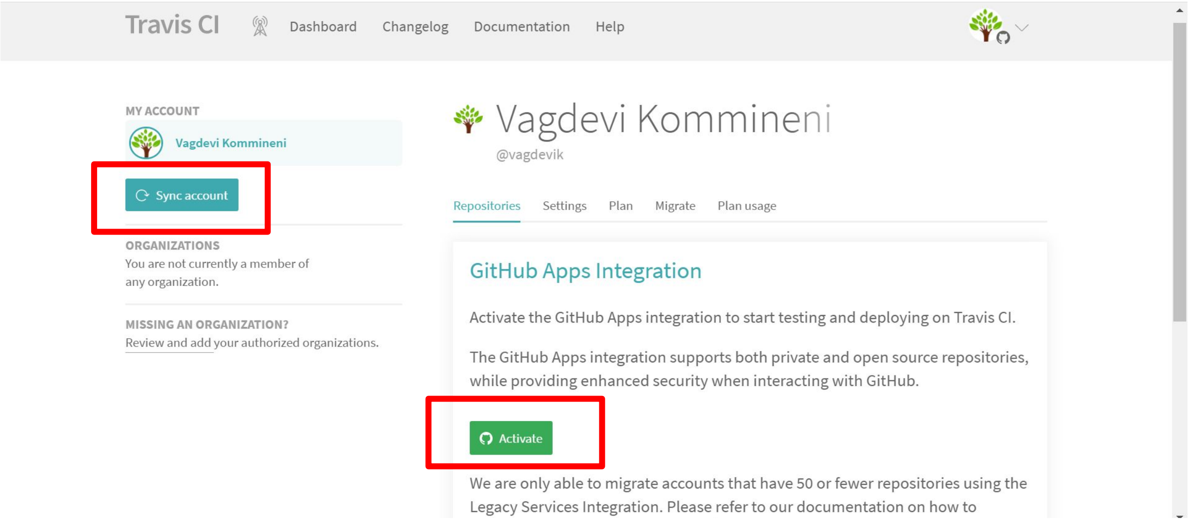
Start building for free!

Sign up

Already have an account? Sign in

Travis Continuous Integration

- Goto https://travis-ci.org/account/repositories
- “Sync Account”



MY ACCOUNT

Vagdevi Kommineni

Sync account

ORGANIZATIONS

You are not currently a member of any organization.

MISSING AN ORGANIZATION?

Review and add your authorized organizations.

Vagdevi Kommineni

@vagdevik

GitHub Apps Integration

Activate the GitHub Apps integration to start testing and deploying on Travis CI.

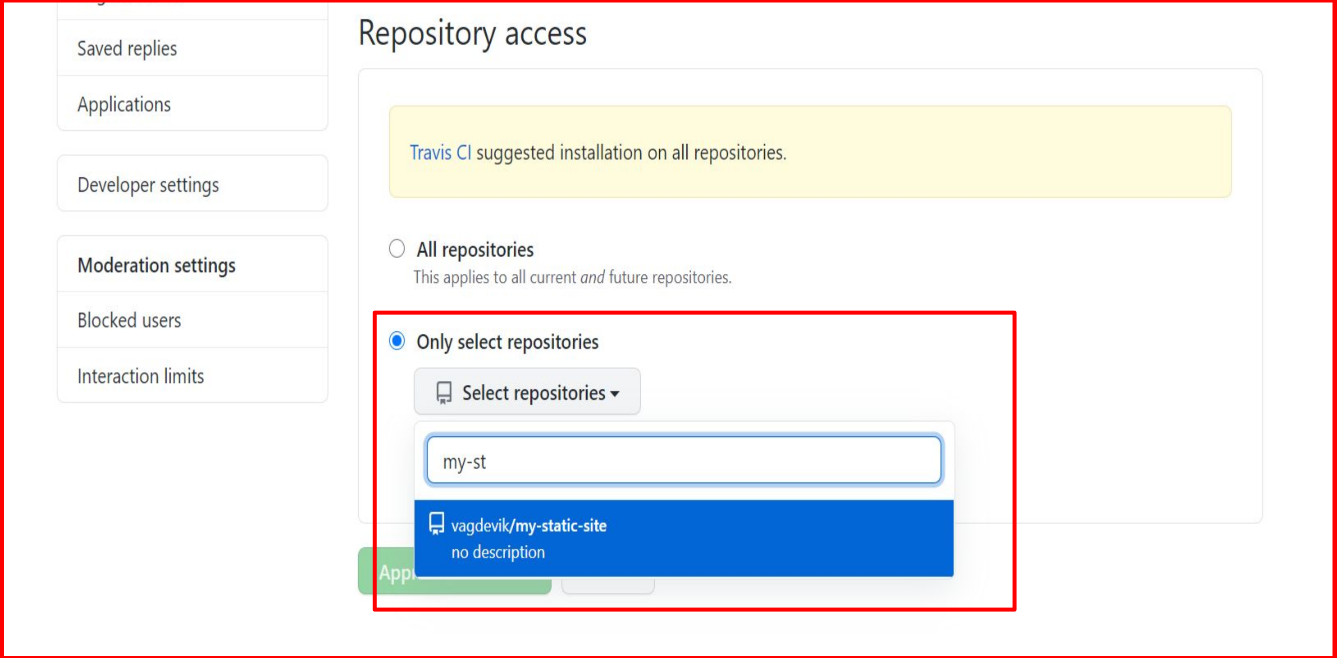
The GitHub Apps integration supports both private and open source repositories, while providing enhanced security when interacting with GitHub.

Activate

We are only able to migrate accounts that have 50 or fewer repositories using the Legacy Services Integration. Please refer to our documentation on how to

Continuous Integration & Deployment with AWS

- Search for your repository, select and Approve it.



Repository access

Travis CI suggested installation on all repositories.

○ All repositories

This applies to all current *and* future repositories.

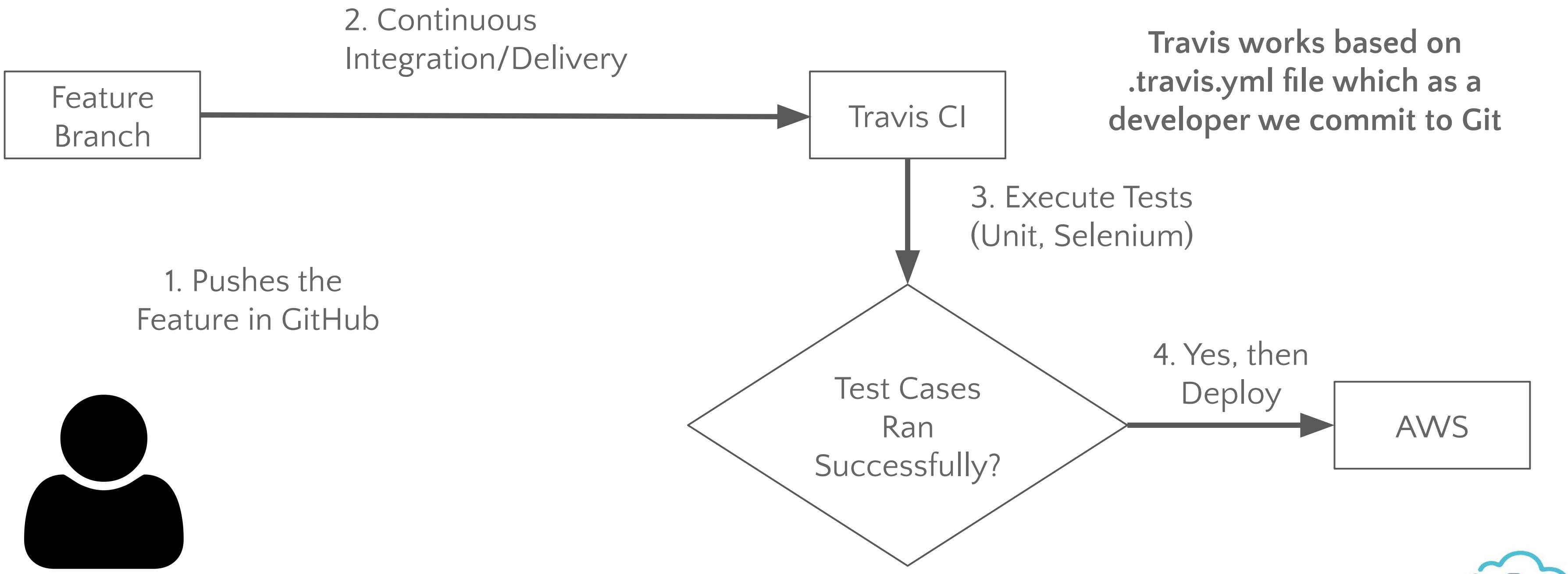
● Only select repositories

my-st

vagdevik/**my-static-site**

no description

Continuous Integration & Deployment with AWS



Feature Branch

2. Continuous Integration/Delivery

Travis CI

3. Execute Tests (Unit, Selenium)

Test Cases Ran Successfully?

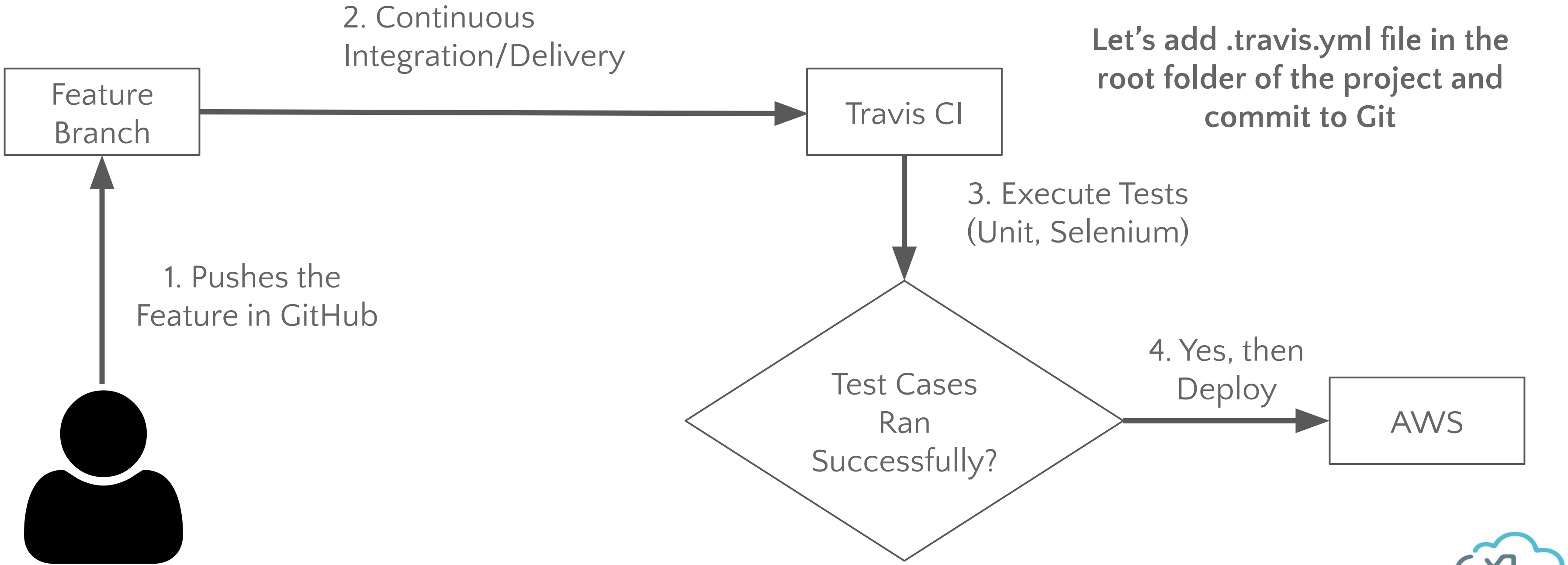
4. Yes, then Deploy

AWS

1. Pushes the Feature in GitHub

Travis works based on .travis.yml file which as a developer we commit to Git

Continuous Integration & Deployment with AWS

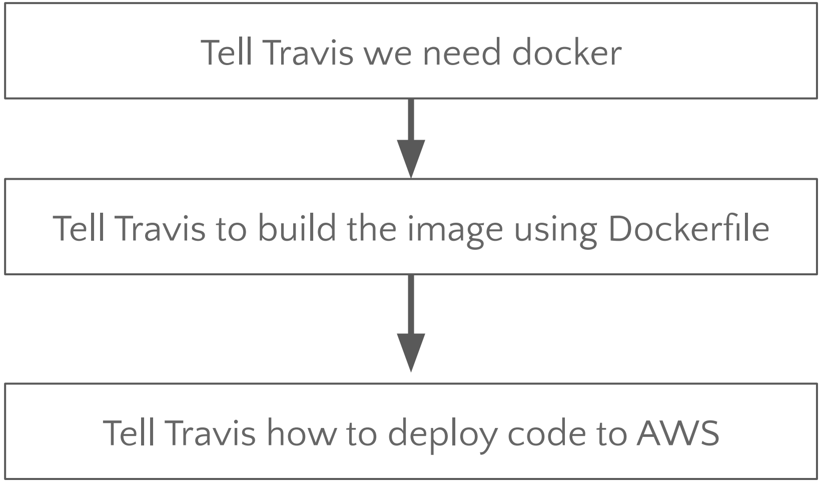


2. Continuous Integration/Delivery
Feature Branch
Travis CI
1. Pushes the Feature in GitHub
3. Execute Tests (Unit, Selenium)
Test Cases Ran Successfully?
4. Yes, then Deploy
AWS

Let’s add .travis.yml file in the root folder of the project and commit to Git

Continuous Integration & Deployment with AWS

- How travis works
    - We specify steps in .travis.yml file



Tell Travis we need docker

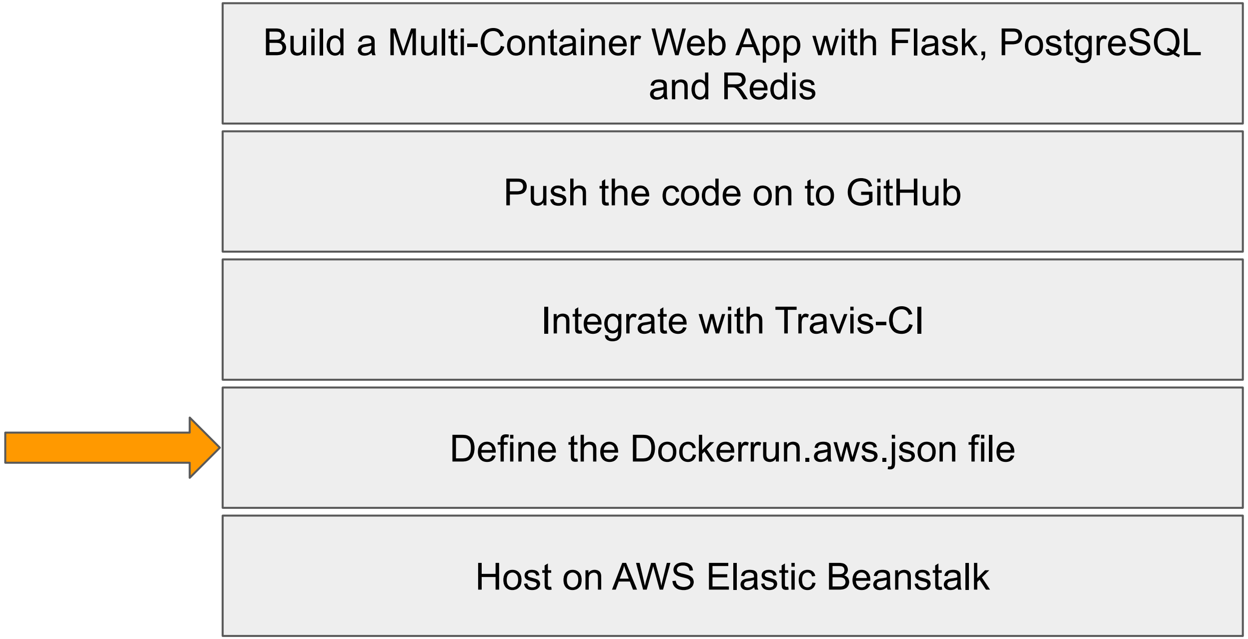
Tell Travis to build the image using Dockerfile

Tell Travis how to deploy code to AWS

Continuous Integration & Deployment with AWS

```
sudo: required

# Tell travis that language is generic
language: generic

# Tell Travis we need docker
services:
    - docker

# Tell Travis to build production images
after_success:
    - docker build -t vagdevik/user-wishlist .

    # Login to docker CLI
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_ID" --password-stdin

    # Push images to docker hub
    - docker push vagdevik/user-wishlist

# Tell Travis how to deploy code to AWS
deploy:
    provider: elasticbeanstalk # We will deploy code to Elastic Beanstalk
    region: ap-south-1 # Specify your region
    app: "wishlist-project-docker-multi-container" # Copy it from Elastic Beanstalk dashboard
    env: "Wishlistprojectdockermulticontainer-env" # Copy it from Elastic Beanstalk dashboard
    access_key_id: $AWS_ACCESS_KEY
    secret_access_key: $AWS_SECRET_KEY
    bucket_name: "elasticbeanstalk-ap-south-1-134650223060" # Elastic Beanstalk will take code from S3 bucket and deploy
it in container. Take it from AWS S3
    bucket_path: "docker" # Folder where Travis CI will upload the code in zip file in this folder. Take it from AWS S3
    on:
        branch: Dockerrun # Deploy only when there are changes on master branch
```

Multi-Container App on AWS



Build a Multi-Container Web App with Flask, PostgreSQL and Redis

Push the code on to GitHub

Integrate with Travis-CI

Define the Dockerrun.aws.json file

Host on AWS Elastic Beanstalk

Introduction to the Dockerrun.aws.json

- docker-compose file is to start and connect all the containers of the app
- This is useful in local
- When deploying on AWS, we define Dockerrun.aws.json
- docker-compose is for local, Dockerrun.aws.json is for AWS
- docker-compose is yml file, Dockerrun.aws.json is json file.
- Simply put, Dockerrun.aws.json is the json form of the docker-compose file.

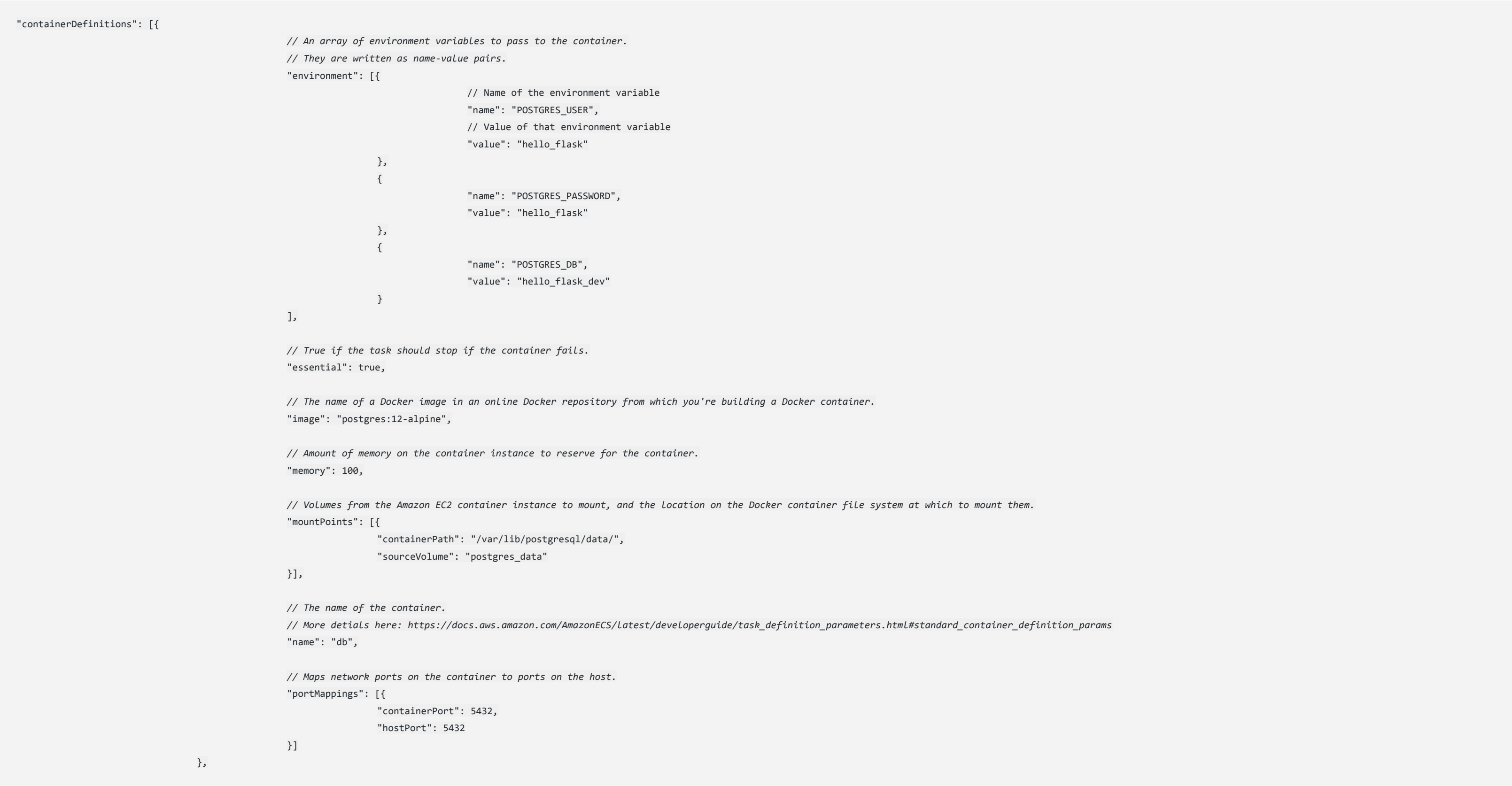
Understanding the Dockerrun.aws.json - **AWSEBDockerrunVersion**

- **AWSEBDockerrunVersion:** Specifies the version number as the value 2 for multi-container Docker environments.

```
"AWSEBDockerrunVersion": 2
```

Understanding the Dockerrun.aws.json - **containerDefinitions**

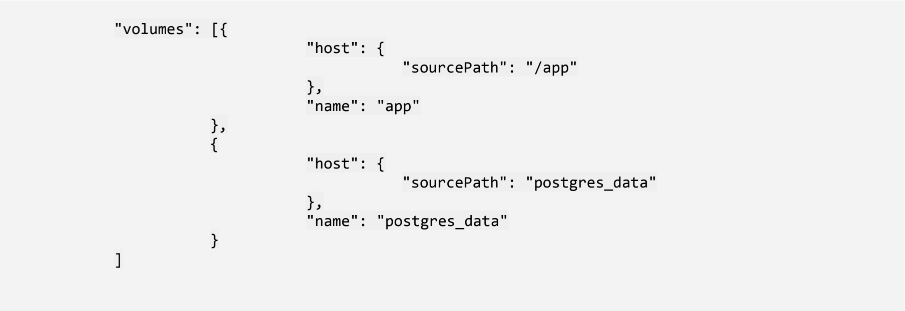
- **containerDefinitions:** An array of container definitions



```
"containerDefinitions": [{
        // An array of environment variables to pass to the container.
        // They are written as name-value pairs.
        "environment": [{
                // Name of the environment variable
                "name": "POSTGRES_USER",
                // Value of that environment variable
                "value": "hello_flask"
            },
            {
                "name": "POSTGRES_PASSWORD",
                "value": "hello_flask"
            },
            {
                "name": "POSTGRES_DB",
                "value": "hello_flask_dev"
            }
        ],

        // True if the task should stop if the container fails.
        "essential": true,

        // The name of a Docker image in an online Docker repository from which you're building a Docker container.
        "image": "postgres:12-alpine",

        // Amount of memory on the container instance to reserve for the container.
        "memory": 100,

        // Volumes from the Amazon EC2 container instance to mount, and the location on the Docker container file system at which to mount them.
        "mountPoints": [{
            "containerPath": "/var/lib/postgresql/data/",
            "sourceVolume": "postgres_data"
        }],

        // The name of the container.
        // More detials here: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html#standard_container_definition_params
        "name": "db",

        // Maps network ports on the container to ports on the host.
        "portMappings": [{
            "containerPort": 5432,
            "hostPort": 5432
        }]
    },
```

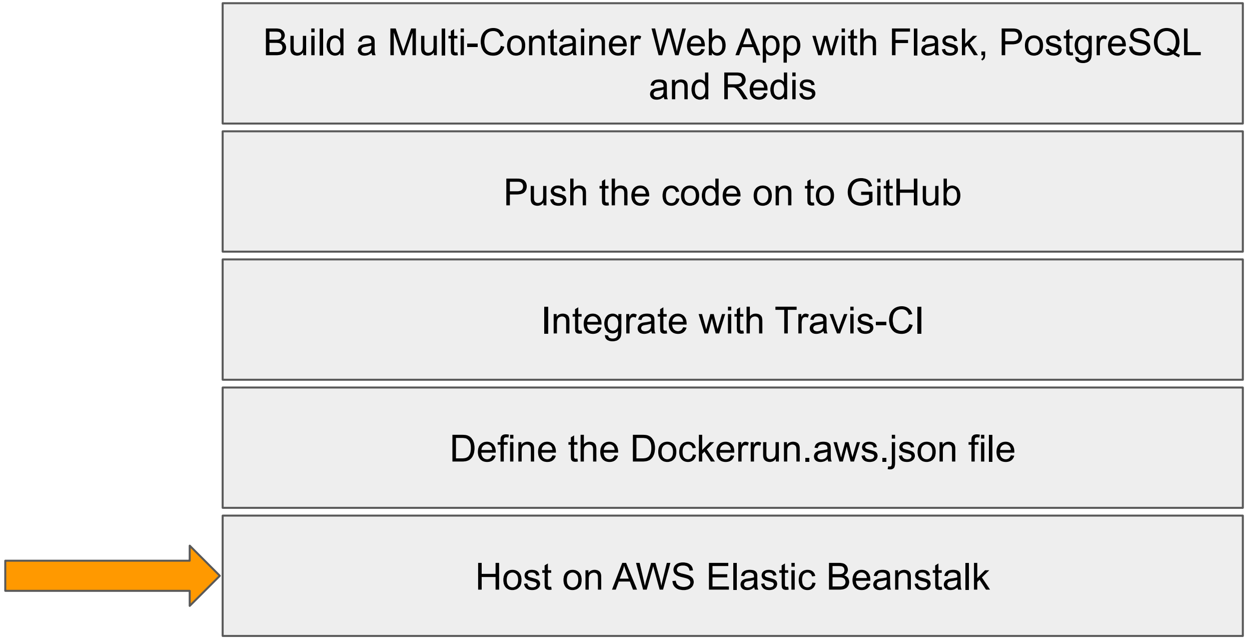
Understanding the Dockerrun.aws.json

- **volumes:** Creates volumes from folders in the Amazon EC2 container instance, or from your source bundle (deployed to /var/app/current). Mount these volumes to paths within your Docker containers using **mountPoints** in the container definition



```
"volumes": [{
                        "host": {
                                    "sourcePath": "/app"
                        },
                        "name": "app"
            },
            {
                        "host": {
                                    "sourcePath": "postgres_data"
                        },
                        "name": "postgres_data"
            }
]
```

Multi-Container App on AWS



Build a Multi-Container Web App with Flask, PostgreSQL and Redis

Push the code on to GitHub

Integrate with Travis-CI

Define the Dockerrun.aws.json file

Host on AWS Elastic Beanstalk