

# Cloud portfolio

## Preface

The application code, project report and pipeline configuration can also be found in the public repository on GitHub<sup>[1]</sup>.

## Introduction

For this assignment we were tasked to create a delivery pipeline using CI/CD to automating testing and deployment of our application code. We were to choose our desired tools and create the pipeline as well as creating a working infrastructure of code to test and build.

## Goals

This report will showcase a possible approach to this and justify the decisions made by creating this type of pipeline as well as the underlying principles it is built on. The goal for this approach is to minimize the costs to stay as a free tier subscription on the cloud provider as well as incorporating relevant technologies to become familiarized. Additionally the intention was to create a Continuous deployment pipeline meaning the whole process of releasing changes into production is entirely automated removing any manual tasks related to this and being able to focus on the application code and its features.

Given that the focus was on developing a pipeline the application itself is relatively simple. It is comprised of a flask-webserver "Hello world!" application. The application and the tests is a modified version of the provided examples from the course textbook (Agarwal, 2021, Building a CI pipeline with GitHub Actions)<sup>[2]</sup>.

## Implementation

### Python & Flask

As stated earlier the application is simple in nature to allow focus to be directed towards the Continuous deployment itself and remove any complexities from the application itself. The application is simply made up of a flask webserver returning a greeting message upon a visit to the hosted site.

```
app.py > {} os
You, 13 hours ago | 1 author (You)
1  from flask import Flask
2  from flask import make_response
3  import os      You, 13 hours ago • Import errors.
4
5  app = Flask(__name__)
6
7
8  @app.route("/")
9  def hello_world():
10     return "This is built with a Continious Deployment pipeline!"
11
12
13  @app.route("/")
14  def default(page):
15     response = make_response("The page %s does not exist." % page, 404)
16
17     return response
18
19
20  # Starts the server on localhost with port 80 as a default
21  if __name__ == "__main__":
22     app.run(host="0.0.0.0", port=int(os.environ.get("WEBSERVER_PORT", 5000)))
23
```

## Docker

This is encapsulated to run on a Docker container which has many advantages when developing an application. The first being that the container itself is CI compliant meaning the unit tests can easily be incorporated during the build process centring the relevant logic in the same place<sup>[2-1]</sup>. Having the application run in a Docker container is also a beneficial solution in the long-term as this can be scaled horizontally by creating multiple instances of this container giving consistency as well as being portable as they can be run on any system capable of running Docker.

The properties of the container is specified in a Dockerfile which makes up the Image that defines how the container will operate and its dependencies .

```

You, 2 seconds ago | 1 author (You)
# Docker file inspired by the following:
You, now * Uncommitted changes
# Agarwal, G. (2021) Modern DevOps Practices. 1st edn. Packt Publishing.
# Available at: https://www.perlego.com/book/2931160/modern-devops-practices-pdf
# https://docs.docker.com/language/python/containerize/

ARG PYTHON_VERSION=3.11.4
FROM python:${PYTHON_VERSION}-slim as base

# Prevents Python from writing pyc files.
ENV PYTHONDONTWRITEBYTECODE=1

# Keeps Python from buffering stdout and stderr to avoid situations where
# the application crashes without emitting any logs due to buffering.
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Create a non-privileged user that the app will run under.
ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser

# Download dependencies as a separate step to take advantage of Docker's caching.
RUN --mount=type=cache,target=/root/.cache/pip \
    --mount=type=bind,source=requirements.txt,target=requirements.txt \
    python -m pip install -r requirements.txt

# Switch to the non-privileged user to run the application.
USER appuser

# Copies everything in the project directory to the current directory in the container.
# In this case, everything in the project directory here will be copied into the root folder of the container.
COPY . .

RUN python3 app.test.py
# Exposes the port where the server will listen
EXPOSE 5000

# Run the application.
CMD ["python", "app.py"]

```

*This Dockerfile is a modified version of the one provided in the course textbook<sup>[2-2]</sup>*

As can be seen on line 44 of the provided image, this approach allows for integrating the in this case low-level tests in the outline for the container itself. The tests are basic as there is no application complexity it ensures that the landing page is available and that it will return correct status code upon an attempt to access non existent address.

app.test.py > ...

You, 1 second ago | 1 author (You)

```
1 import unittest
2 from app import app
3
4 """
5 Tests:
6 - HTTP packet recieved is OK status code 200 at landing page
7 - HTTP packet is returned with status code 404 at unavailable site.
8
9 """
```

You, 2 days ago | 1 author (You)

```
10 class AppTestCase(unittest.TestCase):
11     def test_index(self):
12         tester = app.test_client(self)
13         response = tester.get("/", content_type="html/text")
14         self.assertEqual(response.status_code, 200)
15
16     def test_default(self):
17         tester = app.test_client(self)
18         response = tester.get("xyz", content_type="html/text")
19         self.assertEqual(response.status_code, 404)
20
21
22 if __name__ == "__main__":
23     unittest.main()
24
```

*The low-level tests consisting of unit tests*

## Terraform

In order to host this application it needs the sufficient infrastructure to do so. This is where Terraform comes in as it defines the necessary resources that the application requires.

For starters it needs a virtual machine to run the container. Which in this case is a arbitrary Linux machine using the latest LTS version to combat the risk of becoming

outdated as long as possible.

```
GitHub Workflow - YAMEL GitHub Workflow (github-workflow.json) | You, 2 seconds ago | 1 author (You)
# Workflow for building and deploying to GCP
1  name: build-deploy
2
3
4  on:
5    push:
6      branches:
7        - main
8
9  jobs:
10   build-and-deploy:
11     # Arbitrary version that is LTS.
12     runs-on: ubuntu-22.04
13     environment: skytjenester-H23
14
15     steps:
16       - name: Checkout repository
17         uses: actions/checkout@v2
18
19       # Builds and pushes the docker image to the Docker Hub
20       - name: Build and push Docker image
21         run: |
22           docker login -u ${vars.DOCKER_USERNAME} -p ${secrets.DOCKER_ACCESS_TOKEN}
23           docker build . --file Dockerfile --tag ${vars.DOCKER_USERNAME}/${vars.DOCKER_IMAGE_NAME}
24           docker push ${vars.DOCKER_USERNAME}/${vars.DOCKER_IMAGE_NAME}:latest
25
26       # Google cloud CLI
27       - name: Set up Google Cloud SDK
28         uses: google-github-actions/setup-gcloud@v1
29         with:
30           project_id: '${vars.GOOGLE_PROJECT_NAME}'
31
32       # Google cloud authentication
33       - name: Set up Google Cloud Authentication
34         uses: google-github-actions/auth@v2
35         with:
36           credentials_json: '${secrets.GOOGLE_CREDENTIALS}'
37
```

In order for the VM to be able to communicate publicly it needs to be connected to a network that allows for this. Therefore a network with firewalls allowing for both SSH to manually connect to the computer and TCP for listening at port 5000 to host the webserver . This is similar to the approach used in the Assignment 4 only that this makes it so it has eternal communication.

```

54
55 # Network
You, 2 days ago | 1 author (You)
56 resource "google_compute_network" "vpc_network" {
57     auto_create_subnetworks = false
58     name                     = "skytjenester-net"
59 }
60
61 # Subnet for the instances
You, 2 days ago | 1 author (You)
62 resource "google_compute_subnetwork" "default" {
63     name           = "skytjenester-subnet"
64     ip_cidr_range = "10.0.1.0/24"
65     region         = "europe-west1"
66     network        = google_compute_network.vpc_network.id
67 }
68
69 # Firewall configuration to allow ssh
You, 2 days ago | 1 author (You)
70 resource "google_compute_firewall" "ssh" {
71     name = "allow-ssh"
You, 2 days ago | 1 author (You)
72     allow {
73         ports    = ["22"]
74         protocol = "tcp"
75     }
76     direction     = "INGRESS"
77     network        = google_compute_network.vpc_network.id
78     priority       = 1000
79     source_ranges = ["0.0.0.0/0"]
80     target_tags   = ["ssh"]
81 }
82
83 # Firewall config to allow tcp
You, 2 days ago | 1 author (You)
84 resource "google_compute_firewall" "flask" {
85     name     = "flask-app-firewall"
86     network = google_compute_network.vpc_network.id
87
88     # Firewall rule name
89     target_tags = ["flask"]
You, 2 days ago | 1 author (You)
90     allow {
91         protocol = "tcp"
92         ports    = ["5000"]
93     }
94     source_ranges = ["0.0.0.0/0"]
95 }
96

```

*The tags specified are applied to the VM in the same script*

## GitHub Actions

All these components make up the pipeline which is orchestrated by the GitHub Actions which is specified in the workflow file. As GitHub actions only assigns a temporary

machine which is not necessarily the same each time, creating an environment for the workflow is required to be able to execute the tasks independent of local environment.

## Environments / Configure skytjenester-H23

















### Deployment branches and tags

Limit which branches and tags can deploy to this environment based on rules or naming patterns.

No restriction ▾

### Environment secrets
















Secrets are encrypted environment variables. They are accessible only by GitHub Actions in the context of this environment.

 DOCKER_ACCESS_TOKEN	Updated last week		
 GOOGLE_CREDENTIALS	Updated 2 days ago		
 SSH_PASSPHRASE	Updated yesterday		
 SSH_PRIVATE_KEY	Updated 20 hours ago		
 TF_BACKEND_BUCKET	Updated yesterday		
 Add secret			

*Contains secret information such as authentication details etc. that needs to be encoded (Terraform bucket does not strictly speaking need to be secret)*

## Environment variables

Variables are used for non-sensitive configuration data. They are accessible only by GitHub Actions in the context of this environment. They are accessible using the [vars context](#).

<b>DOCKER_CONTAINER_NAME</b> flask-webserver	Updated 20 hours ago	 
<b>DOCKER_IMAGE_NAME</b> flask-webserver	Updated 2 days ago	 
<b>DOCKER_USERNAME</b> edvardsn	Updated 6 minutes ago	 
<b>GOOGLE_PROJECT_NAME</b> skytjenester	Updated yesterday	 
<b>GOOGLE_ZONE</b> europe-west1-c	Updated yesterday	 
<b>SSH_PUB_KEY</b> ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC+77CtRtCcFh0uyypvq; ...	Updated 20 hours ago	 
<b>SSH_USERNAME</b> pette	Updated 20 hours ago	 
 Add variable		

*Contains non-sensitive information*

The first step is dedicated to testing and building the application. As mentioned earlier given the application size and complexity the low-level tests in the form of unit tests are conducted when building the image for the container that is to be ran. Assuming the tests are passed the Image will be pushed to Docker Hub with the new application code ready to be pulled.



```

github > workflows > build-deploy.yml > {} jobs > {} build-and-deploy > [] steps > {} 5
GitHub Workflow - YAML GitHub Workflow (github-workflow.json) | You, 8 seconds ago | 1 author (You)
1 # Workflow for building and deploying to GCP
2 name: build-deploy
3
4 on:
5   push:
6     branches:
7       - main
8
9 jobs:
10  build-and-deploy:
11    runs-on: ubuntu-latest
12    environment: skytjenester-H23
13
14    steps:
15      - name: Checkout repository
16        uses: actions/checkout@v2
17
18      # Builds and pushes the docker image to the Docker Hub
19      - name: Build and push Docker image
20        run: |
21          docker login -u ${vars.DOCKER_USERNAME} -p ${secrets.DOCKER_ACCESS_TOKEN}
22          docker build . --file Dockerfile --tag ${vars.DOCKER_USERNAME}/${vars.DOCKER_IMAGE_NAME}
23          docker push ${vars.DOCKER_USERNAME}/${vars.DOCKER_IMAGE_NAME}:latest
24
25      # Google cloud CLI
26      - name: Set up Google Cloud SDK
27        uses: google-github-actions/setup-gcloud@v1
28        with:
29          project_id: '${vars.GOOGLE_PROJECT_NAME}'
30
31      # Google cloud authentication
32      - name: Set up Google Cloud Authentication
33        uses: google-github-actions/auth@v2
34        with:
35          credentials_json: '${secrets.GOOGLE_CREDENTIALS}'
36

```

Additionally it extracts the necessary authentication for executing GCP commands for administering resources. Which for this approach is a JSON file manually saved as a secret.

## Terraform

```

38 # Terraform
39 - name: Set up Terraform
40   uses: hashicorp/setup-terraform@v1
41
42 # Initialize Terraform
43 - name: Terraform Initialize
44   run: terraform init -input=false -backend-config="bucket=${secrets.TF_BACKEND_BUCKET}"
45
46 # Plan resources to create test
47 - name: Terraform Plan
48   env:
49     GOOGLE_CREDENTIALS: ${secrets.GOOGLE_CREDENTIALS}
50   run: |
51     terraform plan -var "SSH_PUB_KEY=${vars.SSH_PUB_KEY}" -var "ZONE=${vars.GOOGLE_ZONE}" -input=false -state="gs://${secrets.TF_BACKEND_BUCKET}/d
52
53 # Creates or updates the resources given the current state
54 - name: Terraform Apply
55   env:
56     GOOGLE_CLOUD_KEYFILE_JSON: ${secrets.GOOGLE_CREDENTIALS}
57   run: |
58     terraform apply -auto-approve -input=false -var "SSH_PUB_KEY=${vars.SSH_PUB_KEY}" -var "ZONE=${vars.GOOGLE_ZONE}" -state="gs://${TF_BACKEND_BUCK
59
60 # Returns the ID from the plan, it is optional, this is done after the terraform process

```

From there Terraform will check if the specified resources or rather the state of the resources are inline with the specified resources. As it will be a different environment each time the workflow is executed the state of the resources has to be stored remotely in a "Bucket" to ensure that Terraform can be run idempotent. The bucket is for this approach manually created and stored on GCP using its Cloud Storage API.

```

main.tf > resource google_compute_instance "skytjenester_vm"
    You, 5 seconds ago | 1 author (You)
1 | # Backend configuration used for resource state
    You, 21 hours ago | 1 author (You)
2 | terraform {
    You, 21 hours ago | 1 author (You)
3 |     backend "gcs" {
4 |         bucket = "skytjenester-bucket"
5 |     }
6 | }
7 |
8 | # The public SSH key to assign to the VM
    You, 2 days ago | 1 author (You)
9 | variable "SSH_PUB_KEY" {
10 |     type = string
11 | }
12 |
13 | # The zone which the resources should be created in
    You, yesterday | 1 author (You)
14 | variable "ZONE" {
15 |     type = string
16 | }
17 | # Information related to the provider
    You, 2 days ago | 1 author (You)
18 | provider "google" {
19 |     project = "skytjenester"
20 |     region  = "europe-west1"
21 | }
22 |

```

## SSH

```

59 | # Retrives the IP from the VM as it is ephemeral this is done after the terraform process
60 |
61 | - name: Get the VM IP Address
62 |   id: get-vm-ip
63 |   run: |
64 |     echo gcloud compute instances describe skytjenester-vm --zone ${vars.GOOGLE_ZONE}}
65 |     VM_IP=$(gcloud compute instances describe skytjenester-vm --zone ${vars.GOOGLE_ZONE}} --format='value(networkInterfaces[0].accessConfigs[0].natIP)
66 |     echo "VM_IP=$VM_IP" >> $GITHUB_ENV
67 |     echo $VM_IP
68 |
69 |
70 | # Runs the new application version
71 | - name: SSH and run the new application
72 |   uses: appleboy/ssh-action@v1.0.0
73 |   with:
74 |     host: ${env.VM_IP}}
75 |     username: ${vars.SSH_USERNAME}}
76 |     key: ${ secrets.SSH_PRIVATE_KEY }}
77 |     script: |
78 |       sudo apt update
79 |       sudo apt install -y docker.io
80 |
81 |       sudo systemctl stop docker
82 |       sudo systemctl start docker
83 |       sudo systemctl enable docker
84 |
85 |       sudo docker rm ${vars.DOCKER_CONTAINER_NAME}}
86 |       sudo docker login -u ${secrets.DOCKER_USERNAME}} -p ${vars.DOCKER_ACCESS_TOKEN}}
87 |       sudo docker pull ${secrets.DOCKER_USERNAME}}/${vars.DOCKER_IMAGE_NAME}}:latest
88 |       sudo docker run -d -p 5000:5000 -e WEBSERVER_PORT=5000 --name ${vars.DOCKER_CONTAINER_NAME}} ${secrets.DOCKER_USERNAME}}/${vars.DOCKER_IMAGE
89 |

```

The final step of the workflow is retrieving the IP address from the possibly newly created VM as the external IPs are ephemeral in the free tier of GCP.

From there its simply to use the private SSH key belonging to the previously injected public SSH key to attach to the VM and restart Docker and clean up the previous iteration as well as pulling and starting the new version.

- > ✓ Set up job
- > ✓ Build appleboy/ssh-action@v1.0.0
- > ✓ Checkout repository
- > ✓ Build and push Docker image
- > ✓ Set up Google Cloud SDK
- > ✓ Set up Google Cloud Authentication
- > ✓ Set up Terraform
- > ✓ Terraform Initialize
- > ✓ Terraform Plan
- > ✓ Terraform Apply
- > ✓ Get the VM IP Address and do ssh
- > ✓ SSH and run the new application
- > ✓ Post Set up Google Cloud Authentication
- > ✓ Post Checkout repository
- > ✓ Complete job

*Timeline which showcases the steps taken in the pipeline*

## Showcase

```
> gcloud compute instances list
NAME                ZONE          MACHINE_TYPE  PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP  STATUS
skytjenester-vm     europe-west1-c  f1-micro           10.0.1.4     34.79.1.186  RUNNING
~ took 2s
>
```

← → ↻ ⚠ Ikke sikker | http://34.79.1.186:5000

This is built with a Continuous Deployment pipeline!

# Reflection

There are multiple areas of this approach which can be improved upon such as the deployment. Using SSH to deploy is not an ideal strategy as this can be complex if the number of instances increases. Deploying using Spinmaker and Kubernetes or similar technologies can strive to keep the deployment process more manageable and simplify the process of incorporating more sophisticated deployment processes such as Blue/Green deployment but require more resource which are not optimal when trying to stay within the free tier (Agarwal, 2021, Continuous Deployment and Automation ).

Another point of improvement is the manual configurations used to run this pipeline such as the GCP Bucket which ideally should be replaced with HashiCorp cloud storage or similar methods. Additionally it might be beneficial to incorporate a security vault as the number of necessary secrets could increase when scaled.

Lastly as the application is simplistic there are not much to test. As the complexity of the application increases incorporating both mid-level and high-level tests into the test suite and workflow will prove beneficial.

As a final note the pipeline manages to deploy new application code as well as be idempotent. This automates the manual process of testing, building and deploying new features of the application. This has been a time consuming process of configuring the pipeline but very educational and accomplishes the initial goals for the project.

## References

---

1. <https://github.com/Edvardsn/InfrastructreAsCode>↵
2. Agarwal, G. (2021) Modern DevOps Practices. 1st edn. Packt Publishing. (Accessed: 5 December 2023).↵↵↵