Deployment Agents

In the sixth semester at Fontys, we are creating a group project called Dverse, The main goal of this project is to create a system where different agents can work together. Our individual is creating one or more of those agents that can complete tasks in the group project's system. To get a better understanding of what actually is needed to create such an agent, we created a design challenge research question with multiple sub questions. This research paper will be written to search for the answer to a sub question.

Table of contents

Research

How can we deploy a Dockerized Flask application and to what service?

Context

For the individual project, we are creating agents that needs to be implemented in the Group project. To access these bots, we need to deploy them to a certain service. This becomes an API that can be accessed all over the internet and especially in the group project.

There are many ways to deploy a certain API, but what are the good and bad ways. That is what I am going to find out now.

Library

Available product analysis

To answer this question, I first will ask a question to Perplexity AI, which is an AI that can search the web and summarize the found articles into one complete answer.

- **Prompt:** Methods to deploy a Docker Container
- **Answer:** complete answer

- Google Cloud Run (free tier available)
- Heroku (free tier available)
- Azure App Service (free Linux-only tier available)
- DigitalOcean (affordable Docker hosting options)
- AWS (with pay-as-you-go pricing)

Now I know a bit about the services to where to deploy to like: Azure, AWS and DigitalOcean. I know from school we get free credits on Azure. So in our case, Azure is going to be a good first option in order to use it for free (but Azure overall a good option).

- **Prompt:** Methods to deploy a docker container: Development and Operations (DevOps)
- **Answer:** complete answer
  - In summary, the key methods for deploying Docker containers in a DevOps environment include containerization, image versioning and registry, deployment automation, CI/CD integration, consistency across environments, rollback and versioning, and resource efficiency.

Now we know that the deployment is going to happen thought a Pipeline using CI/CD. I already used it many times before, and already thought it was going to be a good option. For the next question, we want to know: How can we deploy a Docker Container to Azure using the CI/CD?

- **Prompt:** How can we deploy a Docker Container to Azure using the CI/CD?
- **Answer:** complete answer
  - The key steps are as following:
    1. Build and push the Docker image to an Azure Container Registry.
    2. Create an Azure Web App for Containers.
    3. Deploy the Docker container to the Azure Web App using the CI/CD pipeline.
  - Possible tutorials:
    - https://www.youtube.com/watch?v=H5hs4LreRS0
    - https://learn.microsoft.com/en-us/azure/app-service/deploy-ci-cd-custom-container?tabs=acr&pivots=container-linux

So, now I know a bit about where and how we are going to deploy our Dockerized Flask Applications to Azure. In order to make this work. I need to make a prototype to find out how the CI/CD is going to work in Azure.

Workshop

Prototyping

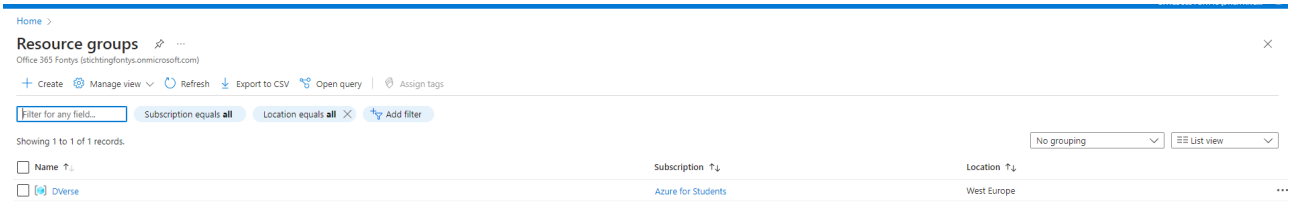
Before starting with prototyping, what is it actually what I want to know? I sort of made the decision to use the Azure as a service to deploy to, and CI/CD as a technique. So now the question remains. How can I set up a CI/CD to push and deploy to Azure?

Azure

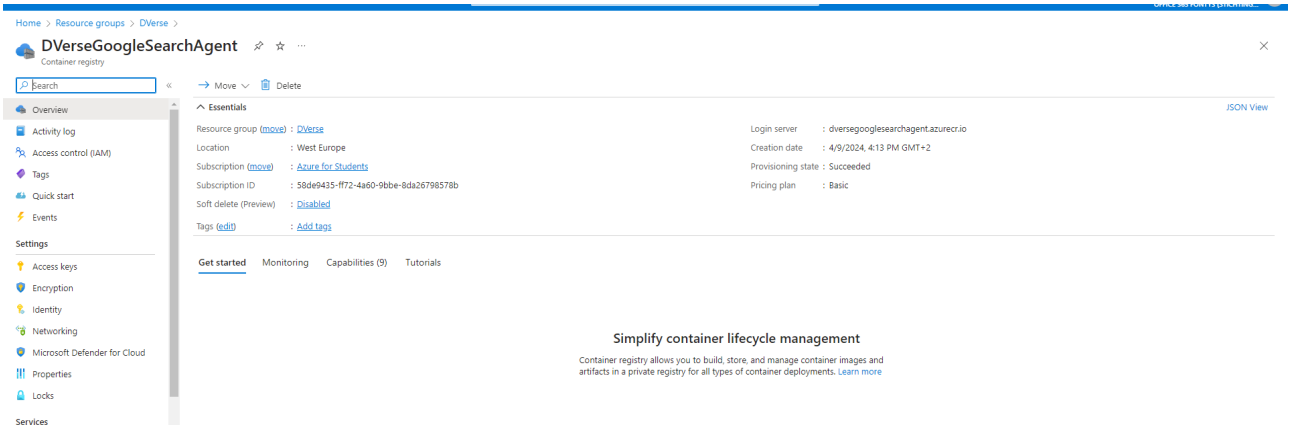
Before creating some code, there are some things I need to do according to this documentation. First, I needed to create a new Resource Group. I called it `Dverse`. In here are going to be all the services that are needed for the project.

After that, I needed an Azure Container Registry, this is going to contain the Docker Container. And at last I need a Azure Web App, this is in the end going to be linked to the Container Registry in order to make it accessible on the internet.

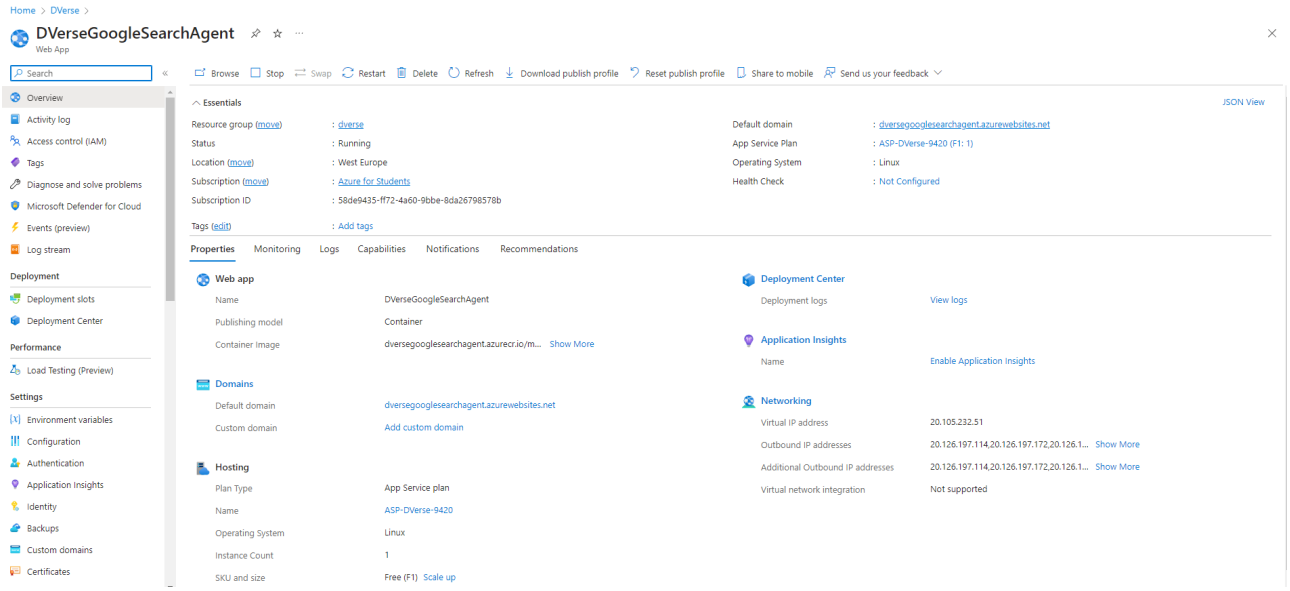
1. Create a new resource group



2. Create a new Container Registry in the DVerse resource group



3. Create a new App Service in the DVerse resource group



So for now this is what I had to do in Azure, the next step is how I can build and push a Docker Container to the Container Registry in Azure.

CI/CD

I already had knowledge about CI/CD, so it is not going to be totally new. How it is going to work out for building and pushing to Azure Container Registry. With a little bit of help from ChatGPT, I created a workflow file in my root directory.

1. In the root of the Github Repository, create a new path: `.github/workflows`. In here are all the Github actions going to be.
2. In here, I created a file called: `PushAndDeployToAzure.GoogleSearchAgent.yml`. This file is going to contain all the required jobs to eventually deploy to Azure.

The workflow is going to need the following steps:

1. Build: Build the docker image
2. Push: Push the docker image to the Container Registry
3. Deploy: Deploy to Web Service from Container Registry

In the end, there are going to be steps involved as well. But for now, I will mainly focus on deployment. We actually need some more things in our workflow like:

- Name: General name of the workflow
- on: This tells Github when to run this workflow, it can be on branch or path and more
- env / args: Here can certain arguments be added when it is used multiple times during the workflow
- And more...

So, now we know what we need. We can set up the first workflow:

1. First we need to have some required and needed steps. Those are as followed.

```
name: Build, Push and Deploy to ACR and Web App

env:
  AZURE_WEBAPP_NAME: DVerseGoogleSearchAgent

on:
  push:
    paths:
      - GoogleSearchAgent/**
      - .github/workflows/PushAndDeployToAzure.GoogleSearchAgent.yml
    branches:
      - main

permissions:
  contents: read
```

As you can see, we start with the `name`. The name how the workflow is called. It is shown on Github and can easily be found by name. The `env` is not required, but is can come in handy, because it tells how the Azure Web App is called, it is used frequently in the file, so this way you can change it at one place and then it changes everywhere.

The `on`, is required, it tells the Github Actions when the workflow should start. In this case, it starts when changed are being made in the GoogleSearchAgent and only on the branch main. At last the `permissions` are being set in order to read certain data.

2. After we put the required steps, we can create the first Jobs.

```
jobs:
  build_and_push:
    runs-on: ubuntu-latest
    steps:
          -
  deploy:
    runs-on: ubuntu-latest
    needs: build_and_push
    steps:
      -
```

As you can see, the `Jobs` are splitted into 2 parts. `build_and_Push`, this is for the Container Registry. And there is a part called `deploy`, it deploys the pushed Docker Container from the

Container Registry to the Web App. In the next step I am going to deep dive into the steps and its actions.

3. In each job are also some steps it takes. Here you can see the build and push job and the steps.

```
build_and_push:
  runs-on: ubuntu-latest
  defaults:
    run:
      shell: bash
      working-directory: GoogleSearchAgent
  steps:
    - uses: actions/checkout@v2

    - uses: azure/docker-login@v1
      with:
        login-server: dversegooglesearchagent.azurecr.io
        username: ${{ secrets.REGISTRY_USERNAME }}
        password: ${{ secrets.REGISTRY_PASSWORD }}

    - name: Build and Push Docker image
      run: |
        docker build -t dversegooglesearchagent.azurecr.io/myapp:${{
github.sha }}
        docker push dversegooglesearchagent.azurecr.io/myapp:${{ github.sha
}}
```

What this code does, it logs in into Azure using some secrets. The `REGISTRY_USERNAME` and `REGISTRY_PASSWORD` needs to be set in Github under `settings` -> `secrets and variables` -> `actions` -> `repository secrets`. Defining those secrets in here, gives Github the ability to use those variables but keep them secret and being able to log in.

After logging in, the step: `Build and Push Docker image` is first building the Docker Container and after then it pushes the Container to the Container Registry in Azure.

4. The last is deploying the Container to the Azure Web Service in order to be able to visit the use the API.

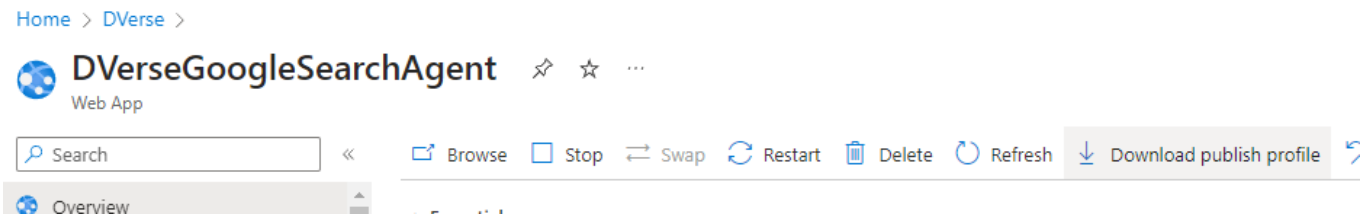
```
deploy:
  permissions:
    contents: none
  runs-on: ubuntu-latest
  needs: build_and_push
```

```
  steps:
    - name: Lowercase the repo name and username
      run: echo "REPO=${GITHUB_REPOSITORY,,}" >>${GITHUB_ENV}

    - name: Deploy to Azure Web App
      id: deploy-to-webapp
      uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ env.AZURE_WEBAPP_NAME }}
        publish-profile: ${{
secrets.GOOGLE_SEARCH_AGENT_AZURE_PUBLISH_PROFILE }}
        images: 'dversegooglesearchagent.azurecr.io/myapp:${{ github.sha }}'
```

This step focuses on the deployment part of the Container to the Azure Web Service. This can be done manually in the Azure Portal, every time a new version of the Container Registry is being pushed, the new version is not actually being used in the Web Service. So that is where this script comes in.

`Deploy the Azure Web App` is used to actually deploy the Container. It requires the `AZURE_WEBAPP_NAME` that is defined on top of the file. Than Azure want to have the publish profile that can be installed from the Azure dashboard from the App Service.



And at last you have to specify the image name that is on the Container Registry. This is the same as used for building and pushing the Docker Container,