Setup Pipeline Docker and Azure

Table of contents

Introduction

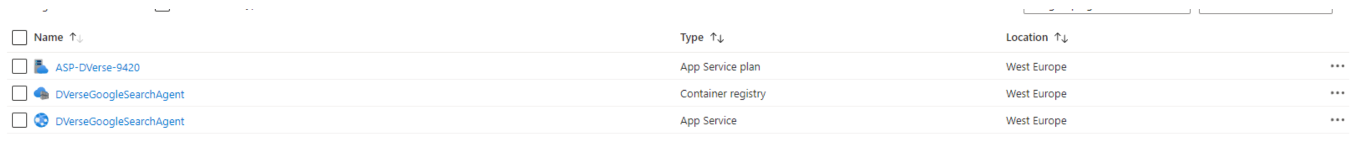
This document is going to help you with setting up CI/CD for building, pushing and deploying a Docker Container to Azure. This tutorial is based on the GitHub pipeline, so if you want to use Gitlab. You have to check their documentation in order to use their Pipeline.

Requirements

Before you can start creating the Pipeline, you need to have some things configured in Azure. We need to have some resources created in order to deploy your Docker Container.

What you need to create in Azure:

1. A Resource Group (if you don't have any)
2. A Container Registry
3. A App Service (Web App) if you want to create a API / Frontend



Name	Type	Location
ASP-DVerse-9420	App Service plan	West Europe
DVerseGoogleSearchAgent	Container registry	West Europe
DVerseGoogleSearchAgent	App Service	West Europe

Some extra stuff:

1. GitHub account with access to Github Actions
2. A GitHub Repository that contains the code you want to deploy
3. A Dockerized Container that is ready to be deployed

Setup

For automating deployment to Azure, we need to create a GitHub Action. This Action needs to be created inside the workflow directory inside the .GitHub directory.

1. Create a folder called: `.github/workflows`

In this folder, are all the Actions. When pushing something to Gitlab, all the files in this workflows directory are being checked. If you tell the file to be triggered, it will be. That is where the Pipeline starts to run.

2. Create a file with a logic name: `PushAzureContainerRegistry.GoogleSearchAgent.yml`

After creating the file, you need to add some code to it in order to actually do something. There are some required things that need to be in the file. We are going to add them first:

3. Add the required things

```
name: Build, Push and Deploy to Azure

on:
  push:
    branches:
      - main

jobs:
  build_and_push:
```

Each of the required things have their own purpose. I will describe them below:

- **Name**: The name of your Pipeline / Github Action.
- **On**: On tells this Action needs to be triggered based on your configuration. In this case, it is **triggered** every time something is pushes to the `main` branch.
- **Jobs**: The jobs section is used to define certain jobs that it should run. Such as building, pushing, testing and more.

So what is next? We need to configure the Jobs:

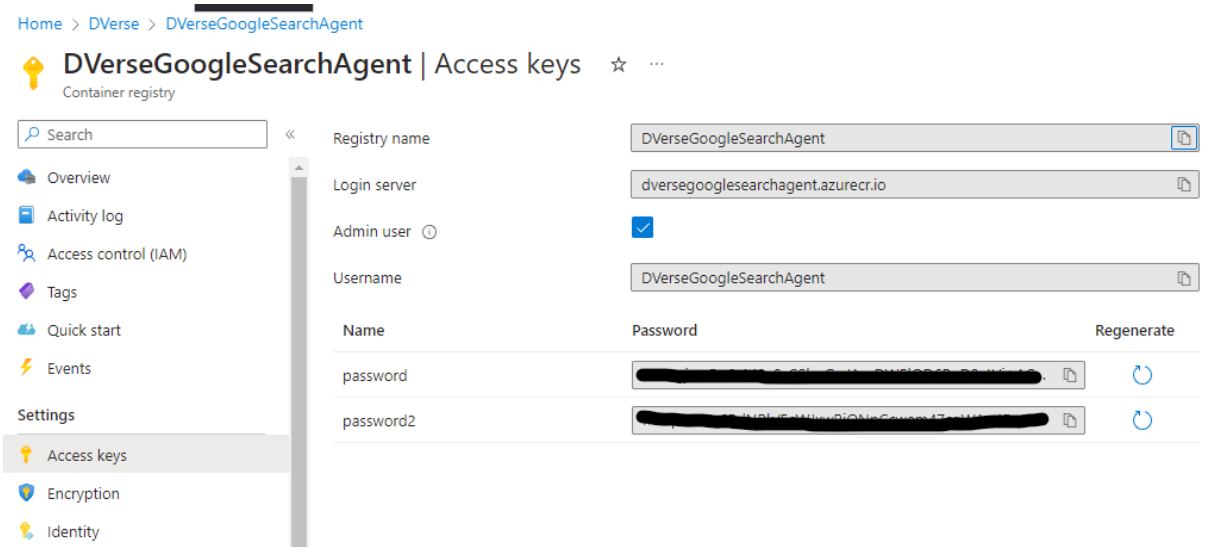
4. Add different types of Jobs:

```
name: Build, Push and Deploy to Azure

on:
  push:
    branches:
      - main

jobs:
  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 }}
```

As you can see, it does a lot, but it actually makes sense. In one step, you need to login Azure. This required 2 secrets: `REGISTRY_USERNAME` and `REGISTRY_PASSWORD`. You can get these secrets from the Container Registry in Azure under the tab Access Keys. There are 2 steps you need to take in order to login using the Pipeline. Go to your Azure Container Registry and copy your username and password.

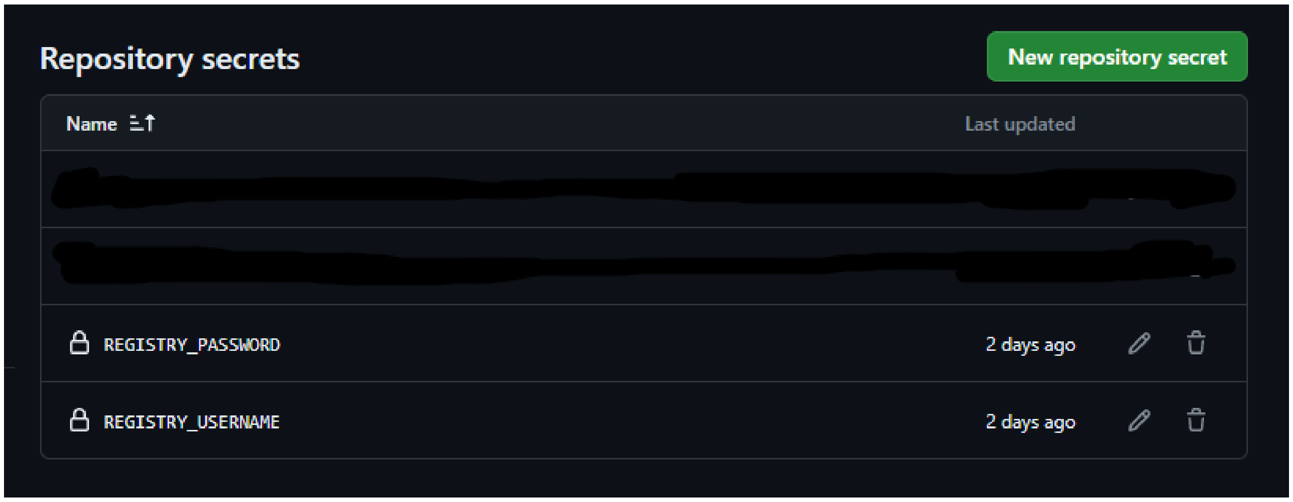
- Get the username and password from Azure



Registry name	DVerseGoogleSearchAgent
Login server	dversegooglesearchagent.azurecr.io
Admin user	✓
Username	DVerseGoogleSearchAgent

Name	Password	Regenerate
password	[redacted]	↻
password2	[redacted]	↻

- Create 2 GitHub Secrets in your repository settings. You can do this under Settings -> Security -> Secrets and Variables -> Actions -> Repository Secrets. Paste the copied username and password here. That should be it.

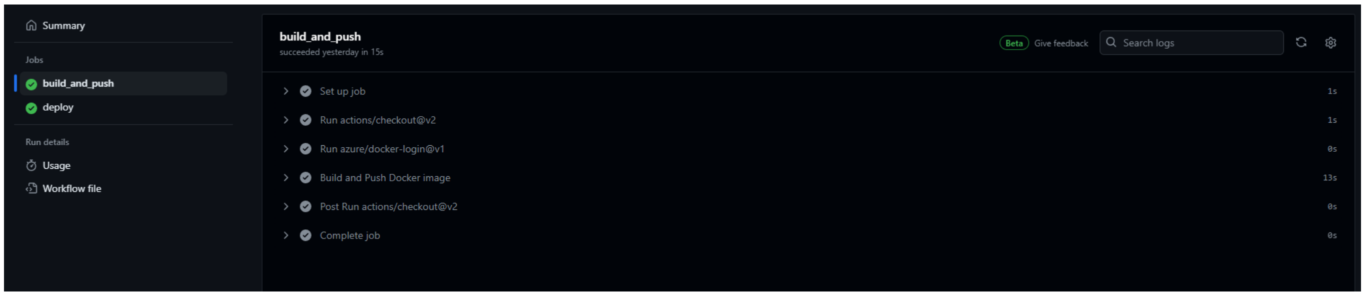


Repository secrets

New repository secret

Name	Last updated
[redacted]	[redacted]
[redacted]	[redacted]
REGISTRY_PASSWORD	2 days ago
REGISTRY_USERNAME	2 days ago

After the login had completed. You can see there are 2 commands that you already know if you have experience with Docker. It builds the image and after that it pushes the image to Azure Container Registry. Then it should look like this inside GitHub:



```
build_and_push
succeeded yesterday in 15s

> Set up job                        1s
> Run actions/checkout@v2           1s
> Run azure/docker-login@v1         0s
> Build and Push Docker image       13s
> Post Run actions/checkout@v2      0s
> Complete job                      0s
```

After we build and pushed the Docker Container to Azure Container Registry. It is not available and accessible on the web. That is where the Azure App Services comes in. The Azure App Service can be connected to the Azure Container Registry, this can be done in the Azure Portal, or also using the CI/CD.

5. Add a new Job to the Pipeline that deploys the Container to the App Service

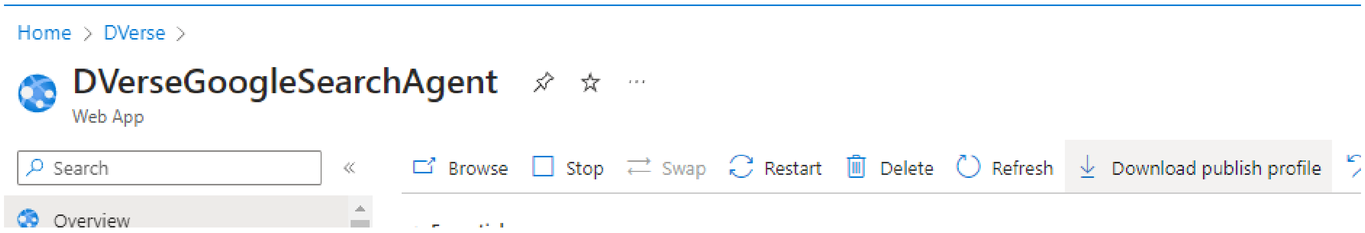
```yaml
name: Build, Push and Deploy to Azure

env:
  AZURE_WEBAPP_NAME: DVerseGoogleSearchAgent

on:
  push:
    paths:
      - GoogleSearchAgent/**
    branches:
      - main
```

```
permissions:
  contents: read

jobs:
  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 }}

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

    steps:

      - 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
}}'
```

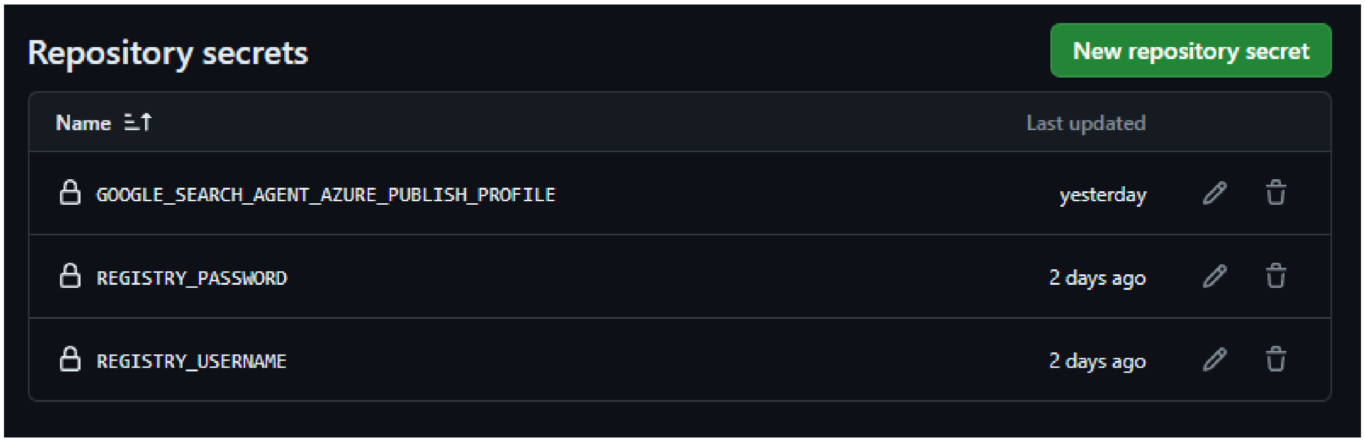
Here you can see the last step for the deployment of the Container Registry. The step is called `deploy`, it has a property called `needs` with the content `build_and_push`. The first job that we created earlier is called like this. What this does, it waits for the `build_and_push` job to be finished. When that job is finished, it starts this one.

Then on top of the file there is a new added env variable called `AZURE_WEBAPP_NAME`. This is the name of the Azure App Service. You require this one in order to deploy to the correct Azure App Service. After that, you have the `publish-profile`. You need to have the `publish-profile` setup as secret in GitHub.

You need to download the publish profile in your Azure App Service dashboard. As you can see on the picture below on the right.



Than you need the content of this file and add it to your GitHub Secrets as you already did with the username and password.


Repository secrets

New repository secret

Name	Last updated
GOOGLE_SEARCH_AGENT_AZURE_PUBLISH_PROFILE	yesterday
REGISTRY_PASSWORD	2 days ago
REGISTRY_USERNAME	2 days ago

And in the end, you have the command:

```
    images: 'dversegooglesearchagent.azurecr.io/myapp:${{ github.sha }}'
```

This specifies the latest image of your Azure Container Registry, and this means that now you have deployed the just earlier pushed image to the Azure App Service.

You can now visit the URL of your Azure App Service and that is it!!

Testing

TODO: Add the test phase in the CI/CD