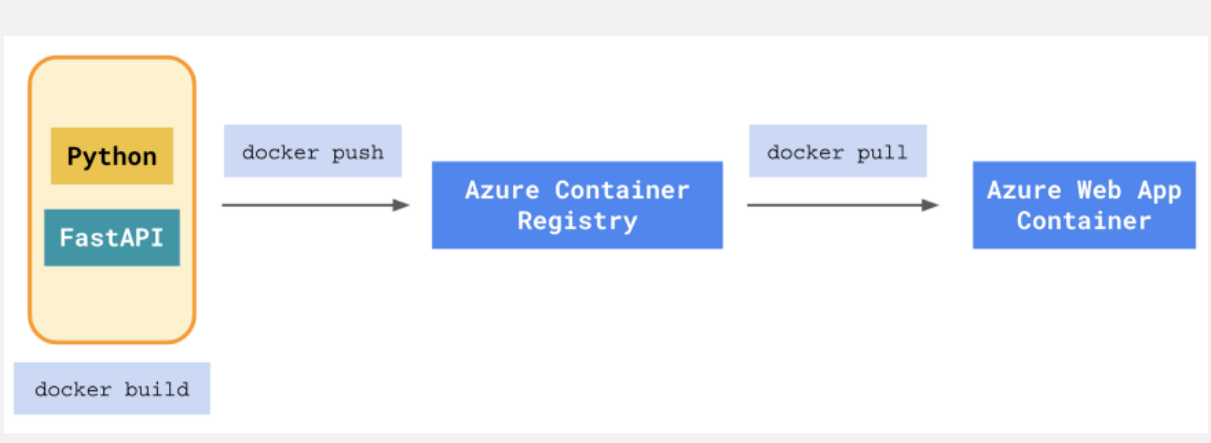
Github Actions

Github Actions is a new workflow automation tool for building, testing, and deploying your code directly from Github.

If you're already using Github for your code repository, there's no need to integrate a third-party tool to automate common developer workflows.

Simply place a Github Actions `.yml` file in your repository, designed to run in response to specific events. (push, pull requests, etc)

Workflow diagram



We will build and push a container image to Azure Container Registry, and pull the image into an Azure Web App for deployment.

The full list of packages and technologies we'll be utilizing include:

- VS Code — as the IDE of choice.
- FastAPI— Python API development framework.
- Docker Desktop— build and run Docker container images on our local machine.
- Azure Container Registry — repository for storing our container image in Azure cloud.
- Azure App Service — PaaS service to host our FastAPI app server.
- Github Actions — automate continuous deployment workflow of FastAPI app.

Docker

Head over to Docker Desktop and install the program.

This allows you to run Docker commands on your local machine.

Build and Run Container Image

Instead of deploying our app as a package, let's build our app as a Docker image, so that the build process is outlined in a `Dockerfile`.

`Dockerfile` makes the build process seamless and reproducible.

Build your container and check that the image has been created with the `docker images` command.

```
docker build . -t fastapi-cd:1.0
docker images
```

Run your container with the image tag you specified earlier, on port 8000.

```
docker run -p 8000:8000 -t fastapi-cd:1.0
```

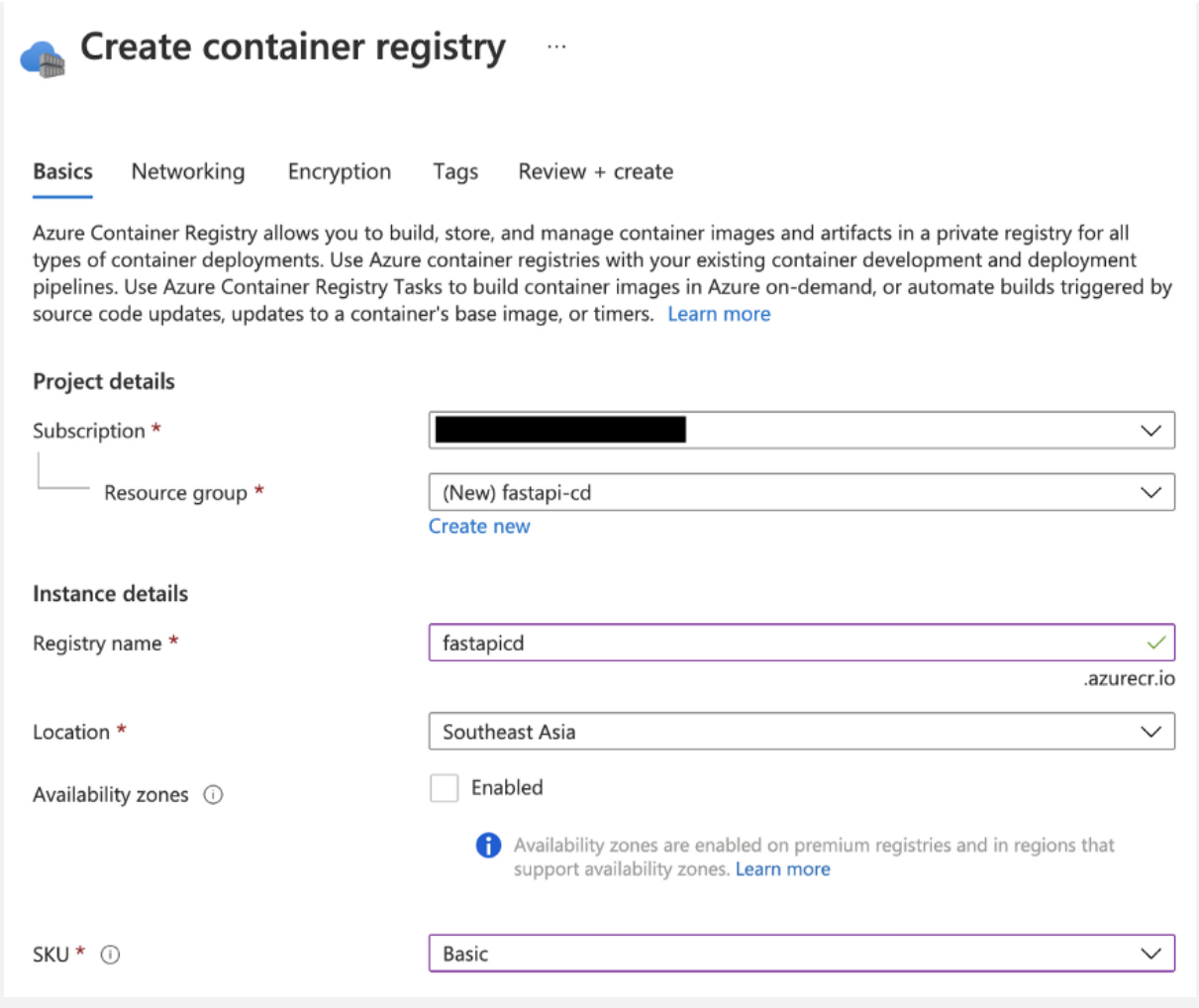
Check your containerized application by going to http://localhost:8000/docs.

You should see the same FastAPI application running.

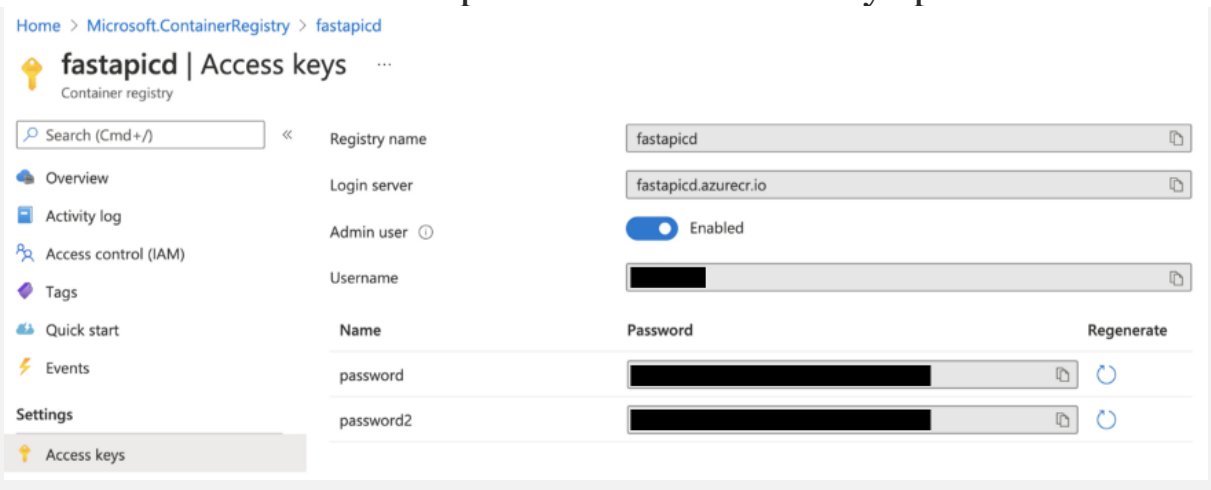
Infrastructure setup

Azure Container Registry

Time to deploy our application in Azure. First off, create an Azure Container Registry.



Enable the admin user option in the Access keys pane.



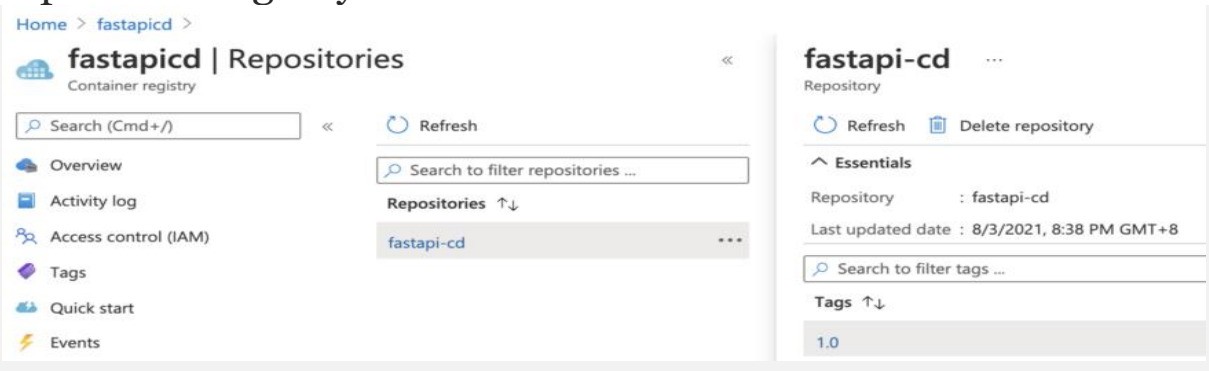
Going back to your VS Code terminal, log in to Azure with your account.

```
az login
az acr login --name fastapicd
```

Build and push your Docker image to the registry server.

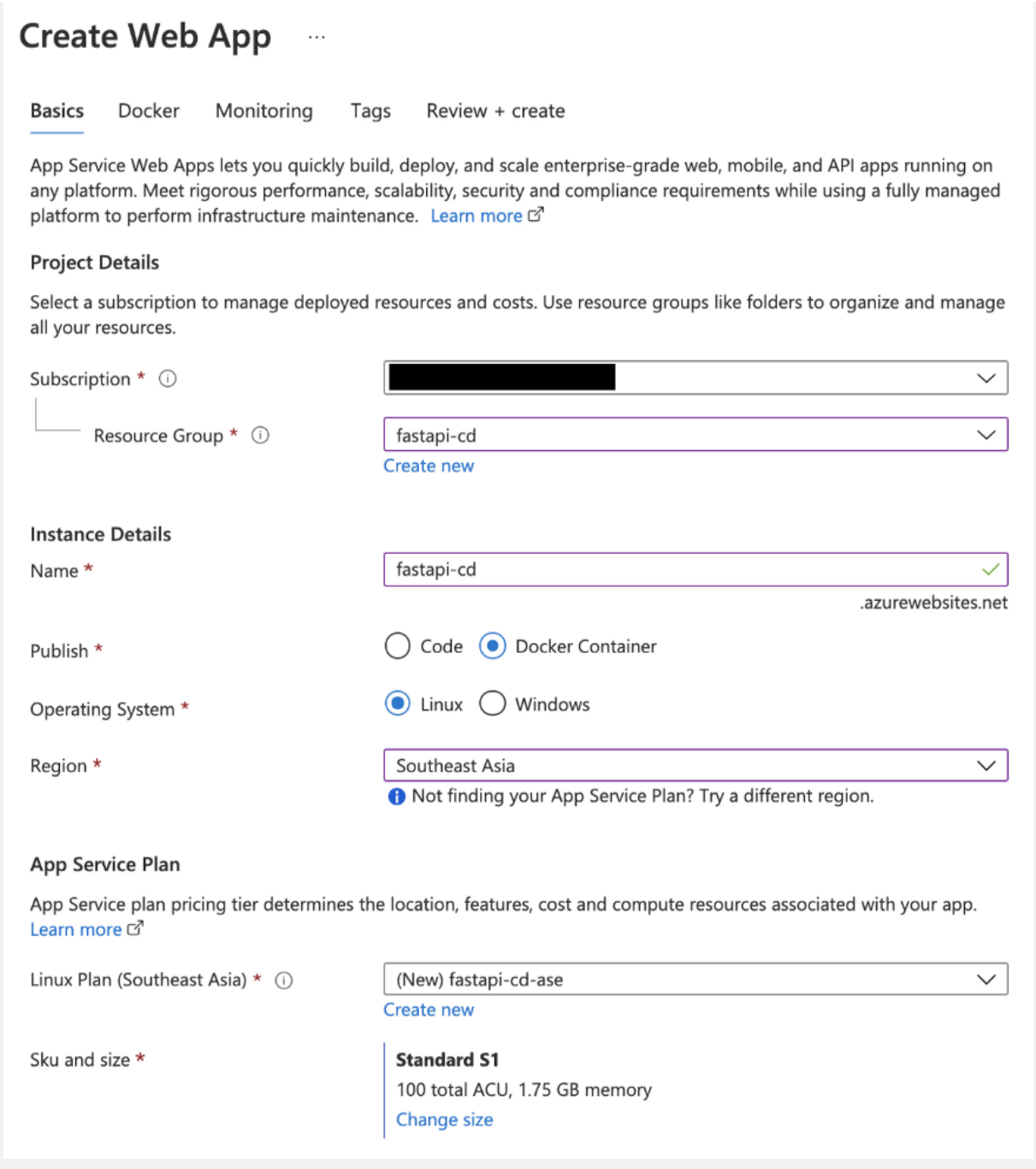
```
docker build . -t fastapicd.azurecr.io/fastapi-cd:1.0docker
images
docker push fastapicd.azurecr.io/fastapi-cd:1.0
```

Check that your Docker image has been successfully pushed to a repo in the registry.

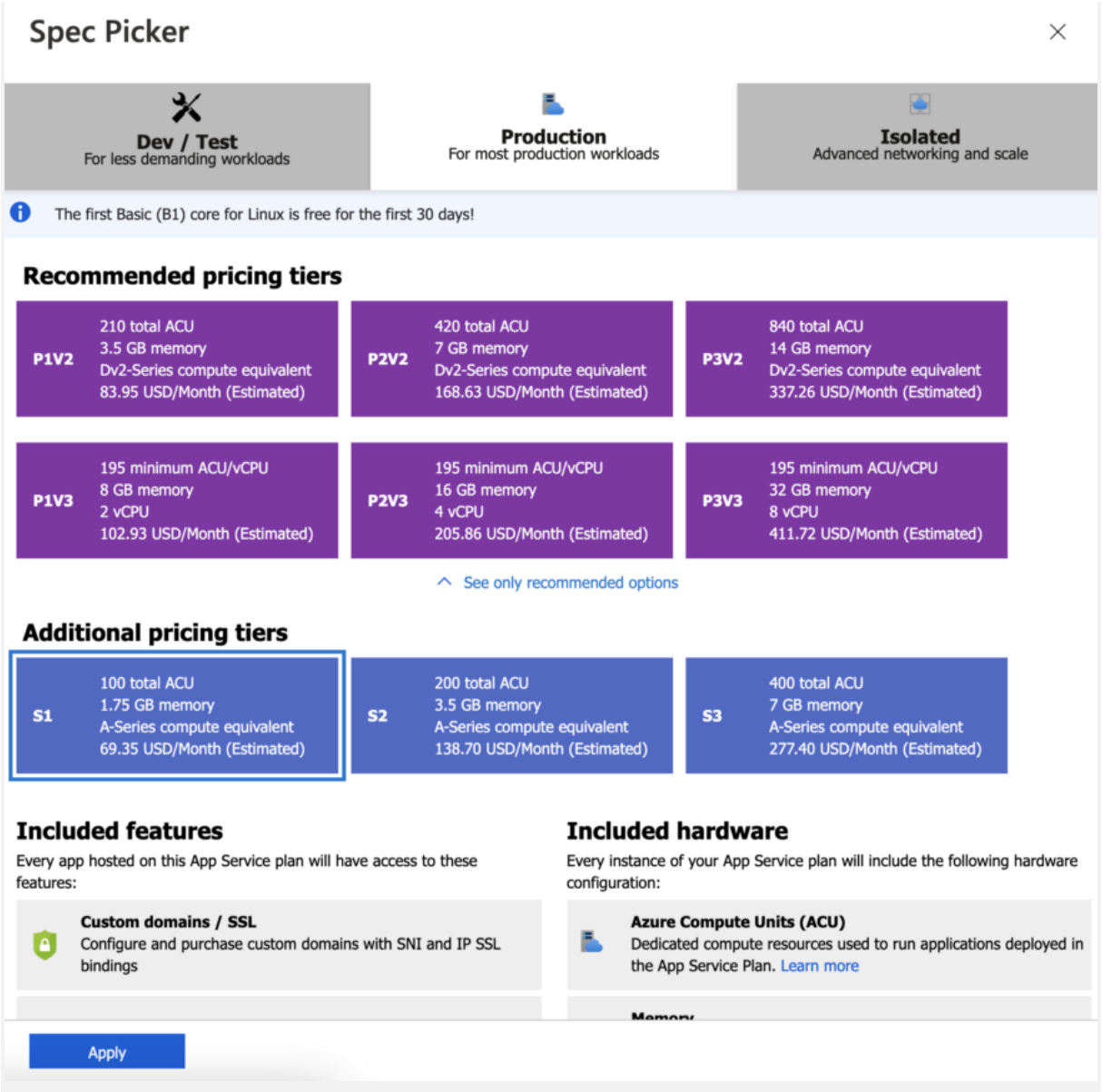


Azure App Service

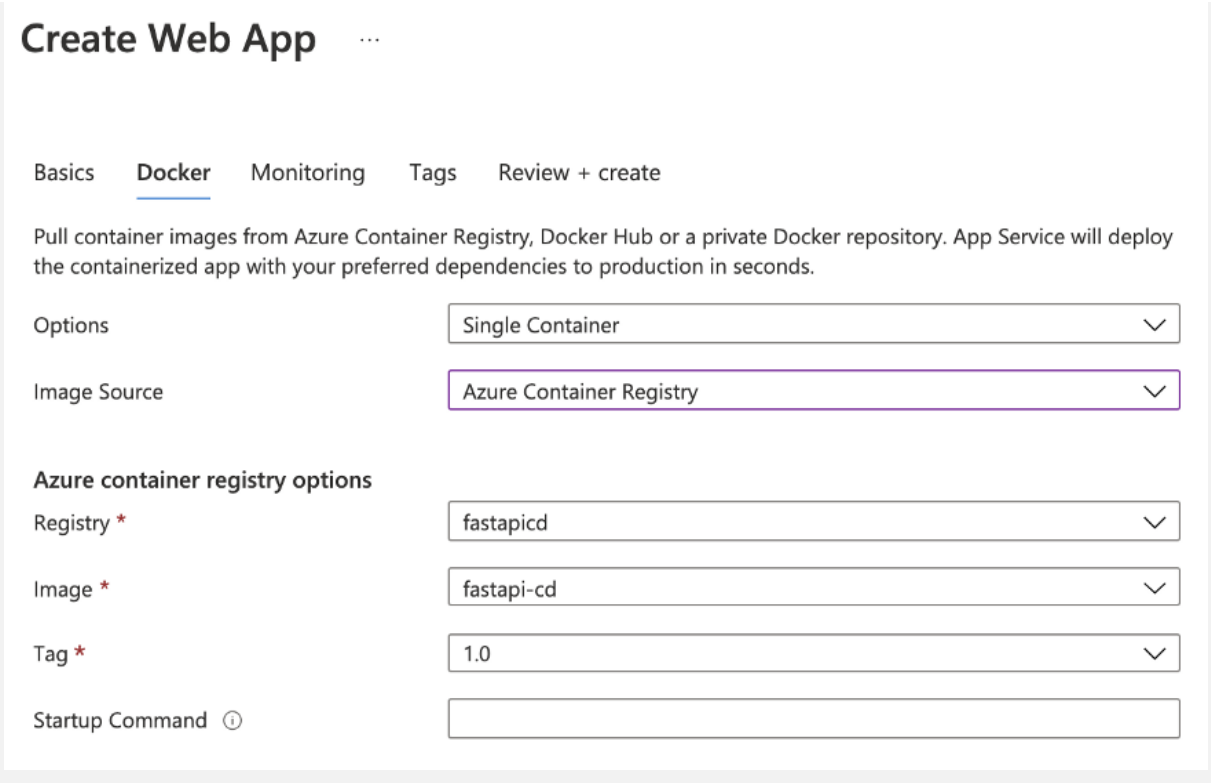
Next, create an App Service resource.



For the App Service Plan, change the size to Standard S1 SKU under the Production Workloads tab.



On the Docker configuration tab, make sure the container image you pushed earlier is selected.



Once resource deployment is complete, check that your web app is running by going to https://<your-webapp-name>.azurewebsites.net/docs.

You should see the FastAPI swagger docs page render.

Github Actions

Now we're ready to create our Github Actions .yml file that will automate deployment of our FastAPI application.

In VS Code, create a `.github/workflows` *directory* and a `prod.workflow.yml` file within.

```
mkdir .github
cd .githubmkdir workflows
cd ..touch .github/workflows/prod.workflow.yml
```

Build your `prod.workflow.yml` file like below:

```yaml
name: Build and deploy to production

on:
  push:
    branches:
      - main

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:

    - name: Checkout GitHub Actions
      uses: actions/checkout@main


    - name: Login via Azure CLI
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}


    - name: Login to Azure Container Registry
      uses: azure/docker-login@v1
      with:
        login-server: fastapicd.azurecr.io
        username: ${{ secrets.REGISTRY_USERNAME }}
        password: ${{ secrets.REGISTRY_PASSWORD }}


    - name: Build and push container image to registry
      run: |
        docker build . -t fastapicd.azurecr.io/fastapi-cd:${{ github.sha }}
        docker push fastapicd.azurecr.io/fastapi-cd:${{ github.sha }}


    - name: Deploy to App Service
      uses: azure/webapps-deploy@v2
      with:
        app-name: 'fastapi-cd'
        images: 'fastapicd.azurecr.io/fastapi-cd:${{ github.sha }}'
        slot-name: 'staging'


    - name: Azure logout
      run: |
        az logout
```

Now, whenever we do a git push to the main branch, Github Actions will run a deployment job that consists of the 5 steps above, each with its own name.

Push this new file to Github using the git commands:

```
git add .
git commit -m "github actions deployment workflow"
git push
```

Service Principal

To automate our deployment workflow with Github Actions, we need to give the actions runner a service principal to authenticate to Azure, and perform the app deployment.

In VS Code terminal, run:

```
az ad sp create-for-rbac --name "github-actions" --role
contributor --scopes
/subscriptions/<GUID>/resourceGroups/fastapi-cd --sdk-auth
```

You can get the subscription GUID from the subscription you used to create the resource group.

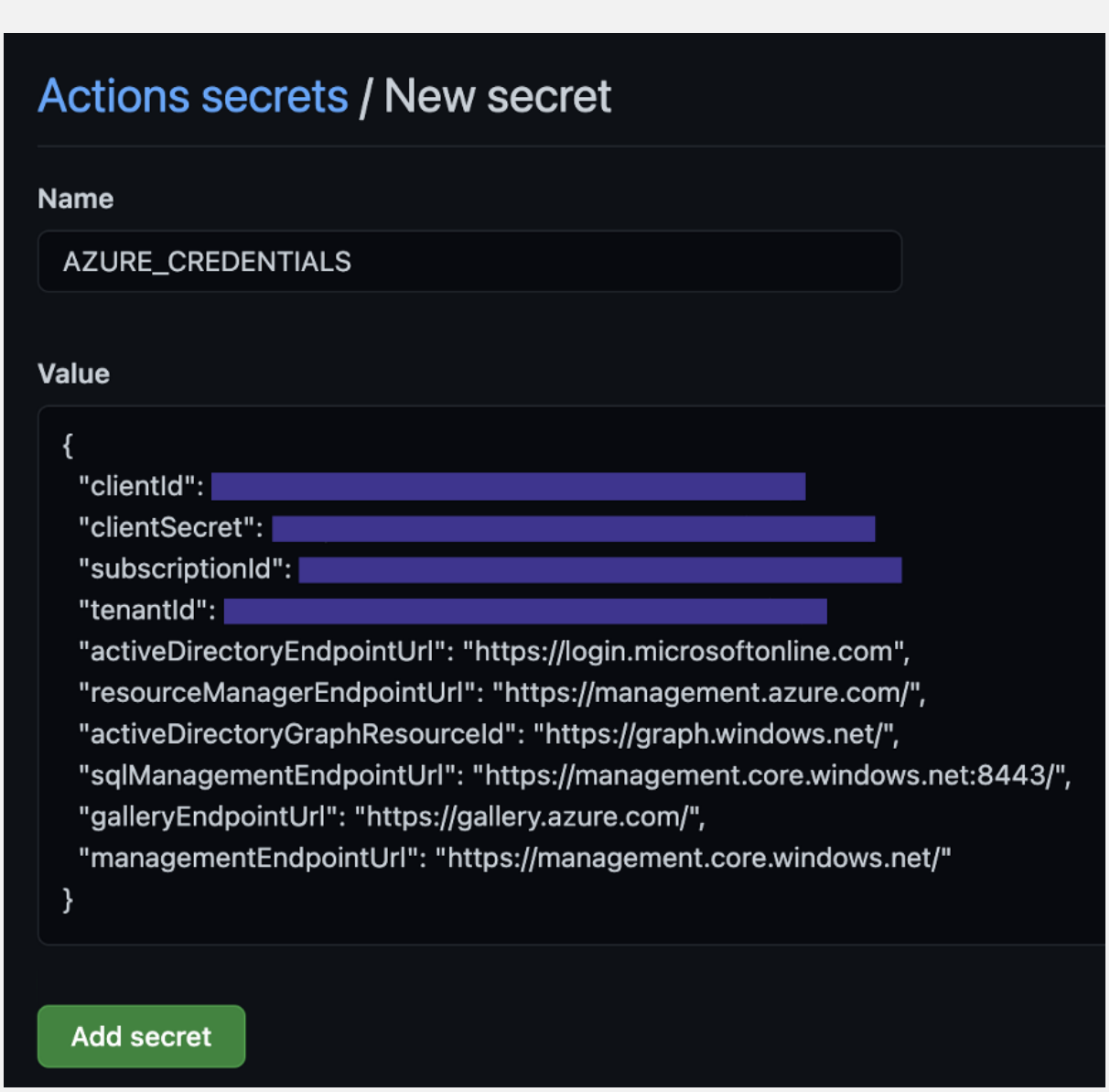
You will get a response like below:

```
{
    "clientId": "<clientId>",
    "clientSecret": "<clientSecret>",
    "subscriptionId": "<subscriptionId>",
    "tenantId": "<tenantId>",
    "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
    "resourceManagerEndpointUrl": "https://management.azure.com/",
    "activeDirectoryGraphResourceId": "https://graph.windows.net/",
    "sqlManagementEndpointUrl":
"https://management.core.windows.net:8443/",
    "galleryEndpointUrl": "https://gallery.azure.com/",
    "managementEndpointUrl": "https://management.core.windows.net/"
}
```

Copy the response and save it. You won't be able to see it again.

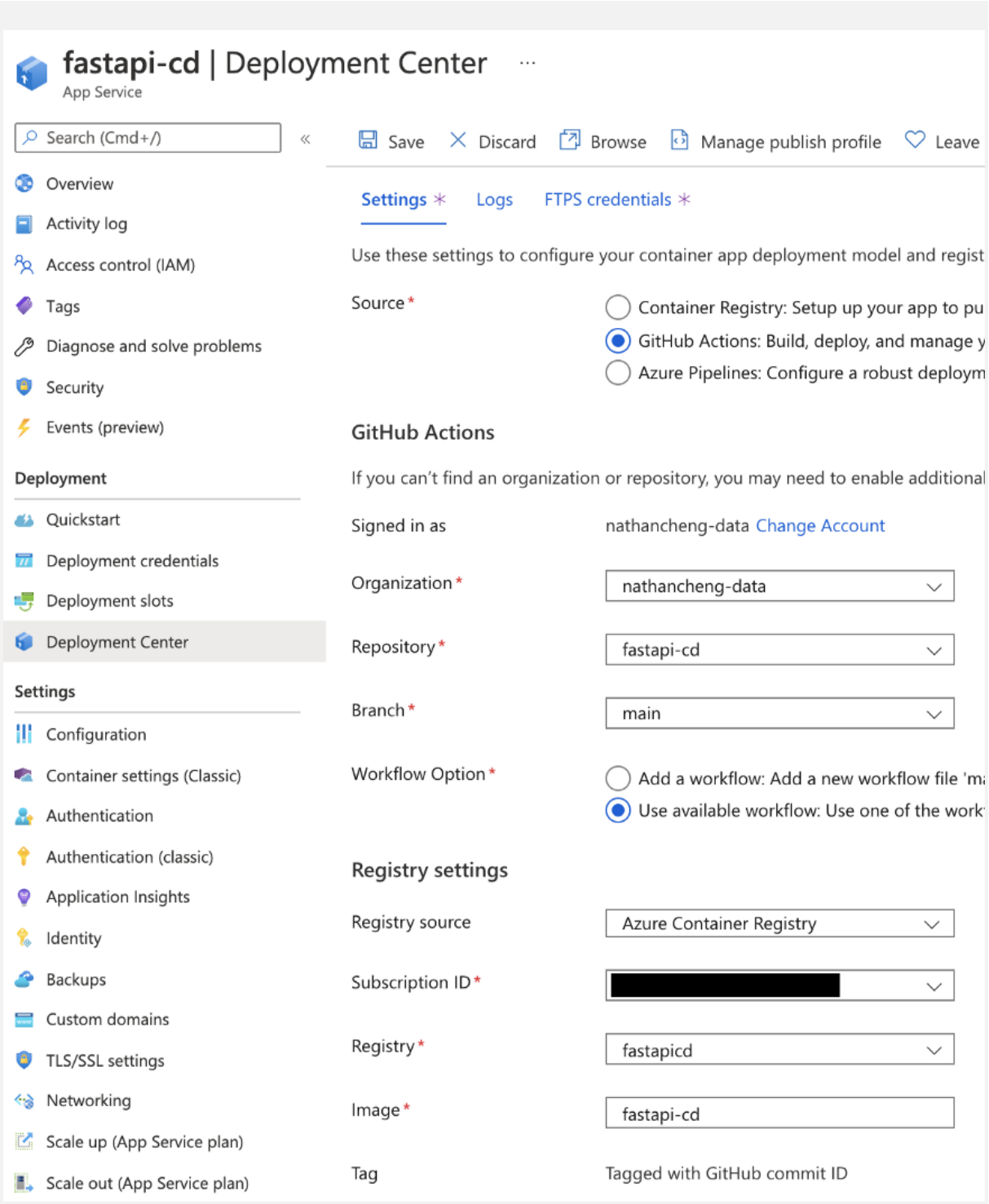
Head over to Github settings, and create 3 Github Secrets:

- AZURE CREDENTIALS: Entire JSON response from above.
- REGISTRY_USERNAME: clientId value from JSON response.
- REGISTRY_PASSWORD: clientSecret value from JSON response.



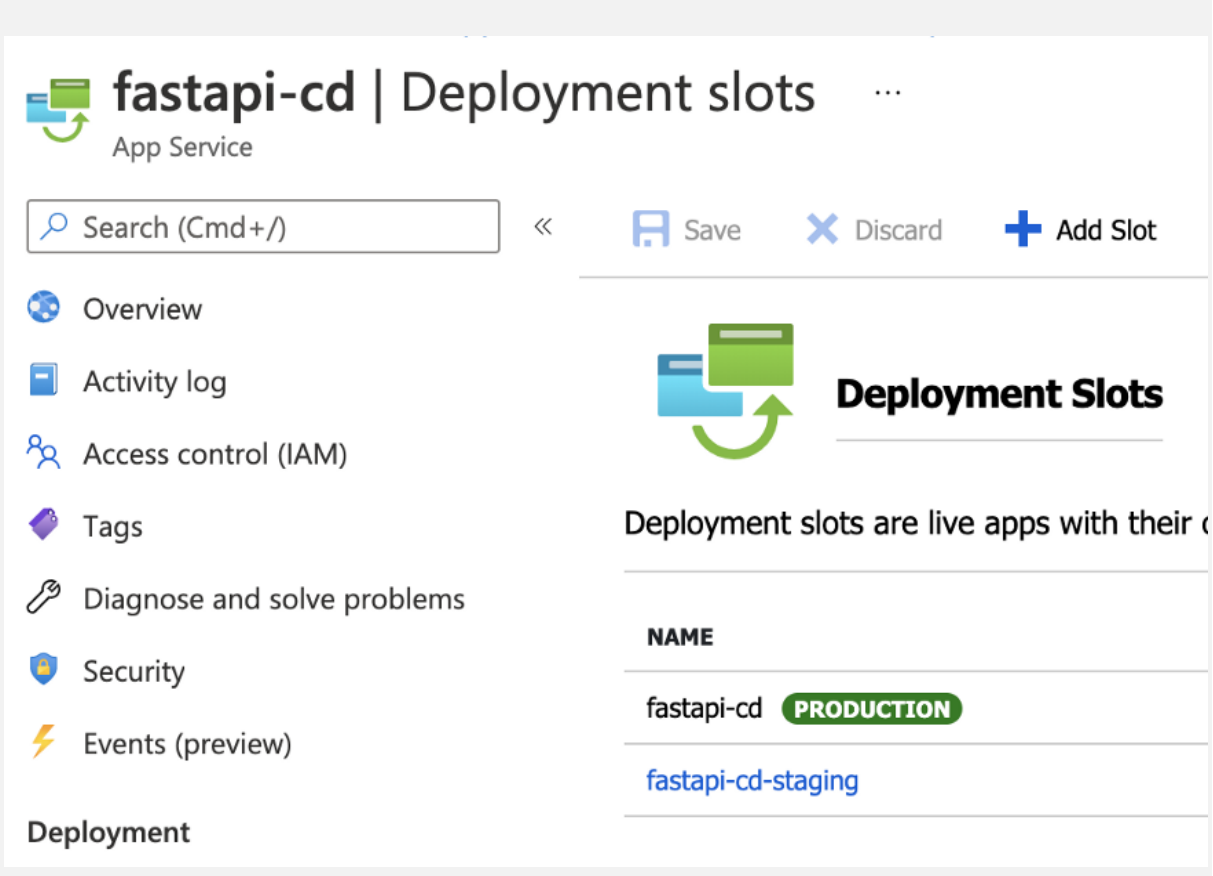
Configure App Service to use Github Actions for continuous deployment

In the Deployment Center pane of the App Service, link your Github repo main branch, and configure deployment to use Github Actions.



Create staging deployment slot

The current app is running in a production slot. To guarantee higher-quality deployments, we will create a staging slot.



Slots are useful for multiple reasons:

- Changes can be released to a smaller % of users in a staging slot, to validate new releases before swapping the staging slot with production.
- Staging slot app instances are warmed up before being swapped into production, guaranteeing that the app is responsive to requests.
- If something goes wrong with the new release, swaps can be performed again to roll back the changes.

Check that the staging app is running like the production app by going to https://<your-webapp-name>-staging.azurewebsites.net/docs.

Configure the same deployment center settings for the staging slot as you did for production.

When you're comfortable swapping the staging app over to production, simply click on Swap in the Deployment Slots pane of the App Service.

I would like to give a big shoutout to all the extensive documentation and YouTube videos that helped to complete the task:

- Deploy a custom container to App Service using Github Actions
- Github Action repo — deploy to App Service
- Set up staging slots in App Service
- Learn Docker in 7 Easy Steps — Full Beginner's Tutorial
- Containerize FastAPI app with Docker
- Push Docker image to Azure Container Registry