**Important:**

In this lab, d25001 is used as a placeholder roll number to make unique names for some of the artefacts like DNS name. Search for d25001 and replace with your Praxis roll number.

In this lab, red font is used for commands. And in the commands, blue highlight is used for assigned names.

Why are some of the variable names so long? The intent was to help understand the flow and steps easily.

Hope you enjoy this lab.

**Assumption: Lab 1 is completed, and the results are retained**

This lab reuses Working dir, files in the working dir, Azure resource group, ACR, Image pushed to ACR etc. from Lab 1.

**Step 2: Set up Azure resources**

**Log in to Azure**

Execute this in CMD:

az login

If you have already logged into azure in a browser, this step should just require you to click Enter again. Your subscription ID will be displayed at this step. Note it.

In lab 1 we have already created the following:

Resource group with name myResourceGroup

ACR with name myacrd25001 it contains image with name

myacrd25001.azurecr.io/my-fastapi-app-image:latest

**Create Azure App Service plan which will host the App**

az appservice plan create --name fastapi-app-plan --resource-group myResourceGroup --sku B1 --is-linux

* in the previous command, if you get any “no quota” errors, try:

*az appservice plan create --name fastapi-app-plan --resource-group myResourceGroup --sku F1 --is-linux --location eastus*

* try different locations from this list: eastus,eastus2,centralus,westus,westus2,southcentralus,northcentralus,westeurope,northeurope,southeastasia,eastasia,japaneast,japanwest,australiaeast,australiasoutheast,brazilsouth,southafricanorth,uae-north,canadacentral,canadaeast

**Create a plan and web app**

az webapp create --name fastapi-webapp-d25001 --plan fastapi-app-plan --resource-group myResourceGroup --deployment-container-image-name myacrd25001.azurecr.io/my-fastapi-app-image:latest

**Create Azure Service Principal**

In the following command, insert your subscription ID and execute.

az ad sp create-for-rbac --name "github-actions-sp" --role contributor --scopes /subscriptions/<your subscription ID>/resourceGroups/myResourceGroup --sdk-auth --query "{client\_id: appId, client\_secret: password, tenant\_id: tenant}" --output json

* Example:

az ad sp create-for-rbac --name "github-actions-sp" --role contributor --scopes /subscriptions/12345678-1234-1234-1234-123456789abc/resourceGroups/myResourceGroup --sdk-auth --query "{client\_id: appId, client\_secret: password, tenant\_id: tenant}" --output json

*The output will be like this. Copy this and save for a future step.  
{*

*"clientId": "your-client-id",*

*"clientSecret": "your-client-secret",*

*"subscriptionId": "your-subscription-id",*

*"tenantId": "your-tenant-id",*

*…}*

**Step 3: Set up GitHub**

Navigate to working dir in a terminal

**Initialize git using:**

git init

**add the working dir to git’s safe list**

git config --global --add safe.directory "D:/working/dir/full/ path”

* Example:

git config --global --add safe.directory "D:/praxis/lab”

**Add all files to the staging area**

git add .

**Commit to the local git repo**

# in the following command, if you get any errors, try typing instead of copy pasting especially the double quote and the hyphen

git commit -m "Initial commit for FastAPI CI/CD lab"

**Create repo in GitHub.com**

Using a browser, log in to GitHub, create a repo with name fastapi-azure-lab2

Note the URL of this repo for e.g.

[https://github.com/<your-username>/fastapi-azure-lab2.git](https://github.com/%3cyour-username%3e/fastapi-azure-lab2.git)

**Add the remote repo to Git**

git remote add origin [https://github.com/<your-username>/fastapi-azure-lab2.git](https://github.com/%3cyour-username%3e/fastapi-azure-lab2.git)

**Rename branch to main**

git branch -M main

**Push your local changes to github**

git push -u origin main

**Verify results**

The repo in GitHub must now contain the files from your working dir. Verify using a browser.

**Set up GitHub secrets**

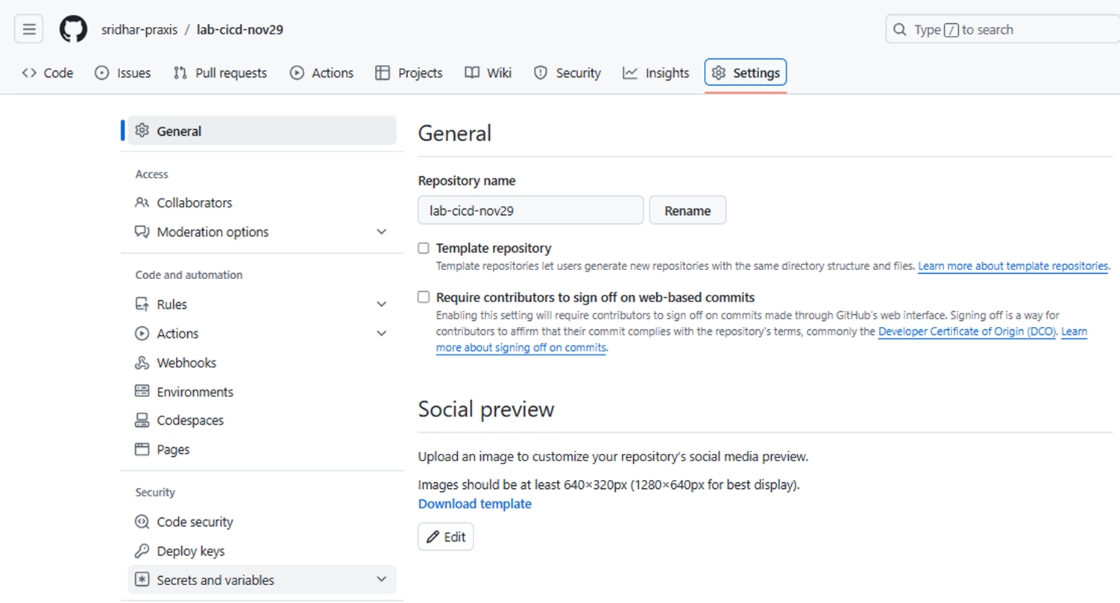
This step involved a few simple yet vital steps. Carefully follow instructions.

In GitHub, click the repo -> settings -> Secrets and Variables -> Actions – New repository secret

This allows a new secret to be set up. Repeat this step for each of the following secrets.

|  |  |
| --- | --- |
| Secret name | Value |
| AZURE\_RESOURCE\_GROUP | myResourceGroup |
| ACR\_NAME | myacrd25001 |
| AZURE\_PLAN | fastapi-app-plan |
| WEBAPP\_NAME | fastapi-webapp-d25001 |
| AZURE\_CREDENTIALS | Paste the JSON exactly as received from the Service Principal step |

Note: *After adding, it will not be visible for security reasons*



For example:

A screen shot of a message

Description automatically generated

After the secrets are added, it will look like this:

A screenshot of a computer

Description automatically generated



**Set up GitHub Actions Workflow**

In GitHub select the repo click “Add file”, paste this in the file name: .github/workflows/ci-cd.yml

This step will create folder structure expected by GitHub.

From the lab2 files copy contents of ci-cd.yml and paste into this file and commit.

Any commits to the main branch in this repo will trigger execution of this YAML file. This initiates steps needed to build docker image of the repo, push the image, update web app with the new image and deploy this web app.

**Observe the progress of CI/CD workflow**

In GitHub -> repo -> Actions ->left panel has the Deploy Workflow, click and observe the run of the ci-cd.yml file.

A screenshot of a computer

Description automatically generated

**Step 4: Access the Web App and test CI/CD**

**Access the webapp**

* The webapp will be created and available thru the URL:
  + <https://fastapi-webapp-d25001.azurewebsites.net/>
* Go to app services in portal.azure.net and see more details of this web app
* Also try URL/docs to test the model prediction
* This can also be accessed using CMD using curl

**Test if web app gets refreshed if FastAPI app is updated**

* # get the latest from origin
* git pull origin main
* # Make some changes to main.py, for e.g. a minor change to the print statement in GET endpoint
* # add the modified file to staging area
* git add main.py
* # commit to local git repo
* git commit -m “Made a minor change to main.py”
* # push to github
* git push origin main
* # this will trigger the re-run of the actions set up in the github repo and deploy the updated version to the endpoint
* # Go to the GitHub actions page and observe the run progress
* # refresh app URL in a browser and see the change appear

**Step 5: Clearing up Azure resources**

* Delete the entire resource group

az group delete --name myResourceGroup --yes --no-wait

* Delete Service Principal (SP)

az ad sp list --display-name "github-actions-sp" --query "[].appId" -o tsv

az ad sp delete --id <output-from-above-command>

* List all app registrations

az ad app list --query "[].{Name:displayName, AppId:appId}" -o table

az ad app delete --id <AppId>