Udacity Flask ML Project – CICD Setup

# Getting Started

## Create an SSH key

In Azure Portal bash CLI, run the following command:

ssh-keygen -t rsa

Prompts can be left blank.

In the outputted text find the public key path, it will look something like:

Your public key has been saved in /home/xx<username>xx/.ssh/id\_rsa.pub

Display the public key by running the cat command with the public key path:

cat /home/xx<username>xx/.ssh/id\_rsa.pub

A complete example of this can be seen below:

Text

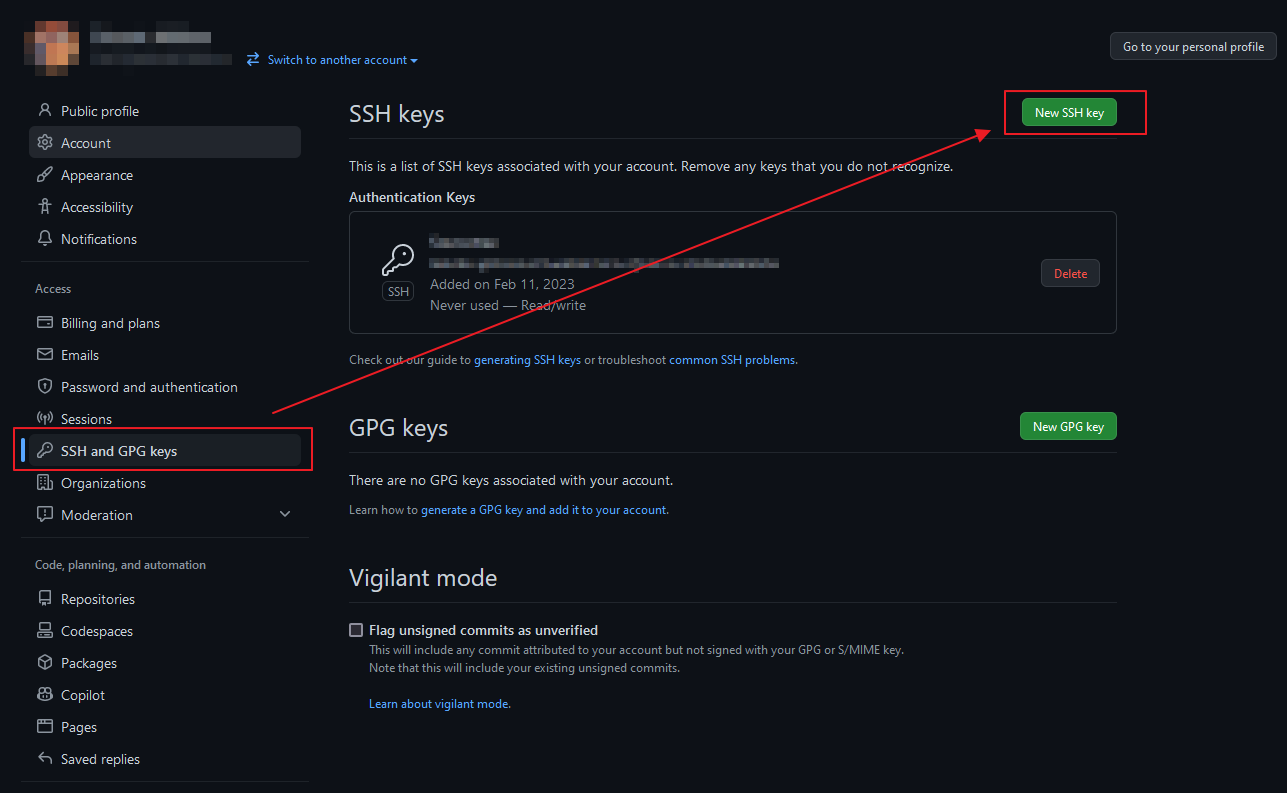
Description automatically generated

Next you must copy the public key into your GitHub account.

## Copy Public Key to GitHub

In your GitHub account, navigate to **Settings**, then find **SSH and GPG Keys** in the side menu

Click **New SSH Key**



Copy the key from your Azure bash terminal into the GitHub SSH key box, then click **Add SSH Key**

A screenshot of a computer

Description automatically generated with medium confidence

Now you will be able to clone the project from your GitHub account into the Azure Portal bash CLI

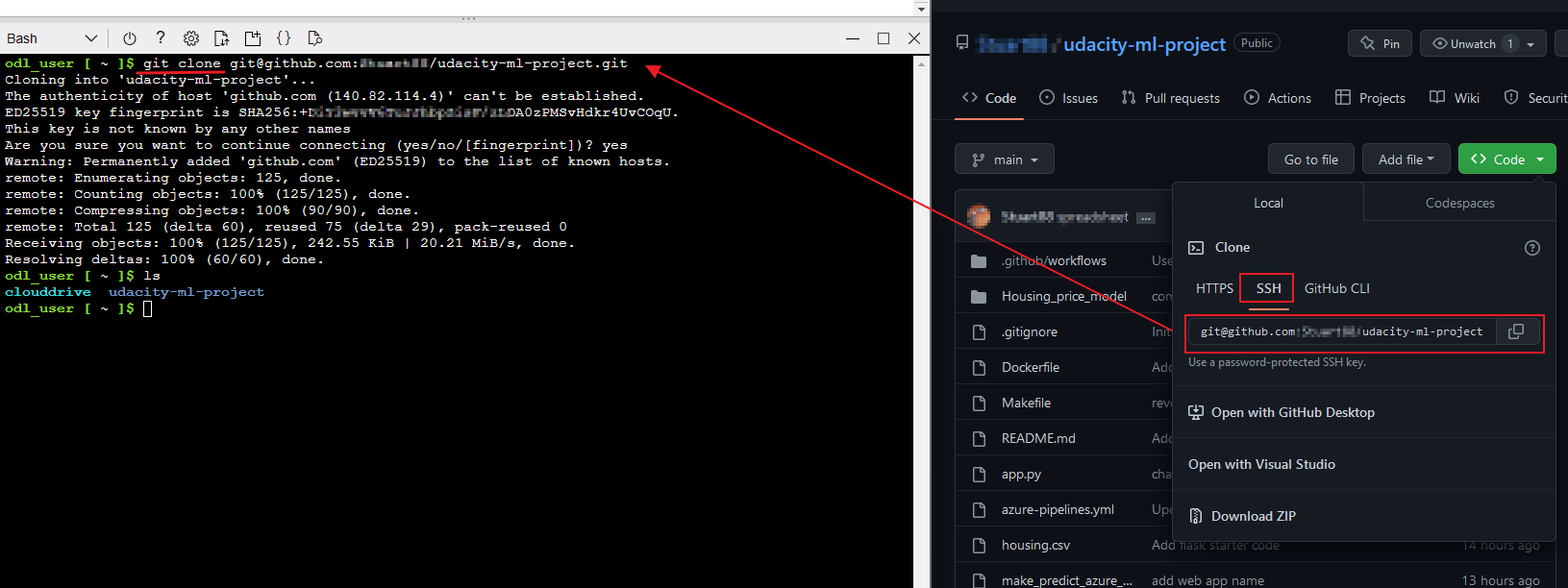
## Clone the Repository

Copy the SSH clone path from your GitHub repository, then run the Azure bash command to clone it into your cloud terminal:

git clone git@github.com:xxgitaccountxxx/udacity-ml-project.git

Be sure to use the SSH clone url, not HTTPS

As a check, you can then run the ls command to see that the project repo has been cloned



Now check the project builds as expected.

## Install, Lint and Test in a Virtual Environment

It is important to use a virtual environment to ensure the configuration and tools being used are what the project expects.

Navigate into the project directory:

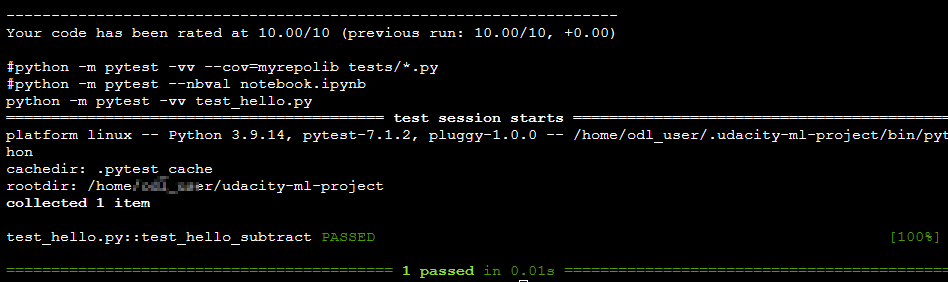
cd udacity-ml-project

Then run the following commands:

make setup  
source ~/.udacity-ml-project/bin/activate  
make all

These steps will create a virtual environment, navigate into the environment, then build the project as defined by the makefile, which runs linting and testing on the underlying code.

The result should be as below:

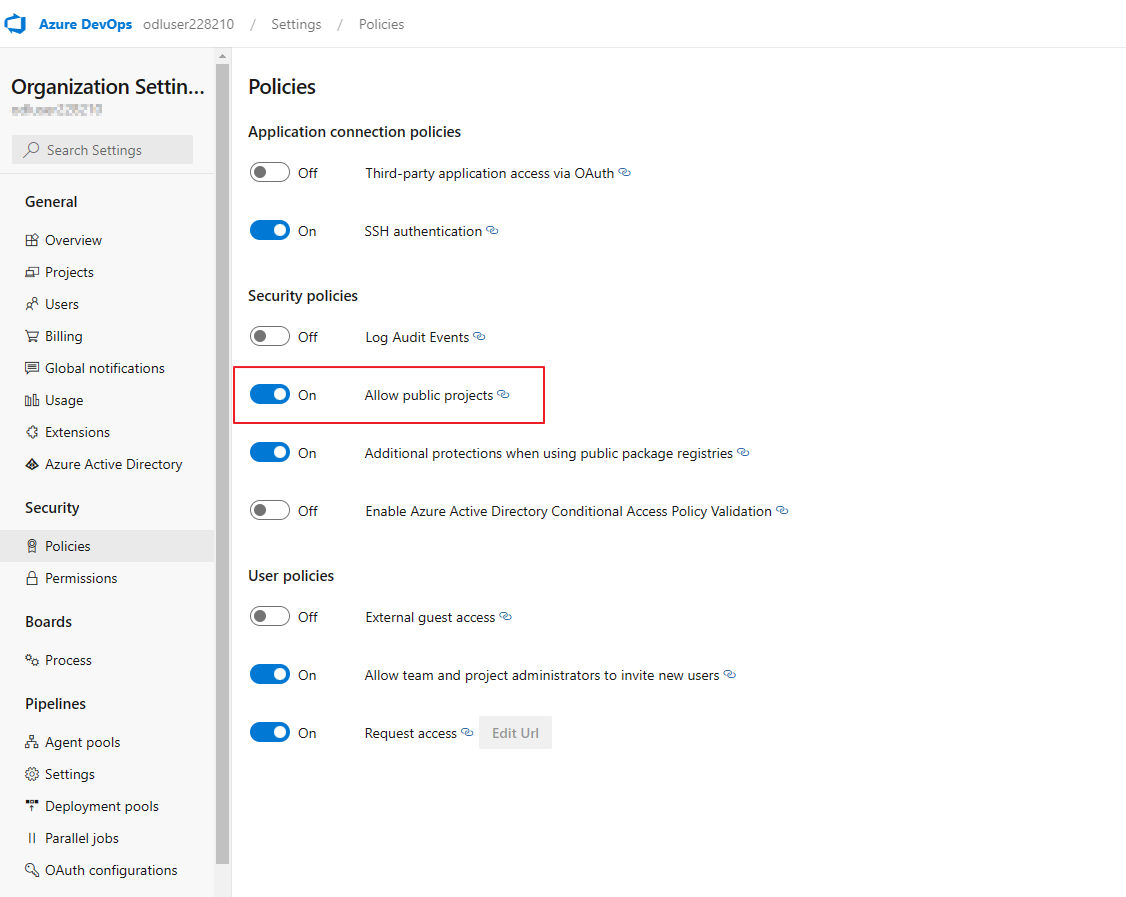


Now we are confident that the project is in good order, and so we can move onto creating a build pipeline in Azure.

## Setup Azure Pipeline

Open your Azure DevOps organisation account.

First, ensure the **Allow public projects** policy is set to **On**



Then create a new project and ensure **Visibility** is set to **Public**

Graphical user interface, application

Description automatically generated

Now go to project settings and add a GitHub connection to your project repository

Graphical user interface, text, application, website

Description automatically generated

Now that we have connected Azure DevOps to GitHub, we will need to create build a web app and a build agent for deploying the project each time the pipeline runs.

## Create a web app using the Azure Portal bash CLI

In the same bash CLI folder as you were in before, enter the following command:

az webapp up --name <YOUR WEB APP NAME> --resource-group Azuredevops --runtime "PYTHON:3.10"

This will create and launch a web app for the project.

Upon completion, you can view the project live on:  
https://<YOUR WEB APP NAME>.azurewebsites.net

Graphical user interface, text, application

Description automatically generated

At this point, if you have correct user privileges in your bash shell, you can run ./make\_predict\_azure\_app.sh to obtain results from the ML application.

*Be sure to first modify make\_predict\_azure\_app.sh file so that it navigates to the URL of the web app you just created.*

Here is an example of the result:

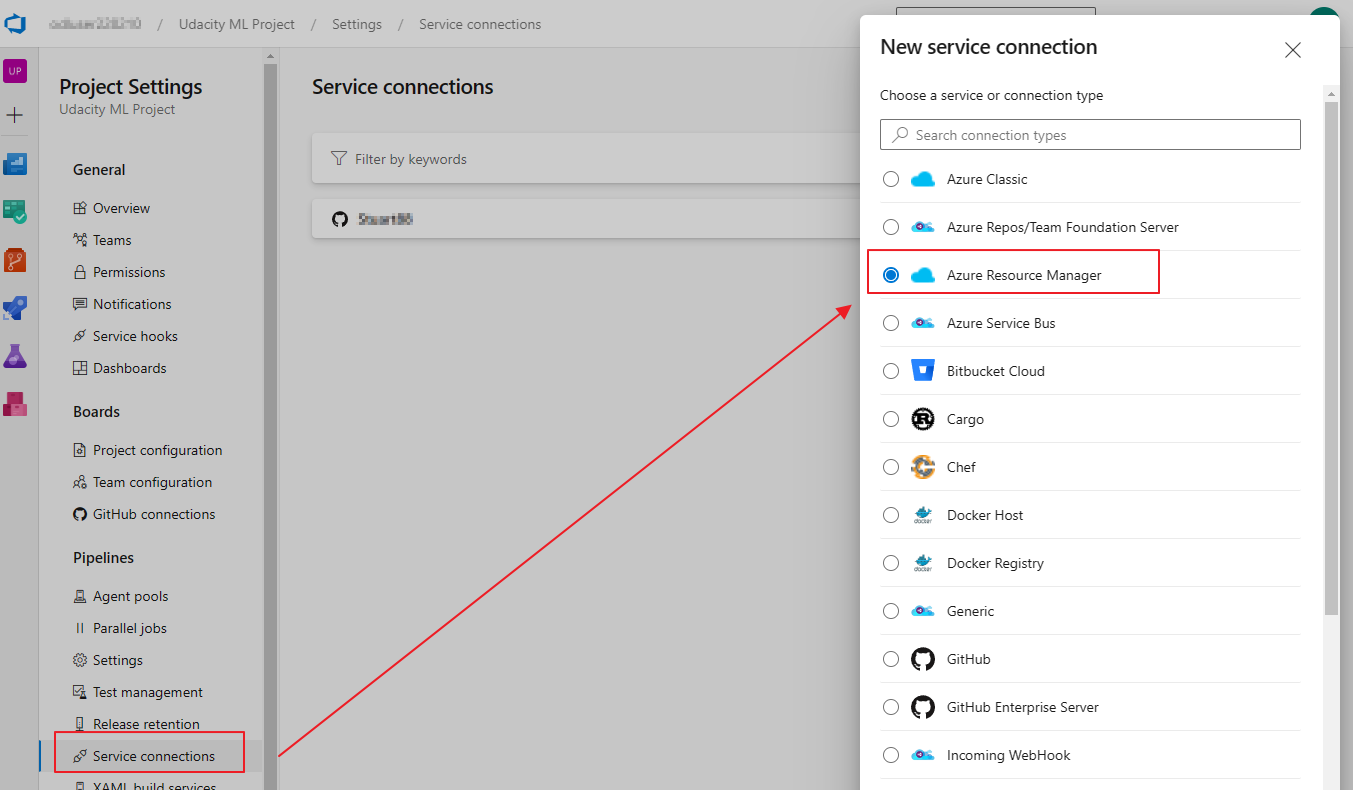


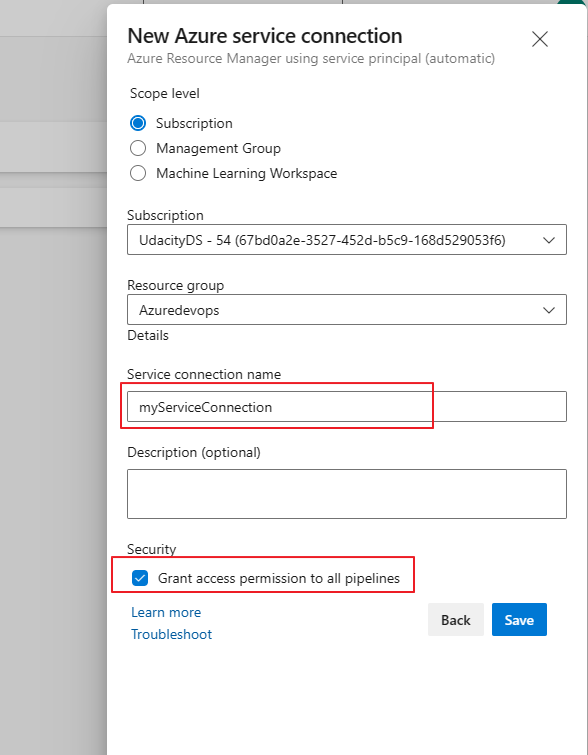
## Create a Service Connection

In Azure DevOps, navigate to Service connections in the settings menu, and create a new service connection.

Select Azure Resource Manager, then name it myServiceConnection and leave everything else as default.

Ensure **Grant access permission to all pipelines** is checked





## Create an Agent Pool

First, create a new Personal Access Token will full permissions granted in your Azure DevOps account. Don’t forget to make a note of the token.

Next, go **settings** and **Agent pools**, then create a new **self-hosted** agent pool called myAgentPool, with **full access permissions granted**.

Graphical user interface, application

Description automatically generated

Next we need to create a Linux Virtual Machine in the Azure Portal for running the build and deployment jobs.

## Create a Linux VM

In Azure Portal, create a new Azure virtual machine called myLinuxVM, using Ubuntu Server 20.04, password authentication (create your own username and password), and allow public inbound ports SSH(22)

Graphical user interface, application, email

Description automatically generated

Leave all else as default, then **Review + Create**

Now we can install the build agent on the virtual machine

## Install a Build Agent on the VM

Note the public ip address of the new VM, then use the bash terminal to navigate into the vm using the command:

ssh devopsagent@xx.xx.xx.xx 🡨 the public ip address

Once inside the vm, perform the following commands to setup Docker

sudo snap install docker  
sudo groupadd docker  
sudo usermod -aG docker $USER  
exit

Restart the VM from within Azure Portal

Now go back to Azure DevOps agent pool, create a new agent, select Linux, then copy the link for downloading the agent. We will use this link in the vm we just created.

Graphical user interface, application

Description automatically generated

Use the ssh command to navigate back into the vm, then run the following commands

curl -O https://<download-url-you-just-copied>../the-file-name...tar.gz  
mkdir myagent && cd myagent  
tar zxvf ../the-file-name...tar.gz  
./config.sh

This will download and unpack the build agent, then begin the configuration process.

In the config process, press Y to accept the agreement, then provide your Azure DevOps org URL, the access token you made earlier, and the name of the agent pool we made earlier.

The image below shows how this should look.

Text

Description automatically generated

Now run the following commands to complete the agent installation and start the service:

sudo ./svc.sh install  
sudo ./svc.sh start

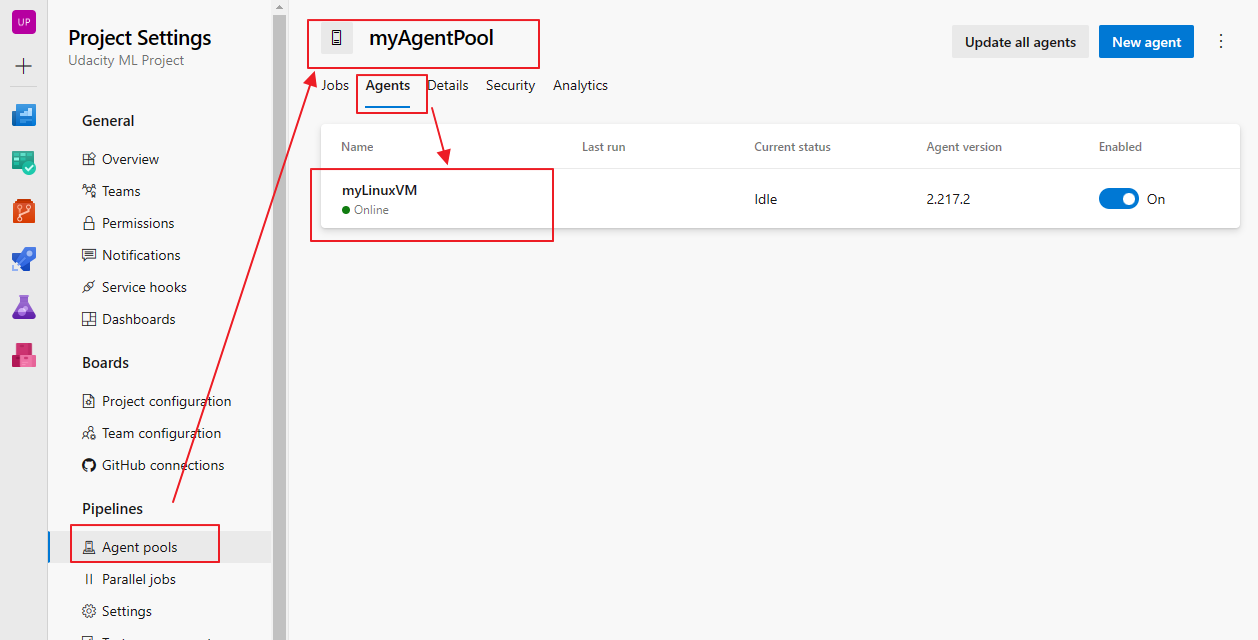
Finally, run the follow specific commands for handling this python application:

sudo apt-get update  
sudo apt update  
sudo apt install software-properties-common  
sudo add-apt-repository ppa:deadsnakes/ppa  
sudo apt install python3.10  
sudo apt-get install libpython3.10  
sudo apt-get install python3.10-venv  
sudo apt-get install python3-pip  
sudo apt-get install python3.10-distutils  
sudo apt-get -y install zip

To overcome pylint problems, run the following:

pip install pylint==2.13.7  
pip show --files pylint  
echo $PATH  
export PATH=$HOME/.local/bin:$PATH  
echo $PATH

Now in Azure DevOps you can navigate to the Agents tab of **myAgentPool** to confirm the newly created agent exists.



Next we will create an Azure deployment pipeline.

## Create an Azure Deployment Pipeline

In Azure DevOps, navigate to **Pipelines** and **Create a Pipeline**

Select GitHub, and it should show you a list of repositories from your GitHub account which we connected earlier. Select the correct repository for this project.

The pipeline file already exists in the project, and will be shown on the next screen.

In the file, modify webAppName to show the name of the web app you created in the *Create a web app using the Azure Portal bash CLI* section.

Graphical user interface, text, application, email

Description automatically generated

Then click **Save and Run**, and the first pipeline job will start automatically.

You may be asked to give permissions during the Deploy stage, so please do this:

Graphical user interface, application

Description automatically generated

Finally, the deployment will complete.

Graphical user interface, text, application, email

Description automatically generated

If you encounter any errors, click on the failed part and inspect any console output.

## CI/CD Pipeline is Complete.

Any time changes are committed to the main branch of the project repository, the pipeline with test and lint the project, the automatically deploy the changes to the live web app.

## Logging

To view a tail log of your web app, run the following in the Azure Portal bash terminal:

az webapp log tail -n <YOUR WEB APP NAME>

This will produce a live tail log of the app while it runs.

For example, here we can see the result of running the script ./make\_predict\_azure\_app.sh from another bash terminal:

Text

Description automatically generated with medium confidence

# Improvements

To improve the project in future, it would be best to set up staging and production environments. The CICD pipeline would automatically deploy to staging, whereas prod should be limited to manual triggering.