**Azure CI/CD Pipeline**

**Overview**

Build a Github repository from scratch and create a scaffolding that will assist in performing both Continuous Integration and Continuous Delivery. You'll use Github Actions along with a Makefile, requirements.txt and application code to perform an initial lint, test, and install cycle.

**Project Plan**

**Trello board**

**<https://trello.com/b/ureV4tWM/udacity-project2-ci-cd>**

**Spreadsheet**

<https://docs.google.com/spreadsheets/d/17mHGmy7t3YEYdNc1xbdvcayJWROOpGoN3YX0FY785fc/edit#gid=1348135932>

**CI: Set Up Azure Cloud Shell**

1. Create the Cloud-Based Development Environment

- Create a GitHub Repo

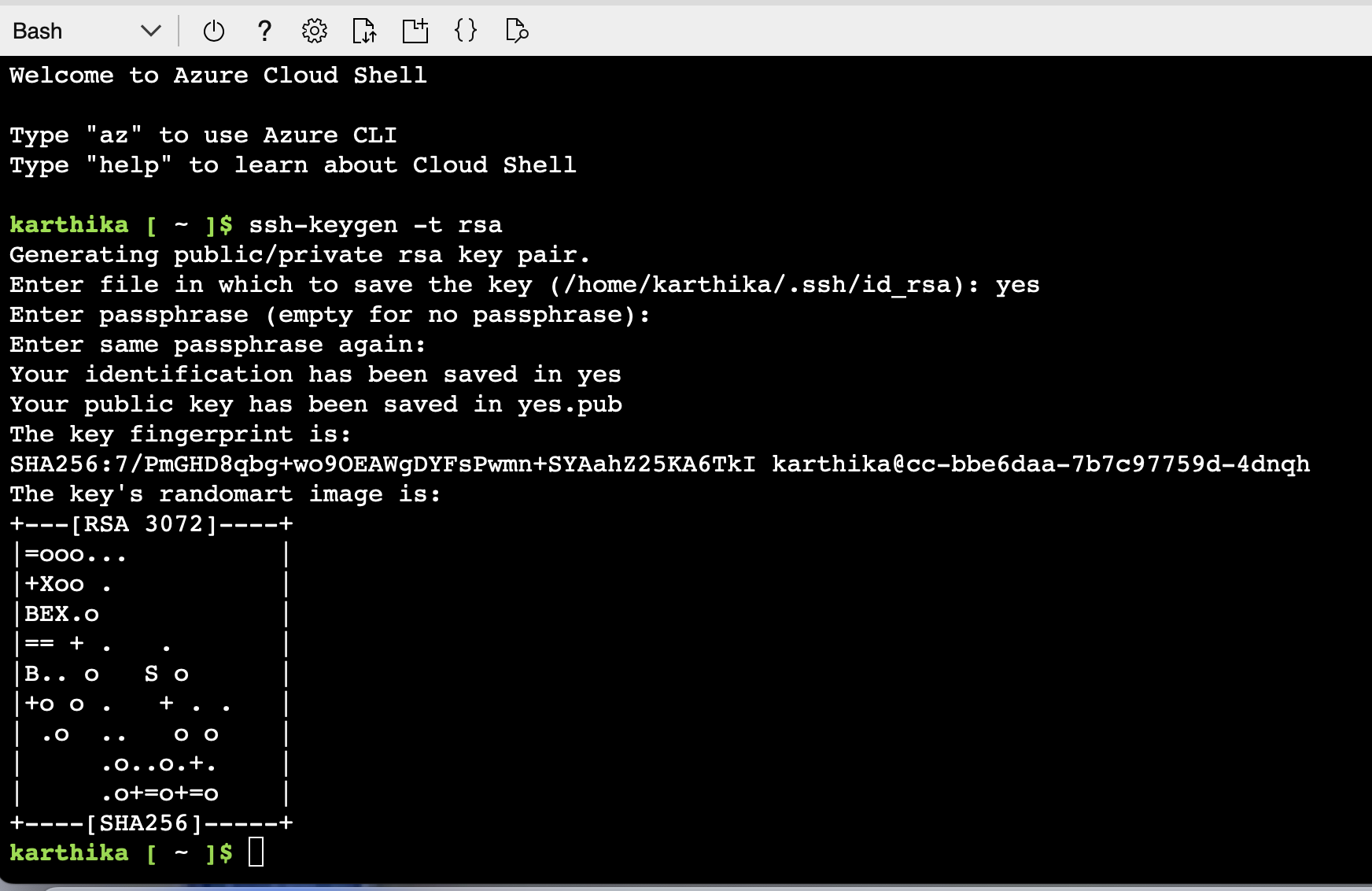
- Launch an Azure Cloud Shell environment and ssh-keys. Upload these to GitHub account

- Take a screenshot showing the project cloned in Azure Cloud Shell

Steps to be followed:

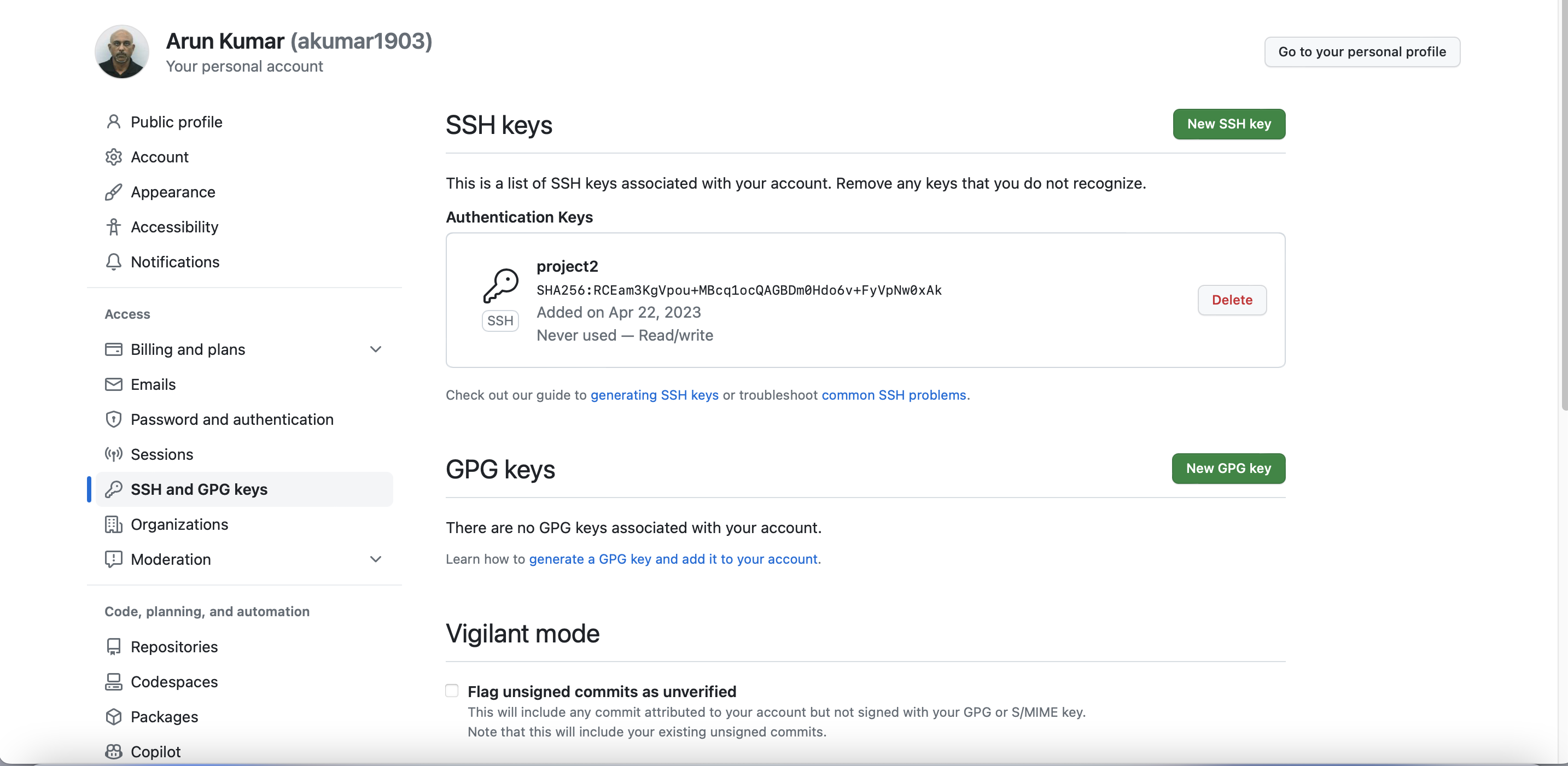
1) From Azure portal open Azure cloud shell

2) Generate key through the command ssh-keygen -t rsa



- Type cat /home/karthika/.ssh/id\_rsa.pub on the Azure Cloud Shell to generate the key

- Copy the key including SSH and open the SSH Keys link in the GitHub repo and copy paste the key



2. Create Project Scaffolding

Now that your environment is set up, you can create the scaffolding for your project and test your code.

Create the Makefile

Inside your GitHub repo, your next step will be to create a file named Makefile and copy the code below into it (remember this needs to use tab formatting). Remember that a Makefile is a handy way to create shortcuts to build, test, and deploy a project.

install:

pip install --upgrade pip &&\

pip install -r requirements.txt

test:

python -m pytest -vv test\_hello.py

lint:

pylint --disable=R,C hello.py

all: install lint test

Create requirements.txt

Inside your GitHub repo, your next step will be to create a file named requirements.txt. It should include the following items below. Remember that a requirements.txt is a convenient way to list what packages a project needs. Another optional best practice would be to "pin" the exact version of the package you use.

pylint

pytest

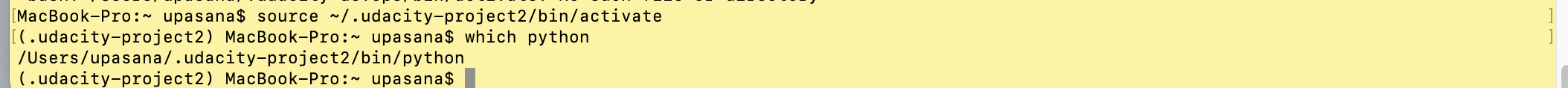
Create the Python Virtual Environment

Create a python virtual environment in local machine Mac. Remember that by creating the virtual environment in a home directory it won't accidentally be checked into your project.

python3 -m venv ~/.udacity-project2

source ~/.udacity-project2/bin/activate

The virtual environment was created inside azure cloud shell and activated



Create the script file and test file.

The next step is to create the script file and test file. This is a boilerplate code to get the initial continuous integration process working. It will later be replaced by the real application code.

First, you will need to create hello.py with the following code at the top level of your Github repo:

def toyou(x):

return "hi %s" % x

def add(x):

return x + 1

def subtract(x):

return x - 1

Next, you will need to createtest\_hello.py with the following code at the top level of your Github repo:

from hello import toyou, add, subtract

def setup\_function(function):

print("Running Setup: %s" % function.\_\_name\_\_)

function.x = 10

def teardown\_function(function):

print("Running Teardown: %s" % function.\_\_name\_\_)

del function.x

### Run to see failed test

#def test\_hello\_add():

# assert add(test\_hello\_add.x) == 12

def test\_hello\_subtract():

assert subtract(test\_hello\_subtract.x) == 9

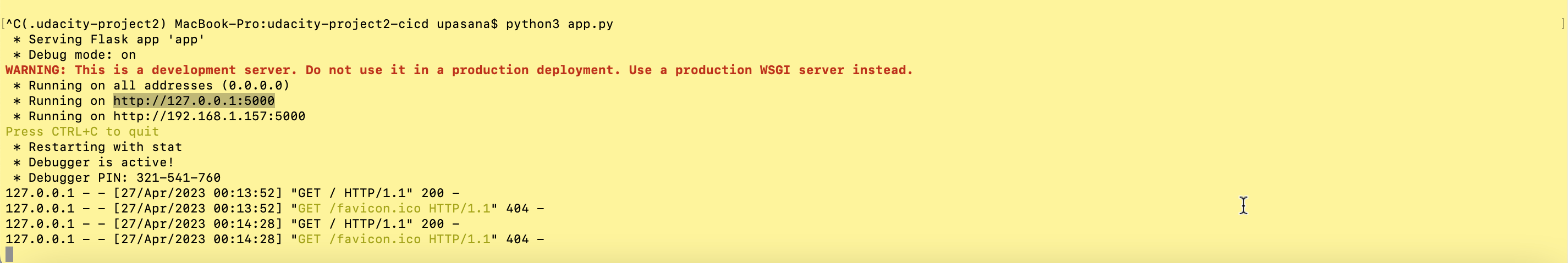
3. Local Test

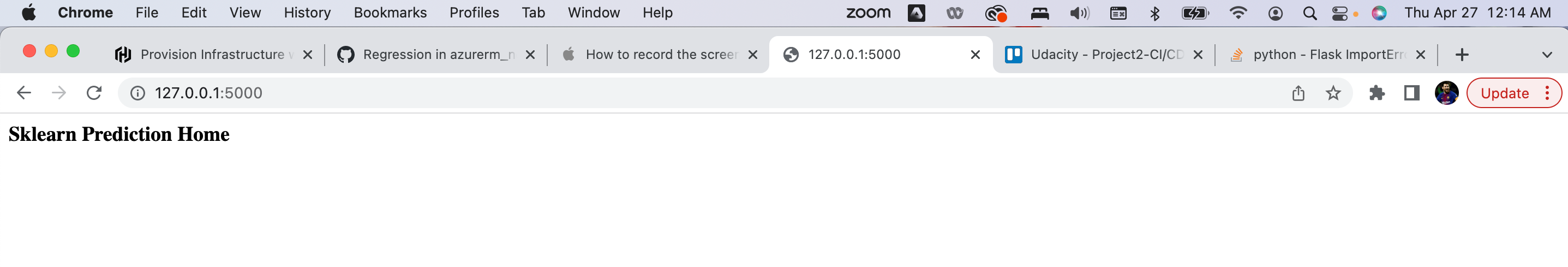
Now it is time to run make all which will install, lint, and test code. This enables us to ensure we don't check in broken code to GitHub as it installs, lints, and tests the code in one command. Later we will have a remote build server perform the same step.

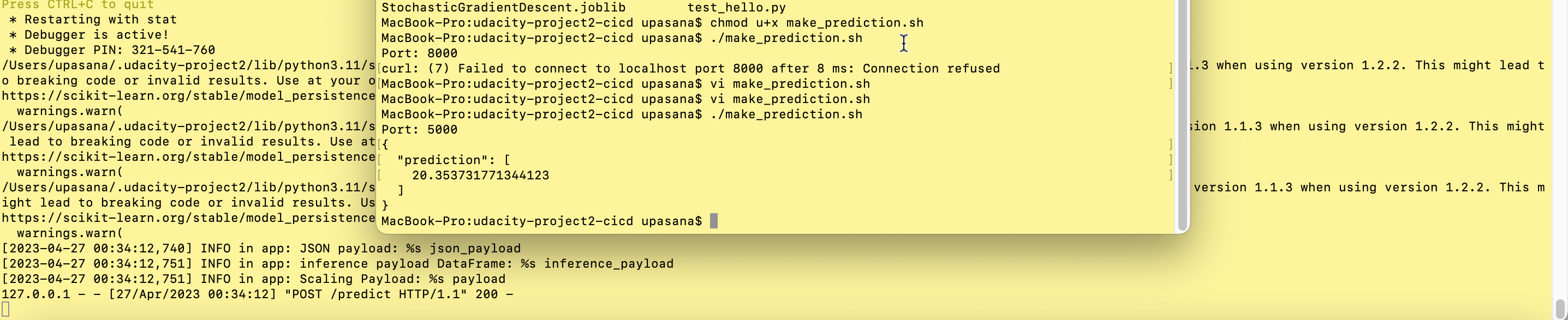


Local test run in the system

Execute app.py to check the predictions and test on the browser for Sklearn prediction







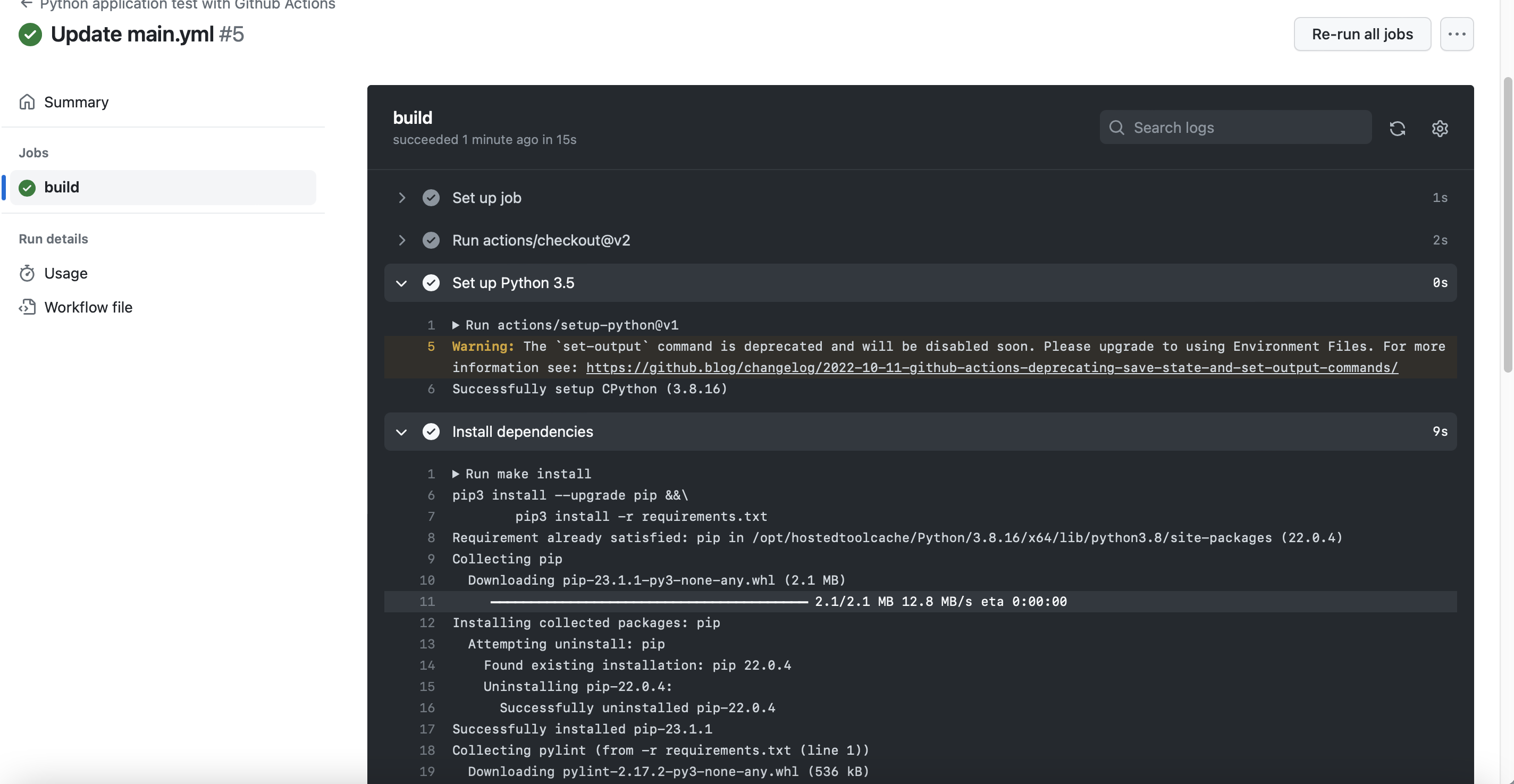
CI: Configure GitHub Actions

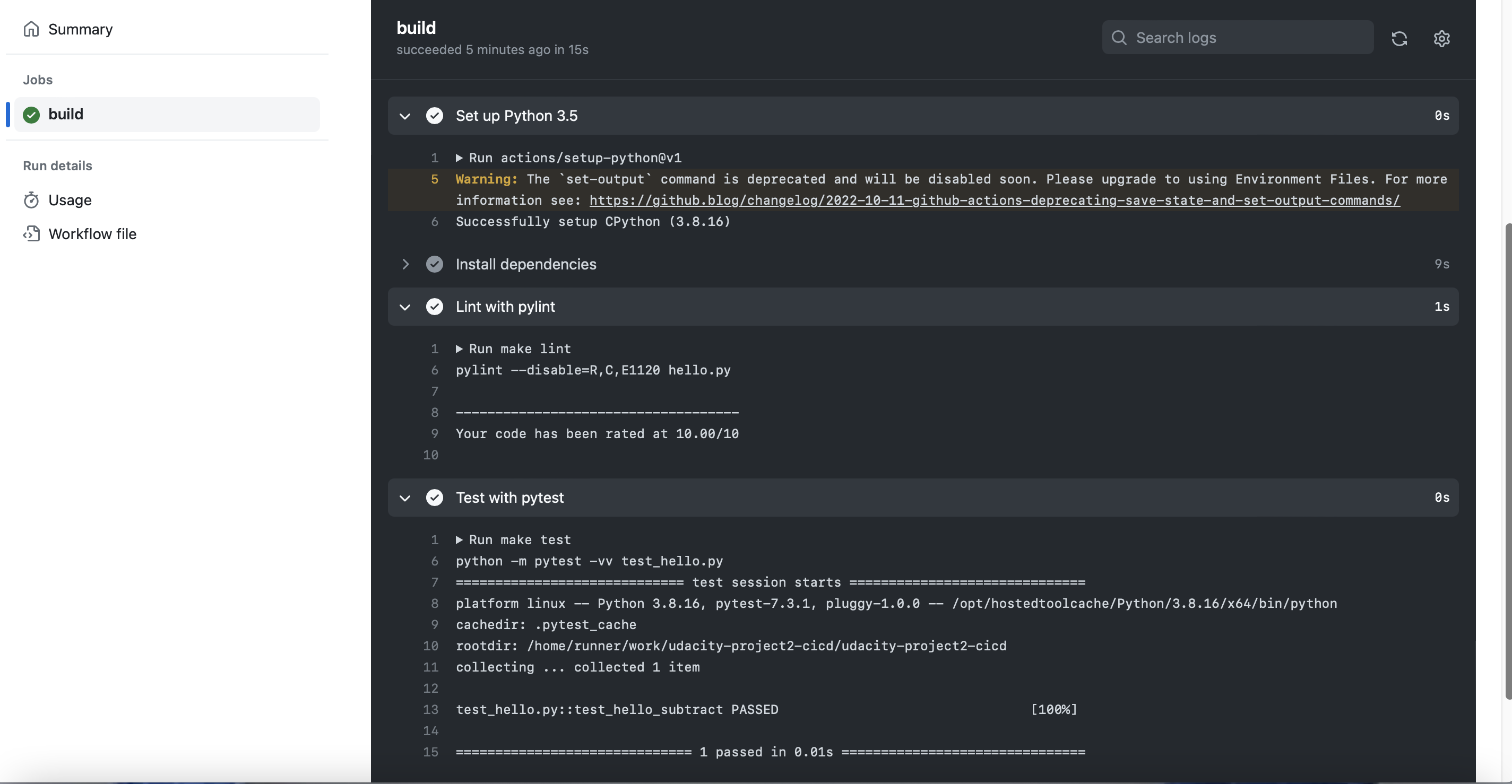
1. Enable Github Actions

Go to your Github Account and enable Github Actions.

2. Replace yml code

Replace the pythonapp.yml code with the following scaffolding code.



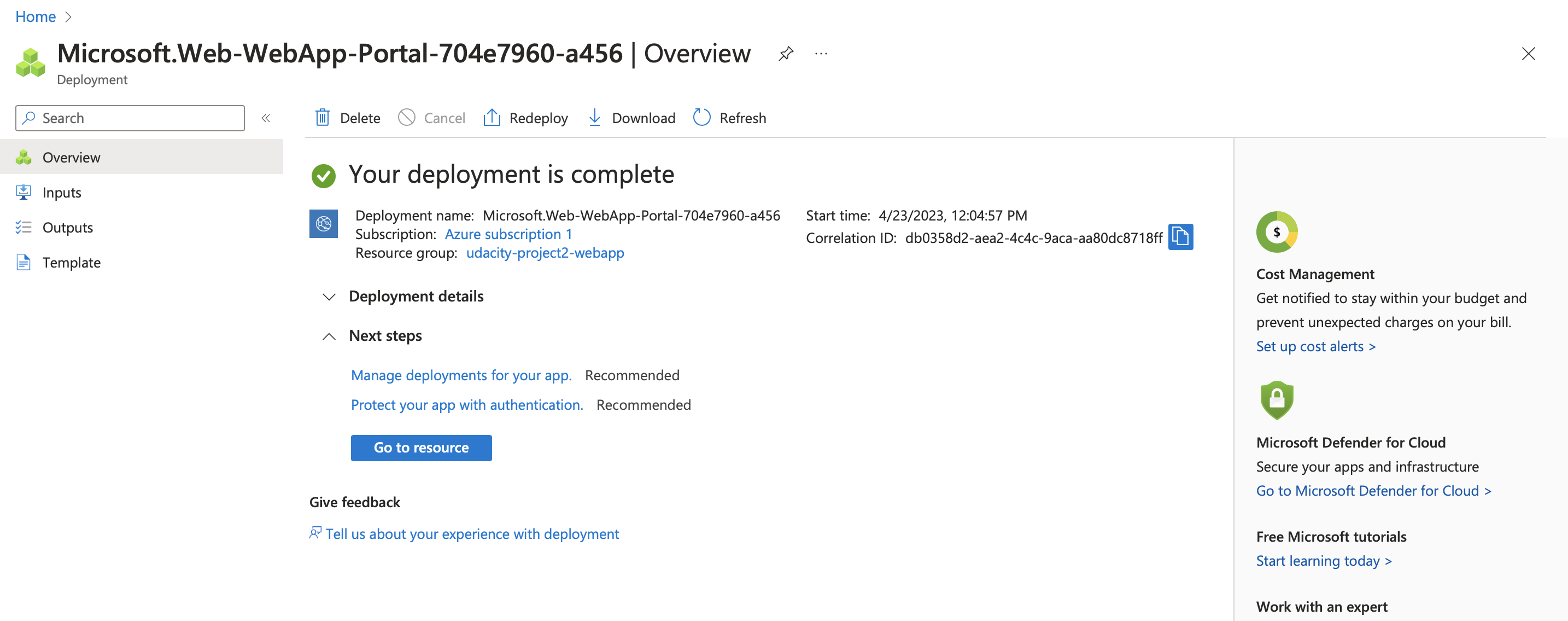


Continuous Delivery on Azure

This will involve setting up Azure Pipelines to deploy the **[Flask starter code](https://github.com/udacity/nd082-Azure-Cloud-DevOps-Starter-Code/tree/master/C2-AgileDevelopmentwithAzure/project/starter_files)** to Azure App Services.

**1. Create Azure App service**

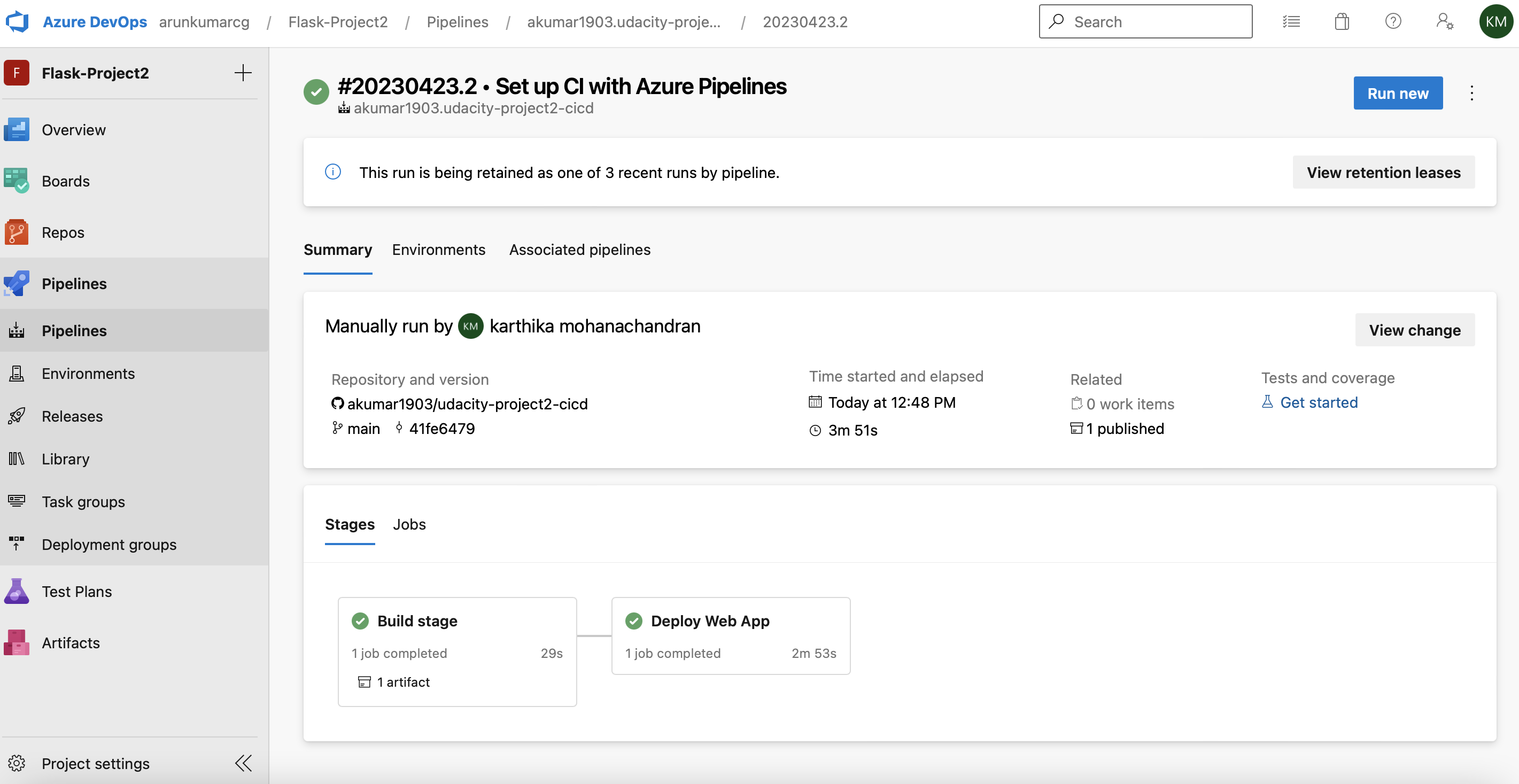
Setting up Azure App Service through Azure Portal



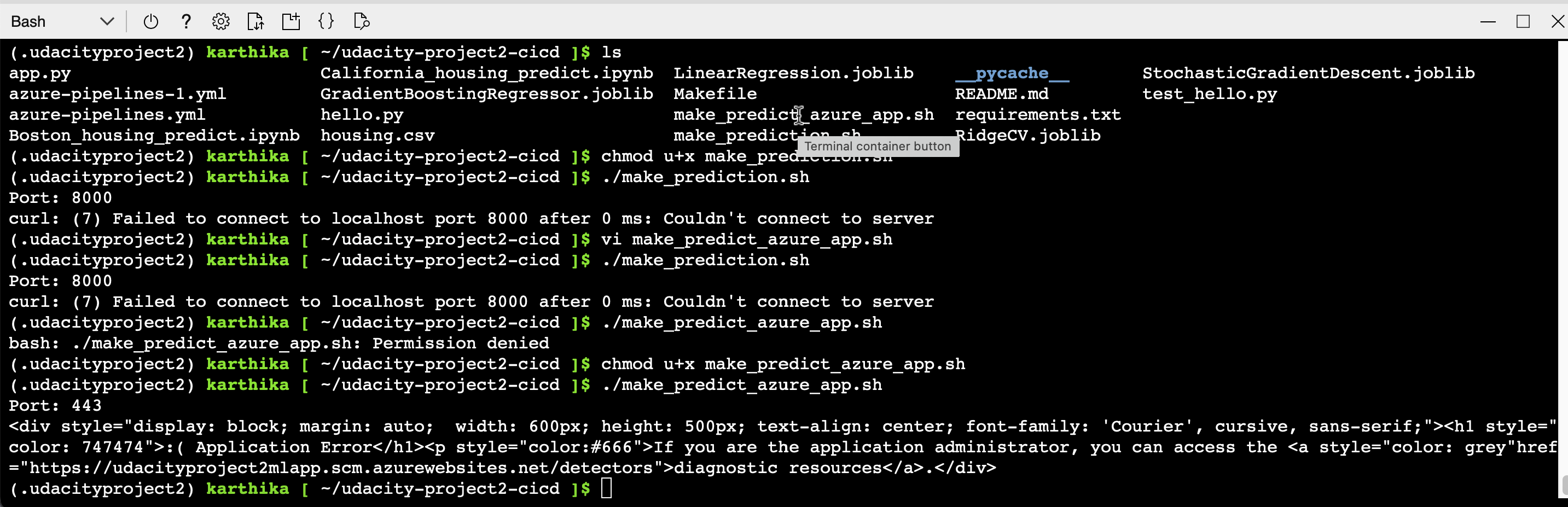
**2. Create CD with Azure Pipelines**

Create Azure Devops project and connect to Azure Pipeline to deploy the flask app

Following the tutorial for steps - <https://learn.microsoft.com/en-us/azure/devops/pipelines/ecosystems/python-webapp?view=azure-devops&WT.mc_id=udacity_learn-wwl>

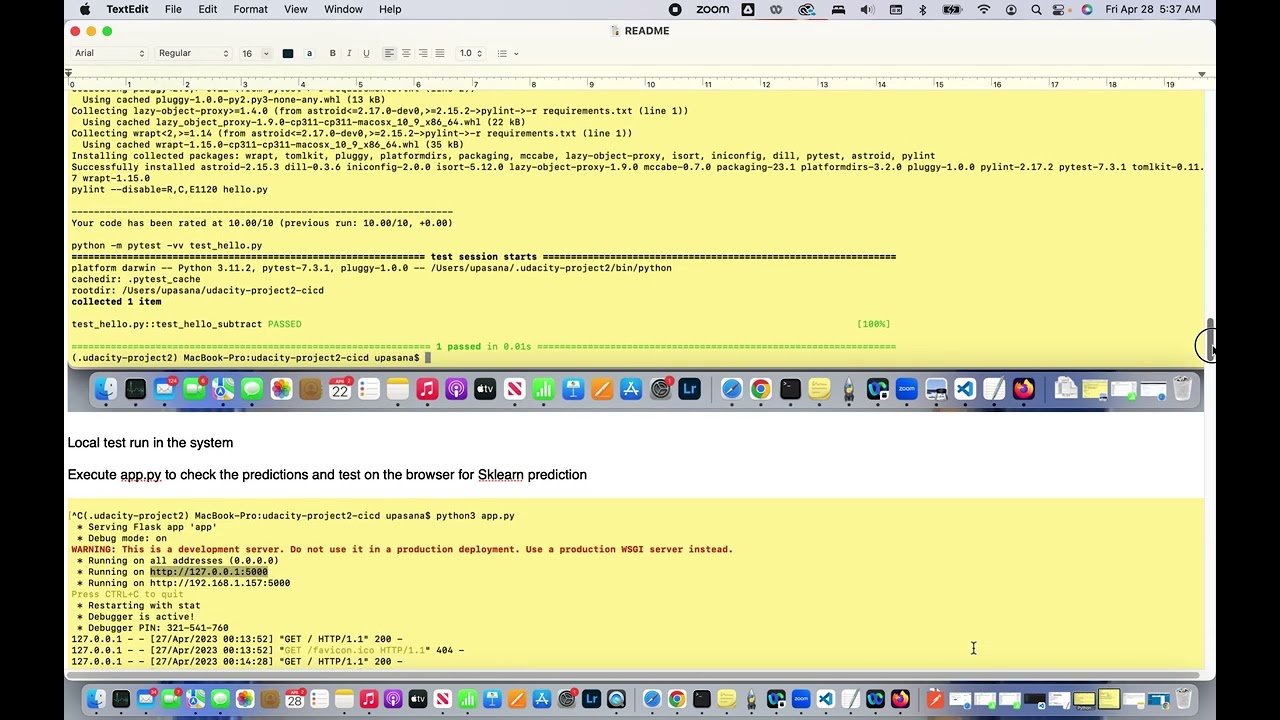


Verify prediction output by calling sh script in cloud shell



Stream logs



Demo Video[](https://www.youtube.com/watch?v=_nbsYe5irJA)