Deploying a Python Web App from a Dev Container in VS Code using GitHub Actions

Overview

The **Visual Studio Code Remote - Containers** extension lets you use a Docker container as a full-featured development environment. It allows you to open any folder inside (or mounted into) a container and use all of VS Code's features like IntelliSense, code navigation, and debugging.

GitHub Actions gives you the flexibility to build an automated software development lifecycle workflow. You can write individual tasks ("Actions") and combine them to create a custom workflow. Workflows are configurable automated processes that you can set up in your repository to build, test, package, release, or deploy any project on GitHub.

With **GitHub Actions** you can build end-to-end continuous integration (CI) and continuous deployment (CD) capabilities directly in your repository.

What’s covered in this lab

In this lab, you will:

1. Work on a Python web app inside the Remote-Containers extension in VS Code
2. Deploy the web app to Azure using the App Service extension
3. Create a workflow with GitHub Actions to add CI/CD to your app

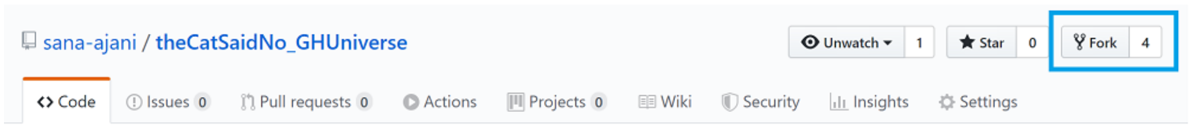
Prerequisites

1. Your Windows machine should have Python 3.7, Docker, and Visual Studio Code, and the VS Code Remote Development extensions installed.
2. You are using a GitHub account and an Azure account made for the purposes of this lab. These have been already logged into from your machine and the account info is saved.

Setting up the GitHub repo

"The Cat Said No" is a simple Python Flask web app.

1. Go to https://github.com/sana-ajani/theCatSaidNo_GHUniverse. Click the "Fork" button in the upper-right hand corner. From there, click the green "Clone" button and copy the URL.



2. Open up the Windows Terminal and run the following command, pasting in the link you just copied.

```
git clone <pasted URL>
```

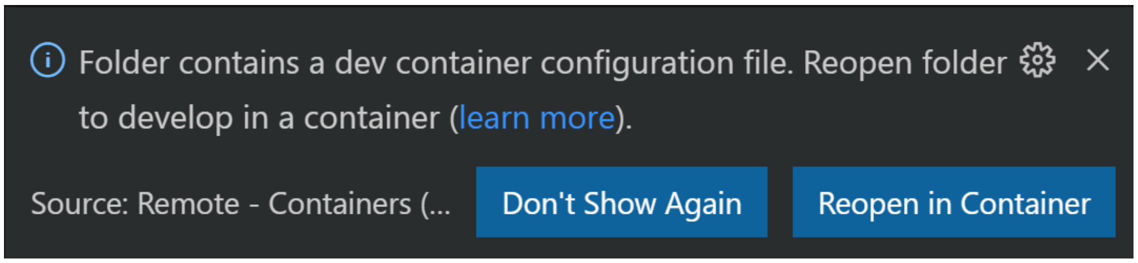
3. Open the repo in Visual Studio Code.

```
cd theCatSaidNo_GHUniverse

code .
```

Open the dev container workspace

1. Notice the repo has a `.devcontainer` folder which contains a `Dockerfile` and a `devcontainer.json`. The dev container tells VS Code how to create a development container that has a specific runtime, extensions, and tools. In this case, the dev container is Python specific and tells VS Code to install the Python and Azure App Service extensions.
2. Click the `Reopen in Container` prompt, or press `F1` and select the `Reopen folder in dev container` command.

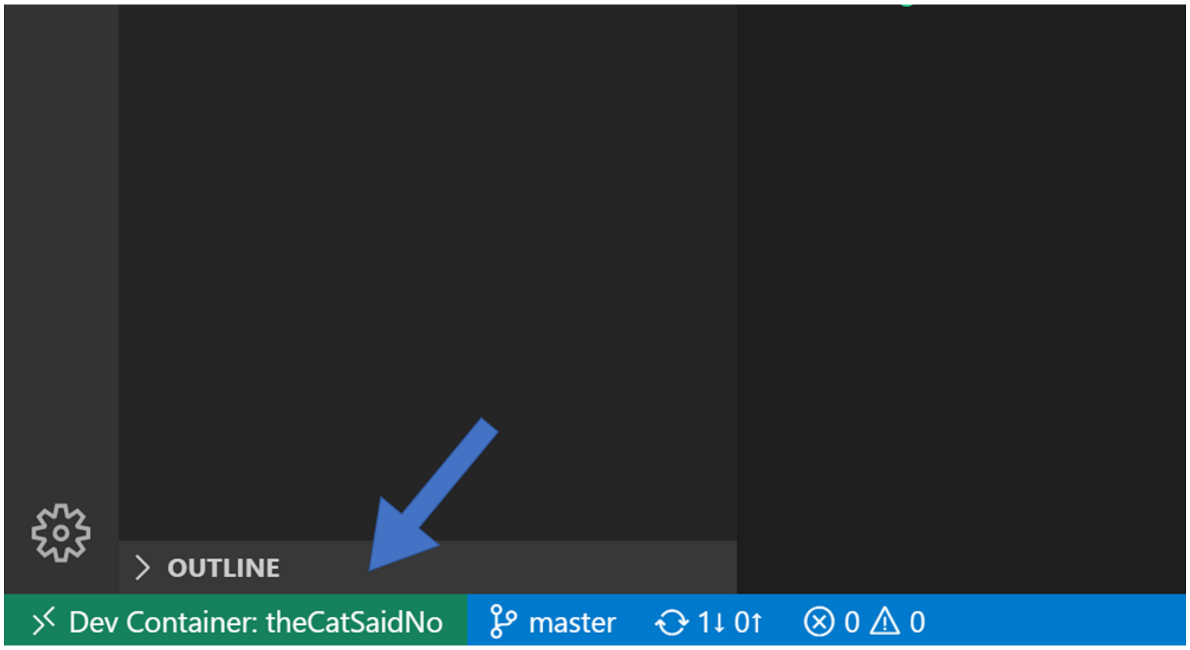


Folder contains a dev container configuration file. Reopen folder to develop in a container (learn more).

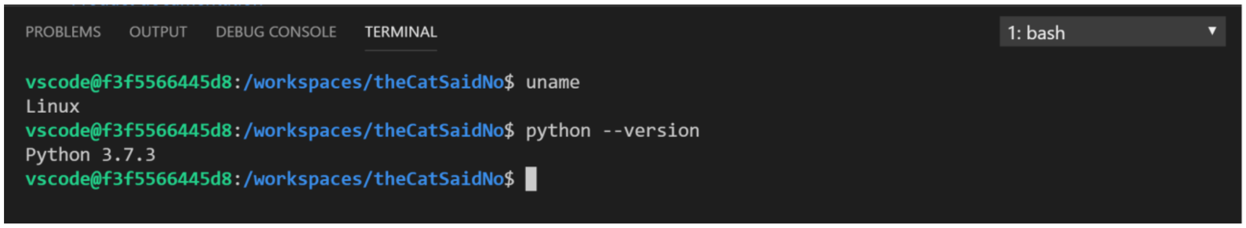
3. Click on "Details" to see what is happening when VS Code is creating the container. VS Code is installing a component called "VS Code Server" in the container so you

can directly interact with code, the file system, and extensions in the remote workspace. Since this is the first time we are creating it, it'll take a few minutes (but the next time you reconnect to an existing container will be pretty quick).

4. Notice the indicator in the bottom left corner tells us we are inside our dev container.



5. Press `F1` and run the command "Open new terminal". Once you are in the new terminal instance, notice we're actually in Bash! Type the command **`uname`** to see that we're in a Linux environment right now. Run the command **`python --version`** to check that the version of Python in our remote container is different than the one that is on our local machine.



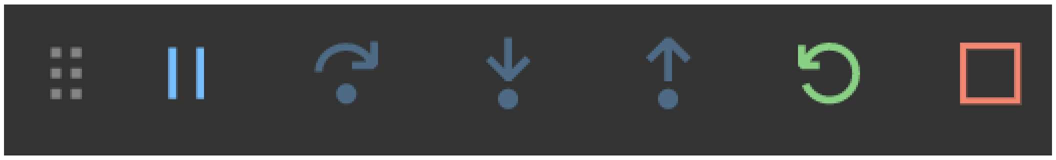
```
vscode@f3f5566445d8:/workspaces/theCatSaidNo$ uname
Linux
vscode@f3f5566445d8:/workspaces/theCatSaidNo$ python --version
Python 3.7.3
vscode@f3f5566445d8:/workspaces/theCatSaidNo$
```

6. Press `F5` to run the app inside the container
7. Open up Edge and go to `localhost:9000` and try out the app!

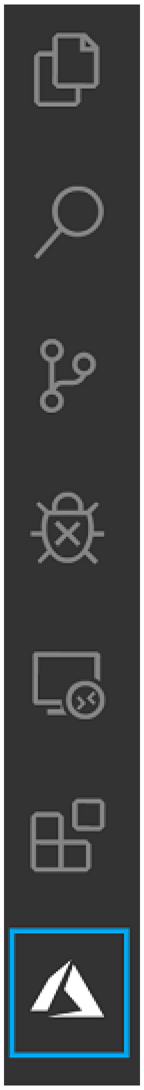
Create an Azure App Service

Instead of running this locally, let's create this as a web app hosted in Azure.

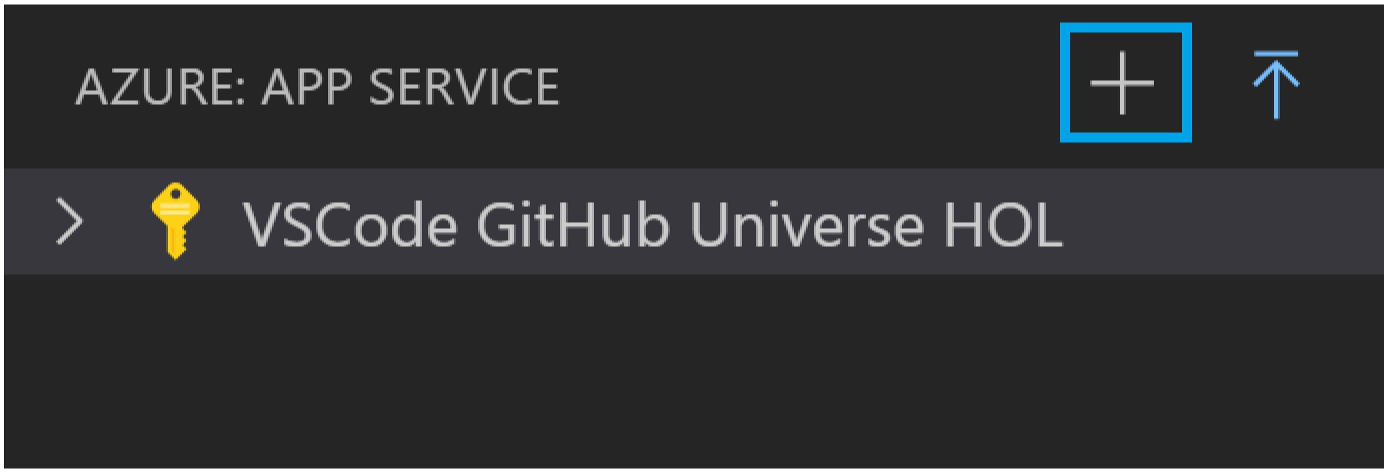
1. Stop the app running locally by clicking on the red square in the debug toolbar.



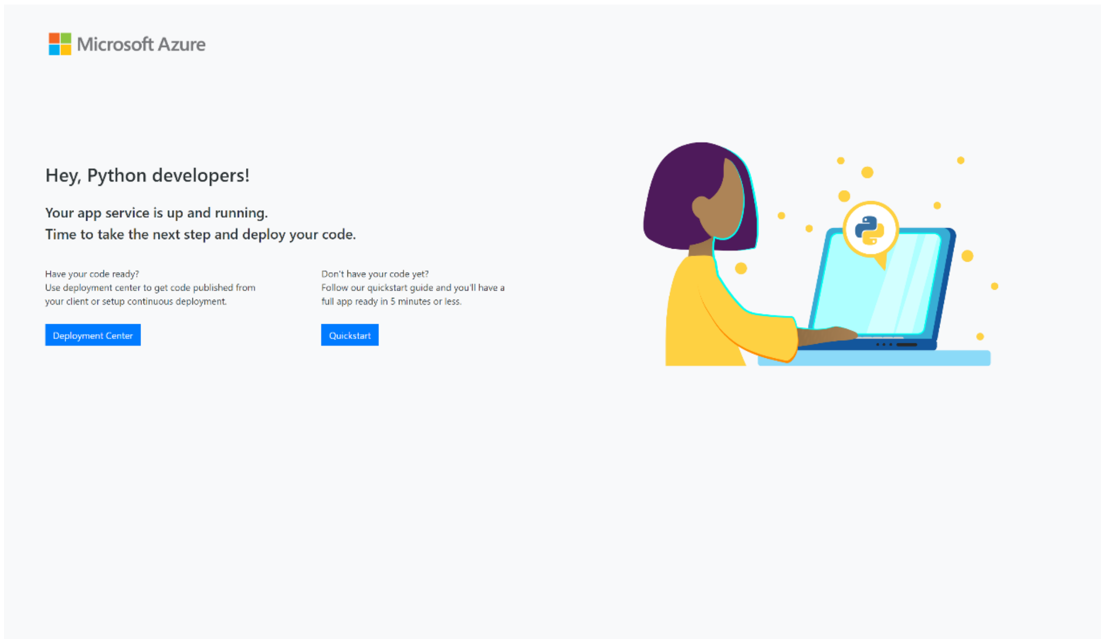
2. Click on the Azure icon in the sidebar.



3. Click on the + icon to create a new app service under the **VSCode GitHub Universe HOL** subscription.



4. Give your webapp a unique name (we recommend calling it **YOUR_NAME-theCatSaidNo**)
5. Select **Linux** as your OS and **Python 3.7** as your runtime.
6. It will take a minute or two to create the app. Once it's done, you'll get a prompt to browse to your new site. Click on "View output" and open the link to your site.
7. The page you browse to will be the default site you see, since we haven't yet deployed anything to the site.



Hey, Python developers!

Your app service is up and running.
Time to take the next step and deploy your code.

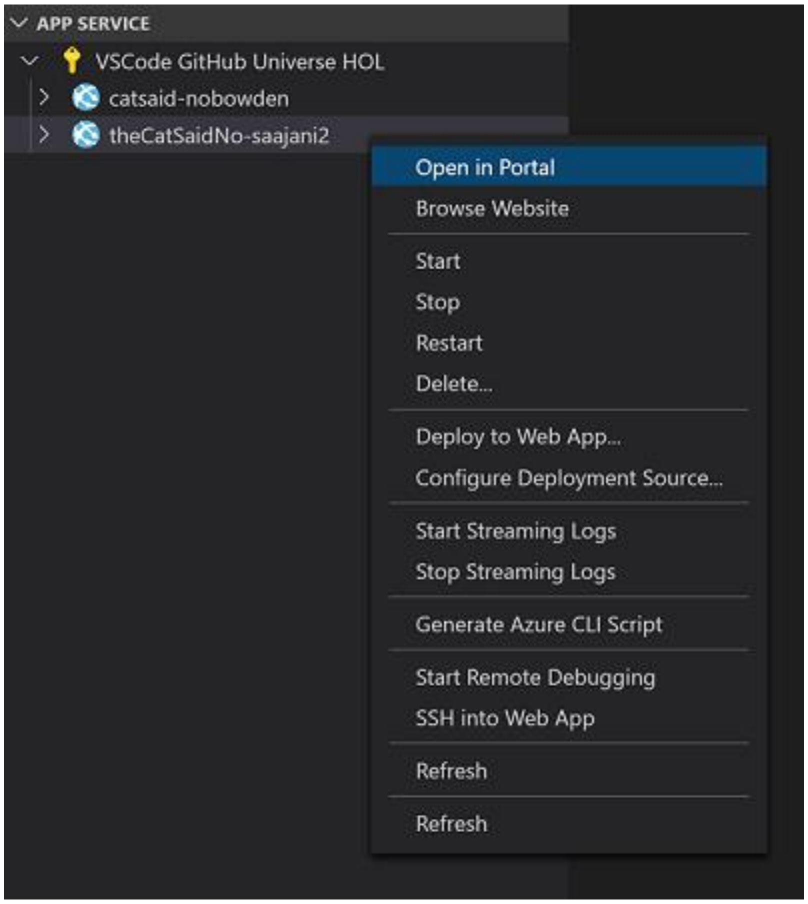
Have your code ready?
Use deployment center to get code published from your client or setup continuous deployment.

Don't have your code yet?
Follow our quickstart guide and you'll have a full app ready in 5 minutes or less.

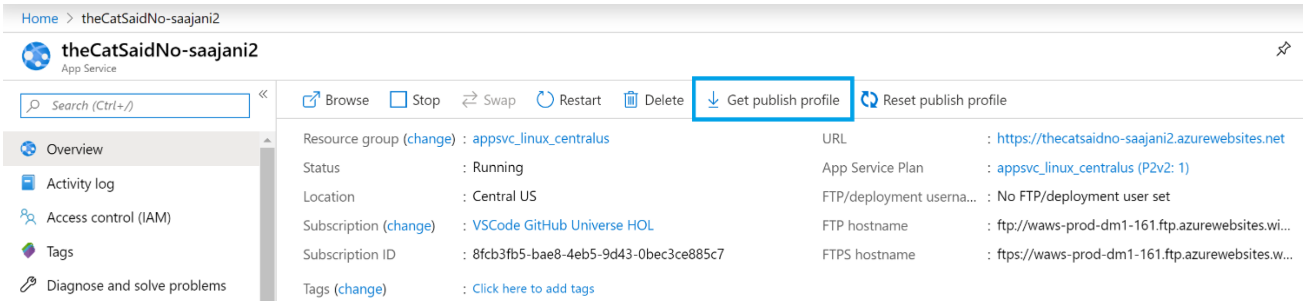
Set up CI/CD with GitHub Actions

We'll use GitHub actions to automate our deployment workflow for this web app.

1. Inside the App Service extension, right click on the name of your app service and choose "Open in Portal".



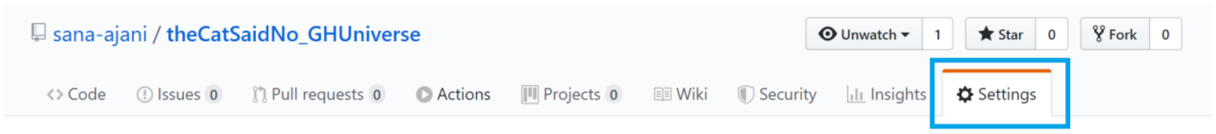
2. From the Overview page, click on "Get publish profile". A publish profile is a kind of deployment credential, useful when you don't own the Azure subscription.



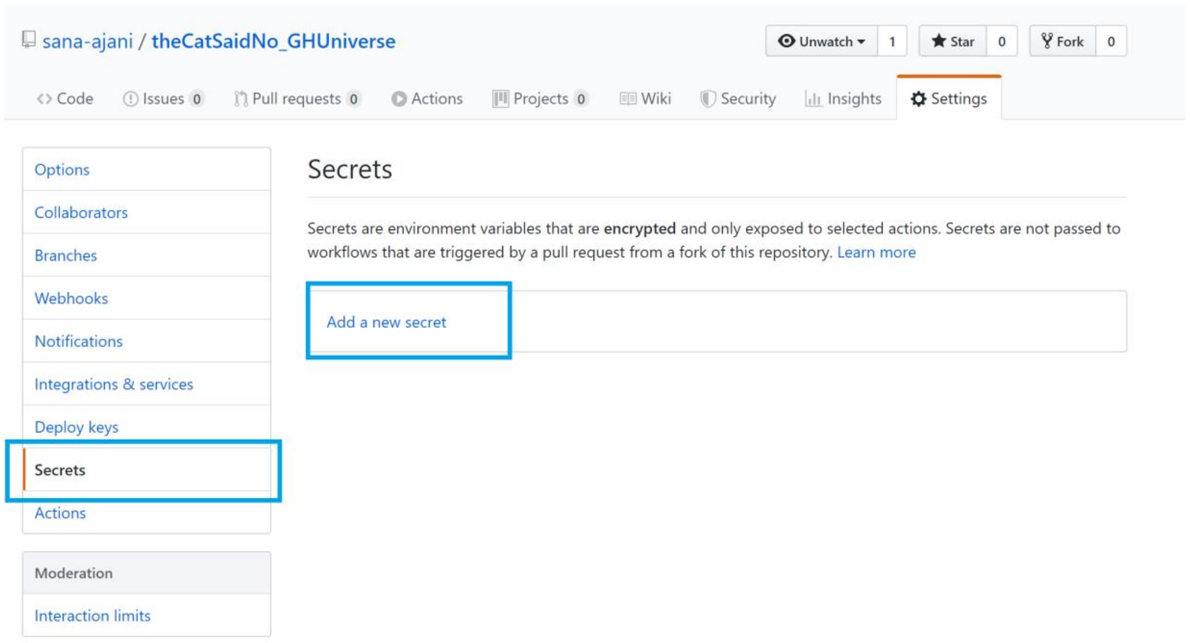
Resource group (change)	: appsvc_linux_centralus	URL	: https://thecatsaidno-saajani2.azurewebsites.net
Status	: Running	App Service Plan	: appsvc_linux_centralus (P2v2: 1)
Location	: Central US	FTP/deployment userna...	: No FTP/deployment user set
Subscription (change)	: VSCode GitHub Universe HOL	FTP hostname	: ftp://waws-prod-dm1-161.ftp.azurewebsites.wi...
Subscription ID	: 8fcb3fb5-bae8-4eb5-9d43-0bec3ce885c7	FTPS hostname	: ftps://waws-prod-dm1-161.ftp.azurewebsites.w...
Tags (change)	: Click here to add tags		

3. Open the settings file you just downloaded in VS Code and copy the contents of the file.

4. We will now add the publish profile as a secret associated with this repo. On the GitHub repository, click on the "Settings" tab.

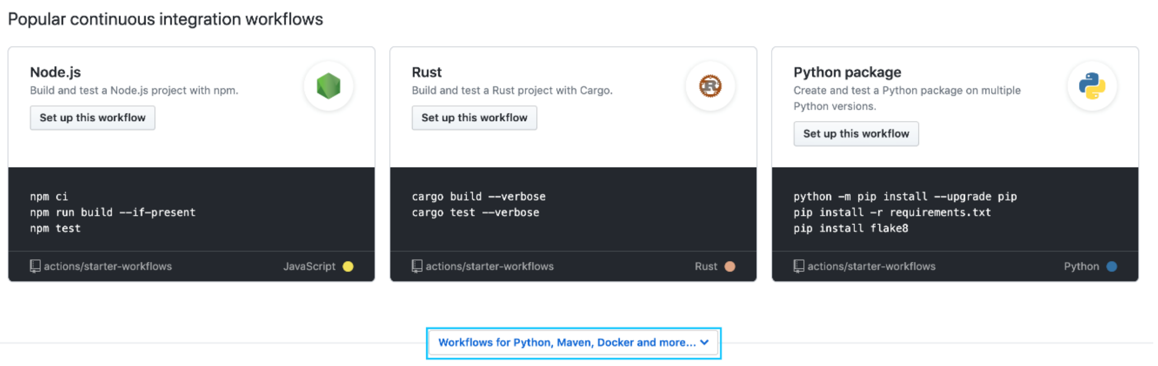


5. Go to "Secrets". Create a new secret and call it "LAB_PUBLISH_PROFILE". Paste the contents from the settings file.



Secrets are environment variables that are **encrypted** and only exposed to selected actions. Secrets are not passed to workflows that are triggered by a pull request from a fork of this repository. Learn more

6. Now click on "Actions" in the top bar. Under the "Popular continuous integration workflows" section, click on the "Workflows for Python, Maven, Docker, and more" button to open up more templates.


Popular continuous integration workflows
Node.js
Set up this workflow
JavaScript
Rust
Set up this workflow
Rust
Python package
Set up this workflow
Python
actions/starter-workflows
actions/starter-workflows
actions/starter-workflows
Workflows for Python, Maven, Docker and more...

Node.js: Build and test a Node.js project with npm.

```
npm ci
npm run build --if-present
npm test
```

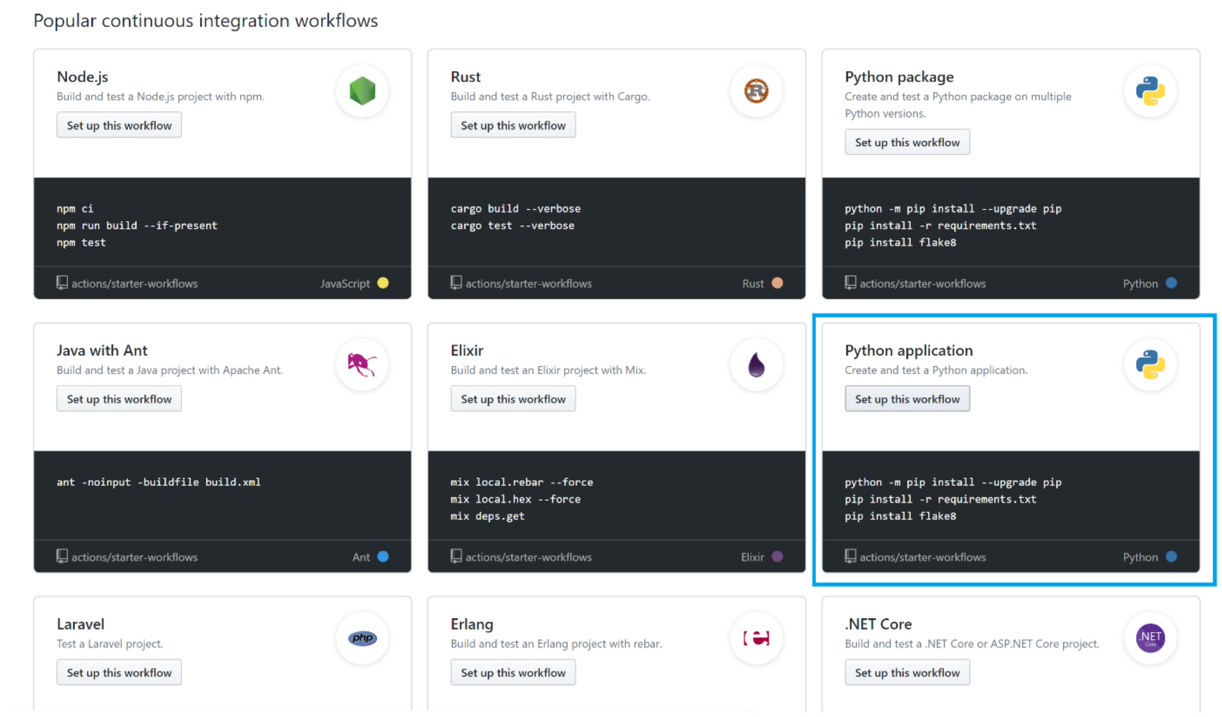
Rust: Build and test a Rust project with Cargo.

```
cargo build --verbose
cargo test --verbose
```

Python package: Create and test a Python package on multiple Python versions.

```
python -m pip install --upgrade pip
pip install -r requirements.txt
pip install flake8
```

7. Find the **Python application** template (not the Python Package one!) and select "Set up this workflow".


Popular continuous integration workflows

Node.js
Build and test a Node.js project with npm.
Set up this workflow

```
npm ci
npm run build --if-present
npm test
```

actions/starter-workflows JavaScript

Rust
Build and test a Rust project with Cargo.
Set up this workflow

```
cargo build --verbose
cargo test --verbose
```

actions/starter-workflows Rust

Python package
Create and test a Python package on multiple Python versions.
Set up this workflow

```
python -m pip install --upgrade pip
pip install -r requirements.txt
pip install flake8
```

actions/starter-workflows Python

Java with Ant
Build and test a Java project with Apache Ant.
Set up this workflow

```
ant -noinput -buildfile build.xml
```

actions/starter-workflows Ant

Elixir
Build and test an Elixir project with Mix.
Set up this workflow

```
mix local.rebar --force
mix local.hex --force
mix deps.get
```

actions/starter-workflows Elixir

Python application
Create and test a Python application.
Set up this workflow

```
python -m pip install --upgrade pip
pip install -r requirements.txt
pip install flake8
```

actions/starter-workflows Python

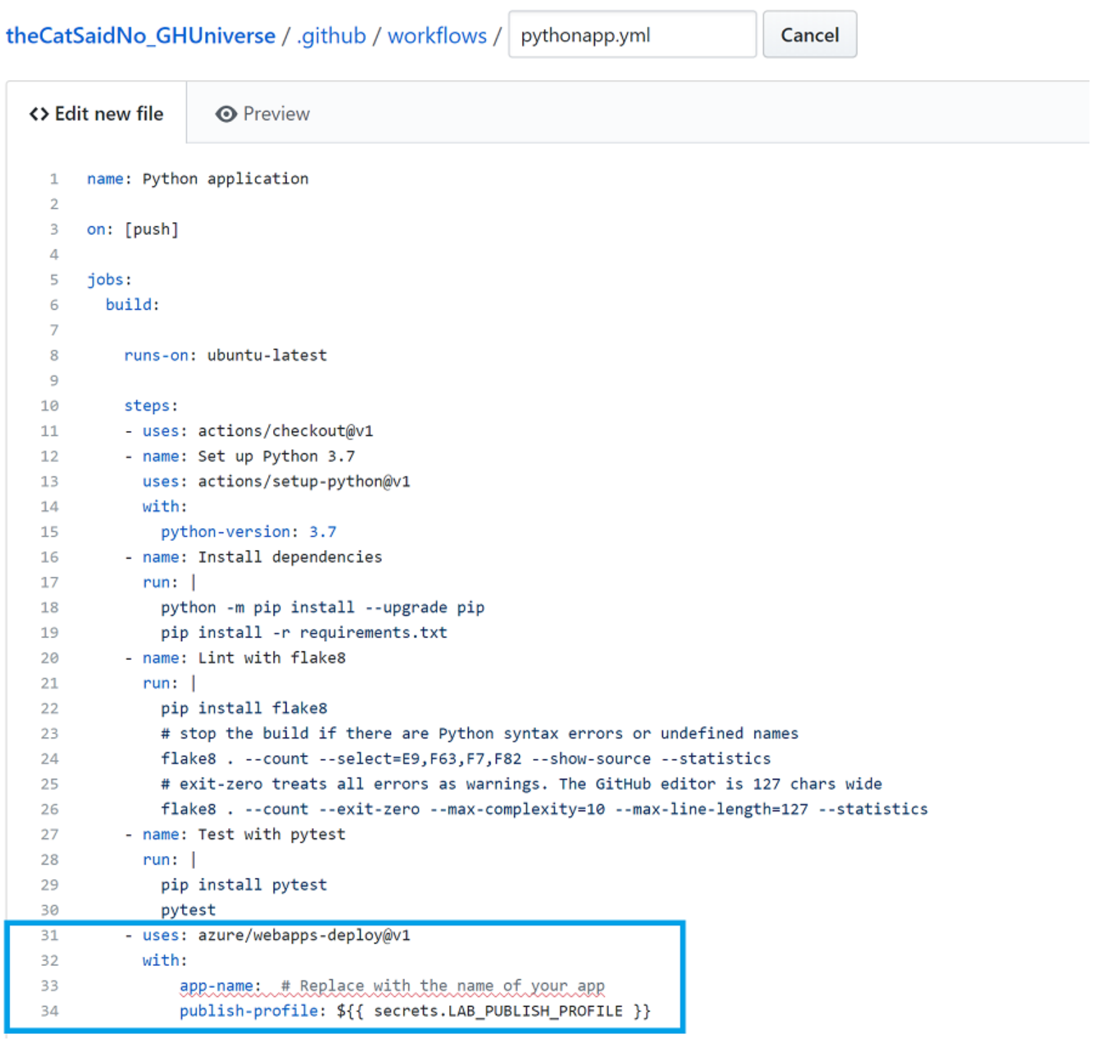
Laravel
Test a Laravel project.
Set up this workflow

Erlang
Build and test an Erlang project with rebar.
Set up this workflow

.NET Core
Build and test a .NET Core or ASP.NET Core project.
Set up this workflow

8. Now, paste these lines of code to the end of the `pythonapp.yml` file in GitHub. Change the `app-name` to the name of your web app. Here, we are using GitHub Azure Actions to login to Azure with the publish profile we stored in GitHub secrets previously.

```
- uses: azure/webapps-deploy@v1
  with:
      app-name:  # Replace with your app name
      publish-profile: ${{ secrets.LAB_PUBLISH_PROFILE }}
```


theCatSaidNo_GHUniverse / .github / workflows / pythonapp.yml
Cancel
<> Edit new file
Preview

```
1  name: Python application
2
3  on: [push]
4
5  jobs:
6    build:
7
8      runs-on: ubuntu-latest
9
10     steps:
11     - uses: actions/checkout@v1
12     - name: Set up Python 3.7
13       uses: actions/setup-python@v1
14       with:
15         python-version: 3.7
16     - name: Install dependencies
17       run: |
18         python -m pip install --upgrade pip
19         pip install -r requirements.txt
20     - name: Lint with flake8
21       run: |
22         pip install flake8
23         # stop the build if there are Python syntax errors or undefined names
24         flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
25         # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
26         flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
27     - name: Test with pytest
28       run: |
29         pip install pytest
30         pytest
31     - uses: azure/webapps-deploy@v1
32       with:
33           app-name:  # Replace with the name of your app
34           publish-profile: ${{ secrets.LAB_PUBLISH_PROFILE }}
```

9. Once you're done, click on "Start commit". Fill in the text box with a commit message, and then click the "Commit change" button, which will trigger the workflow.

10. While the Action is being queued, let's get into the details of what this workflow is actually doing. Go to the `.github/workflows/pythonapp.yml` file to follow along.

    - **Workflow Triggers (line 3)**: Your workflow is set up to run on "push" events to the branch

      ```
      on: [push]
      ```

      For more information, see Events that trigger workflows.

    - **Running your jobs on hosted runners (line 8):** GitHub Actions provides hosted runners for Linux, Windows, and macOS. We specify the hosted runner in our workflow as below.

```
jobs:
 build:
 runs-on: ubuntu-latest
```

- **Using an action (line 11)**: Actions are reusable units of code that can be built and distributed by anyone on GitHub. To use an action, you must specify the repository that contains the action. We are also specifying the version of Python runtime.

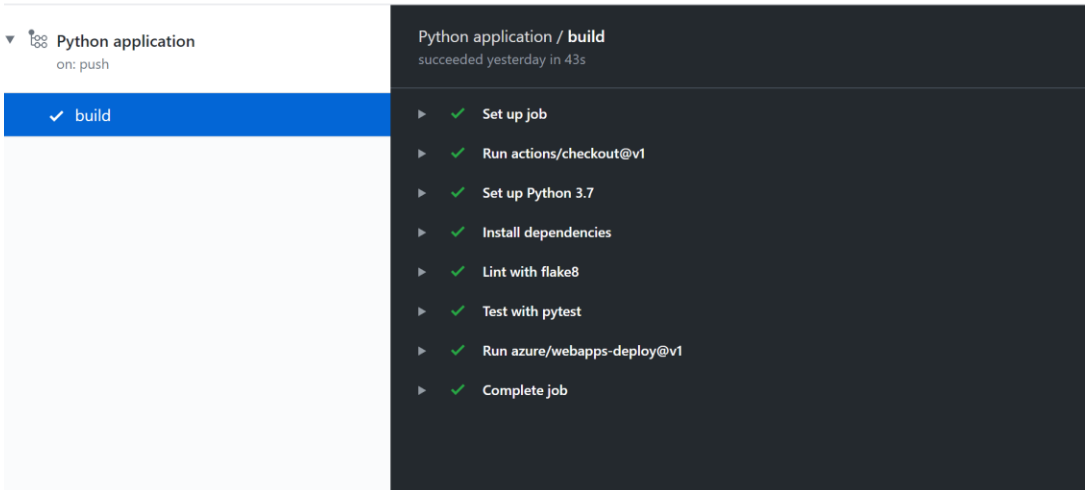
```
- uses: actions/checkout@v1
 - name: Set up Python 3.7
   uses: actions/setup-python@v1
   with:
      python-version: 3.7
```

- **Running a command (line 16)**: You can run commands on the job's virtual machine. We are running the Python commands below to install the dependencies from our requirements.txt, and lint and test our application.

```
  - name: Install dependencies
  run:
      python -m pip install --upgrade pip
      pip install -r requirements.txt
  - name: Lint with flake8
   run:
      pip install flake8
      # stop the build if there are Python syntax errors or undefined 
names
      flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
      # exit-zero treats all errors as warnings. The GitHub editor is 127 
chars wide
      flake8 . --count --exit-zero --max-complexity=10 --max-line-
length=127 --statistics
  - name: Test with pytest
   run:
      pip install pytest
      pytest
```

For workflow syntax for GitHub Actions see here

11. You can go back to the Actions tab, click on your workflow, and see that the workflow is queued or being deployed. Wait for the job to complete successfully before going back to your website.


Python application
on: push
build

Python application / build
succeeded yesterday in 43s

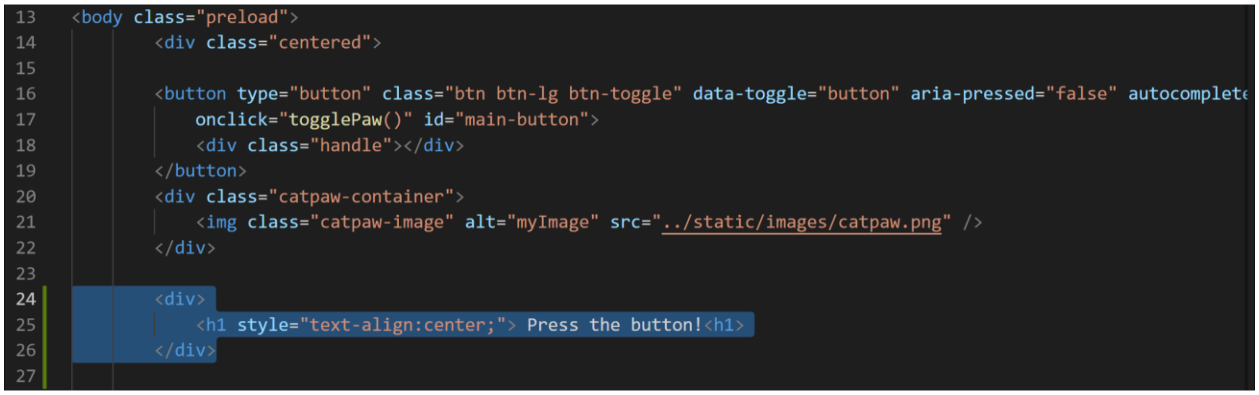
```
✓ Set up job
✓ Run actions/checkout@v1
✓ Set up Python 3.7
✓ Install dependencies
✓ Lint with flake8
✓ Test with pytest
✓ Run azure/webapps-deploy@v1
✓ Complete job
```

Test out your app!

1. Back in VS Code, go to the App Service extension, and right click on your app service and click on "Browse Website".

2. Let's test our GitHub Actions workflow we just made. Add the following lines of code to `templates/home.html` in the body class, after we load in the catpaw image:

```
<div>
    <h1 style="text-align:center;"> Press the button!<h1>
</div>
```



```
<body class="preload">
        <div class="centered">

        <button type="button" class="btn btn-lg btn-toggle" data-toggle="button" aria-pressed="false" autocomplete
            onclick="togglePaw()" id="main-button">
            <div class="handle"></div>
        </button>
        <div class="catpaw-container">
            <img class="catpaw-image" alt="myImage" src="../static/images/catpaw.png" />
        </div>

        <div>
            <h1 style="text-align:center;"> Press the button!<h1>
        </div>

```

3. In the terminal, run the following commands:

```
git add .
git commit -m "test ci/cd"
git push
```

4. Go back to the Actions tab and you can watch the build finishing up. Once you see all the green check marks, go to Edge and browse to your new website!