DEVOPS Bootcamp

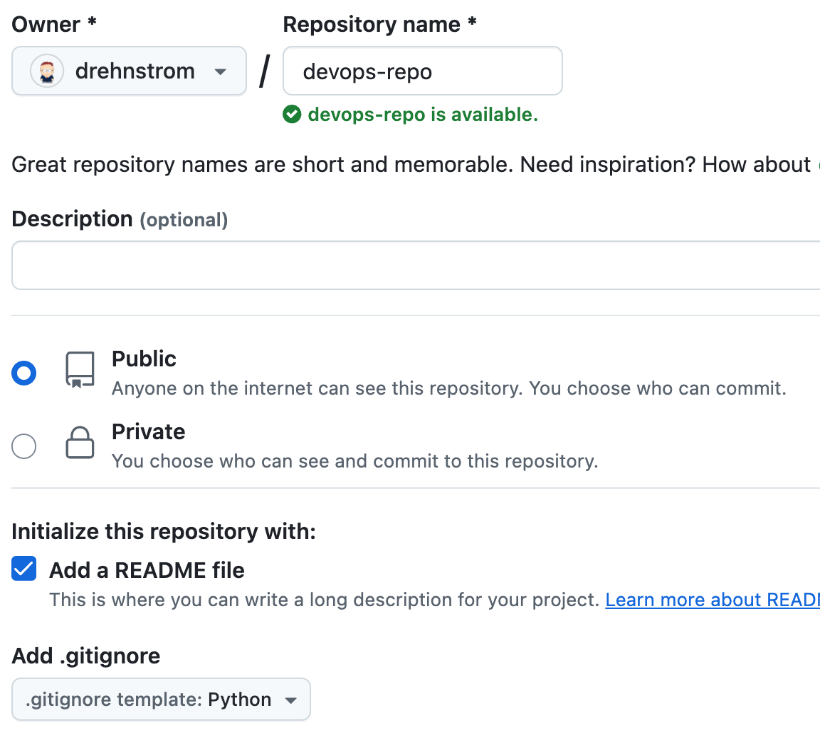
Session 1 Lab: Creating DevOps Pipelines with GitHub Actions

Creating a GitHub Repository

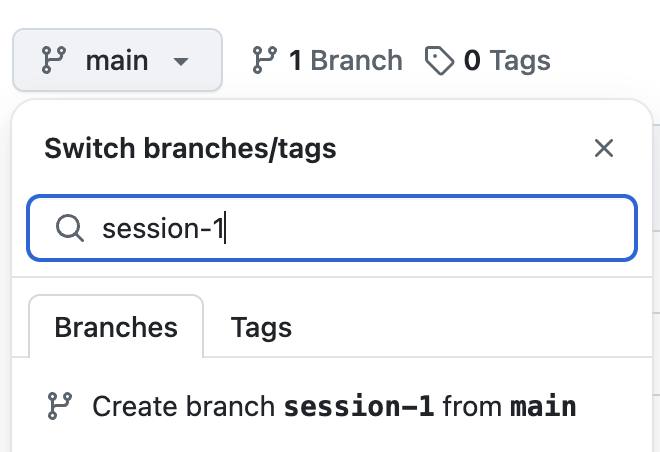
1. Go to <https://github.com> and click the **Sign in** button. If you already have a GitHub account, log in. If you don’t have an account, click the **Create an account** link and follow the instructions.
2. Click the **New** button to create a new repository.



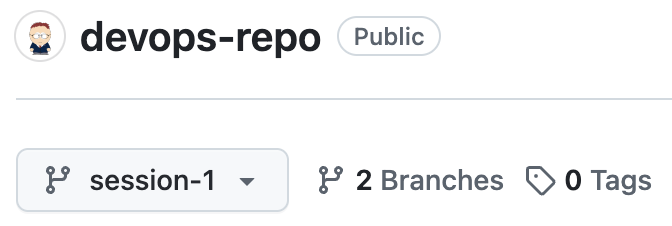
1. Name the repository anything you like. Make the repo **Public**, add a **Readme** file, and select **Python** from the **Add .gitingnore** drop-down.



1. We will do our coding in a branch. Click the dropdown button that reads **main**. Type session-1 for the branch name and click **Create branch…**.

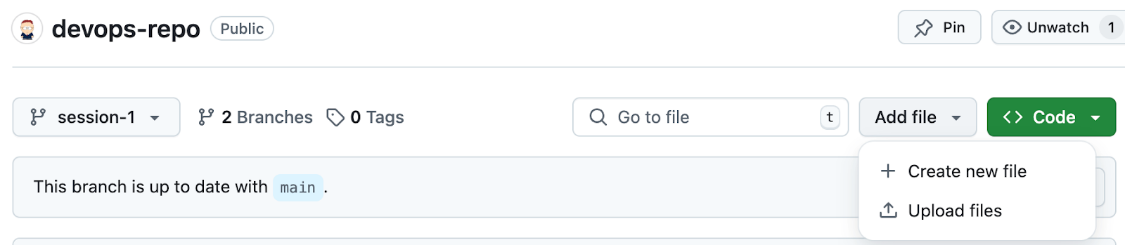


**Note:** You will automatically switch to the new branch. Make sure you name the branch session-1, as this name will be used in our GitHub Actions later.



Adding Code to the Repository

1. From the repo’s code page, click the **Add file** or **+** sign button and create a new file.



1. Name the file app.py and paste the following code into it. Click the **Commit changes** button to save the file.

from flask import Flask, request

from flask\_restful import Resource, Api

app = Flask(\_\_name\_\_)

api = Api(app)

class Home(Resource):

    def get(self):

        return {'version': '1.0'}

class ConvertTemp(Resource):

    def get(self):

        # Get arguments from query parameters

        temp = float(request.args.get('temp'))

        scale = request.args.get('scale').lower()

        target\_scale = request.args.get('target\_scale').lower()

        # Perform temperature conversion

        converted\_temp = self.convert\_temperature(temp, scale, target\_scale)

        if converted\_temp is None:

            return {'error': 'Invalid scale or target scale'}, 400

        return {'converted\_temp': converted\_temp, 'target\_scale': target\_scale}

    def convert\_temperature(self, temp, scale, target\_scale):

        # Conversion logic

        if scale == target\_scale:

            return temp

        if scale == 'celsius':

            if target\_scale == 'fahrenheit':

                return temp \* 9/5 + 32

            elif target\_scale == 'kelvin':

                return temp + 273.15

        elif scale == 'fahrenheit':

            if target\_scale == 'celsius':

                return (temp - 32) \* 5/9

            elif target\_scale == 'kelvin':

                return (temp - 32) \* 5/9 + 273.15

        elif scale == 'kelvin':

            if target\_scale == 'celsius':

                return temp - 273.15

            elif target\_scale == 'fahrenheit':

                return (temp - 273.15) \* 9/5 + 32

        # If scales are invalid

        return None

api.add\_resource(Home, '/')

api.add\_resource(ConvertTemp, '/convert-temp')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Note:** This may look like a lot of code, but it just creates a simple API using Python Flask that converts temperatures into different scales (Fahrenheit, Celsius, and Kelvin).

1. A Python program uses a requirements file to specify its prerequisites. Add another file called requirements.txt, paste the following code into it, and commit the changes.

Flask==3.0.3

Flask-RESTful==0.3.10

pytest==8.3.2

1. We need to test our code. This can be done with automated unit tests using the Python library called **PyTest**. As you just did, add another file called test\_app.py. Paste the following code and commit the changes.

import pytest

from app import app

@pytest.fixture

def client():

    with app.test\_client() as client:

        yield client

# Home route test

def test\_home(client):

    response = client.get('/')

    assert response.status\_code == 200

    assert response.json == {'version': '1.0'}

# Fahrenheit to Celsius

@pytest.mark.parametrize("temp, expected", [(212, 100.0), (32, 0.0), (-40, -40.0)])

def test\_convert\_temp\_f\_to\_c(client, temp, expected):

    response = client.get(f'/convert-temp?temp={temp}&scale=fahrenheit&target\_scale=celsius')

    assert response.status\_code == 200

    assert response.json['converted\_temp'] == pytest.approx(expected, rel=1e-2)

# Celsius to Fahrenheit

@pytest.mark.parametrize("temp, expected", [(100, 212.0), (0, 32.0), (-40, -40.0)])

def test\_convert\_temp\_c\_to\_f(client, temp, expected):

    response = client.get(f'/convert-temp?temp={temp}&scale=celsius&target\_scale=fahrenheit')

    assert response.status\_code == 200

    assert response.json['converted\_temp'] == pytest.approx(expected, rel=1e-2)

# Kelvin to Celsius

@pytest.mark.parametrize("temp, expected", [(273.15, 0.0), (373.15, 100.0), (233.15, -40.0)])

def test\_convert\_temp\_k\_to\_c(client, temp, expected):

    response = client.get(f'/convert-temp?temp={temp}&scale=kelvin&target\_scale=celsius')

    assert response.status\_code == 200

    assert response.json['converted\_temp'] == pytest.approx(expected, rel=1e-2)

# Kelvin to Fahrenheit

@pytest.mark.parametrize("temp, expected", [(273.15, 32.0), (373.15, 212.0), (233.15, -40.0)])

def test\_convert\_temp\_k\_to\_f(client, temp, expected):

    response = client.get(f'/convert-temp?temp={temp}&scale=kelvin&target\_scale=fahrenheit')

    assert response.status\_code == 200

    assert response.json['converted\_temp'] == pytest.approx(expected, rel=1e-2)

# Invalid scale handling

def test\_convert\_temp\_invalid\_scale(client):

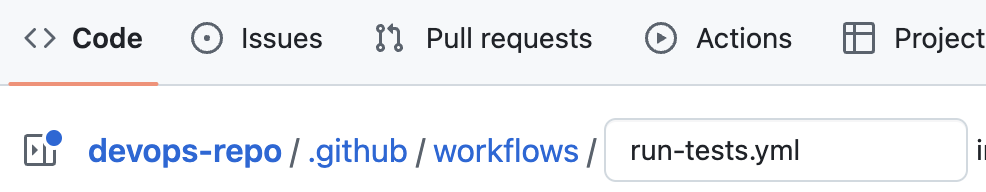
    response = client.get('/convert-temp?temp=100&scale=unknown&target\_scale=celsius')

    assert response.status\_code == 400

    assert 'error' in response.json

Using GitHub Actions to Run Automated Tests

1. GitHub Actions are added to Workflows and put into a folder called .github/workflows.   
     
   Create another file from your repository code page named `.github/workflows/run-tests.yml`.



1. Paste the following code into the file, and commit the change.

name: Run Tests on Push to session-1 and Pull Request to main

on:

  push:

    branches:

      - session-1

  pull\_request:

    branches:

      - main

jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Python

        uses: actions/setup-python@v5

        with:

          python-version: '3.11'  # Specify the Python version you want to use

      - name: Install dependencies

        run: |

          python -m pip install --upgrade pip

          pip install -r requirements.txt

      - name: Run tests

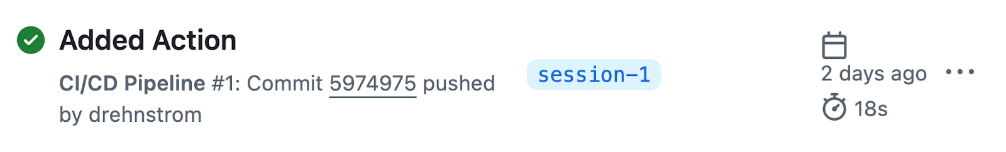
        run: |

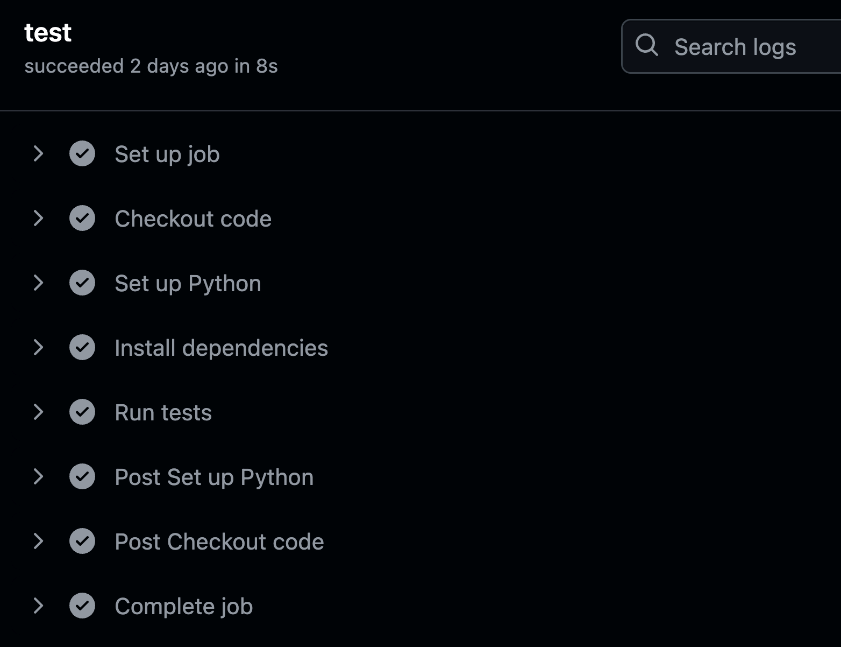
          pytest --maxfail=1 --disable-warnings

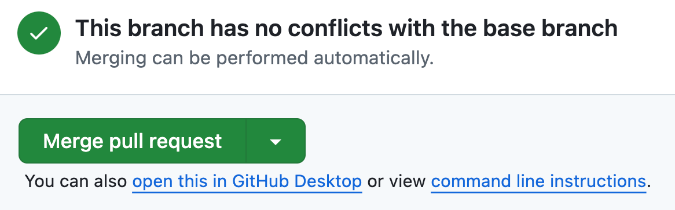
**Note:** In the example above, the workflow runs when we push to the session-1 branch or make a pull request to the main branch.

There is one job called **test** which has four steps. The first two use the built-in checkout and setup-python GitHub actions. The third step installs the Python prerequisites using Python pip. That last step runs the tests. These last two steps run the same commands you would run on your computer if you wanted to run the tests manually.

1. Commit the changes, then click the **Actions** link at the top of the page. You should see your workflow run. Click the workflow to see its details.





1. Try making a pull request to the main branch. As before, you should see the workflow run. To create a Pull request, click on the Pull Requests menu at the top.   
     
     
     
   Then, click the **Compare & pull request** button.    
     
     
     
   Add the title Merge Session 1 to Main and click **Create pull request**.  
     
     
     
   Lastly, go to the Pull Request and click the **Merge pull request** button.   
     
   

1. Experiment a little. Break the code and commit it so you can see the tests fail, then fix the code and make sure the tests pass.
2. Make sure when you are done making experiments you create a pull request from the session 1 branch to the main branch and merge your changes.

--------------------------------------------------------------------------------------------------------------------------

Session-2

Session 2 Lab: Building Docker Images with GitHub Actions

Prerequisite

1. To complete this lab, you must have already done the prior lab. If you have not already done so, complete it using the following link: [Session 1 Lab: Creating DevOps Pipelines with GitHub Actions](https://docs.google.com/document/d/1o5hoIHLMLNlnZdPZ8YY5_MXBPtGc5FpgdkPRkel9T8I/edit?usp=sharing)

Creating a Dockerfile

1. Go to <https://github.com>, sign in, and open the repository you created in the previous lab.
2. Create a new branch called session-2. Make sure to spell the branch exactly because the name will be used later in your code.
3. Click the **Add file** or + sign button, and create a new file called Dockerfile. Paste the following code into it.

FROM python:3.11-slim

WORKDIR /app

COPY . .

RUN pip install gunicorn

RUN pip install -r requirements.txt

# Expose the port on which the Flask app will run

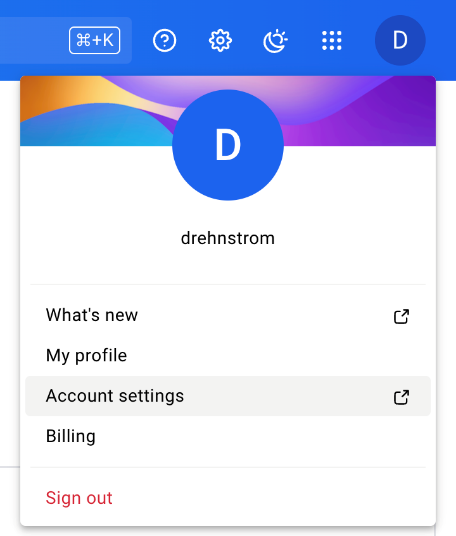
EXPOSE 8080

# Define the command to run the application using Gunicorn on port 8080

CMD ["gunicorn", "--bind", "0.0.0.0:8080", "app:app"]

Creating a Docker Hub Account and Credentials

1. Create a new browser tab, and go to **Docker Hub** using the following link: <https://hub.docker.com/>
2. If you already have a Docker account, sign in; otherwise, click the **Sign up** button to create a new one.
3. Click on your account icon in the upper-right corner and select **Account settings**.



1. From the Account settings page, select **Personal access tokens** from the **Security** section.
2. Click the **Generate new token** button and create a new access token named GitHub Action Token. Make sure the token is Read & Write. Paste the generated token in a text file; you will need it in a minute.

Running Actions with GitHub Secrets and Variables

1. Return to the browser tab with your GitHub repository. Click the **Settings** link.
2. From the Security section on the left, select **Secrets and variables**, and then **Actions**.
3. Click the **New repository secret** button. Create a secret called DOCKER\_HUB\_USERNAME with your Docker account name as the value. Then, create a second secret called DOCKER\_HUB\_ACCESS\_TOKEN with the access token you just created.
4. Return to your GitHub repository **Code** view. Make sure you are using the session-2 branch and navigate to the file .github/workflows/run-tests.yml. Open the file in **Edit** mode by clicking on the pencil icon. Replace the current contents of the file with the following code.

name: CI/CD Pipeline

on:

  push:

    branches:

      - session-1

      - session-2

  pull\_request:

    branches:

      - main

jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Python

        uses: actions/setup-python@v5

        with:

          python-version: '3.11'

      - name: Install dependencies

        run: |

          python -m pip install --upgrade pip

          pip install -r requirements.txt

      - name: Run tests

        run: |

          pytest --maxfail=1 --disable-warnings

  build\_and\_push:

    runs-on: ubuntu-latest

    needs: test

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Docker Buildx

        uses: docker/setup-buildx-action@v2

      - name: Log in to Docker Hub

        uses: docker/login-action@v2

        with:

          username: ${{ secrets.DOCKER\_HUB\_USERNAME }}

          password: ${{ secrets.DOCKER\_HUB\_ACCESS\_TOKEN }}

      - name: Build and push Docker image

        uses: docker/build-push-action@v5

        with:

          context: .

          push: true

          tags: ${{ secrets.DOCKER\_HUB\_USERNAME }}/tech-trek:${{ github.sha }}

**Note:** This workflow now has two jobs: test and build\_and\_push. Take a look at how your secrets are being used in the second job.

There is also a variable GitHub.sha that is used to tag the image. This provides a unique name for every image created. Also note the parameter needs: test ensures that the build\_and\_push job is only run if the test job succeeds.

1. Commit the changes and then switch to the **Actions** tab. After a couple of seconds, your workflow should appear. When the workflow is complete, go back to <https://hub.docker.com> and refresh the page. You should have a new Docker image created.

Merge your Changes

1. Create a **Pull request** and merge the changes you made on the session-2 branch with the main branch. Make sure to confirm and merge the pull request.
2. After you have merged the changes, go to the **Actions** menu. You should see that your pipeline ran when you did the pull request.

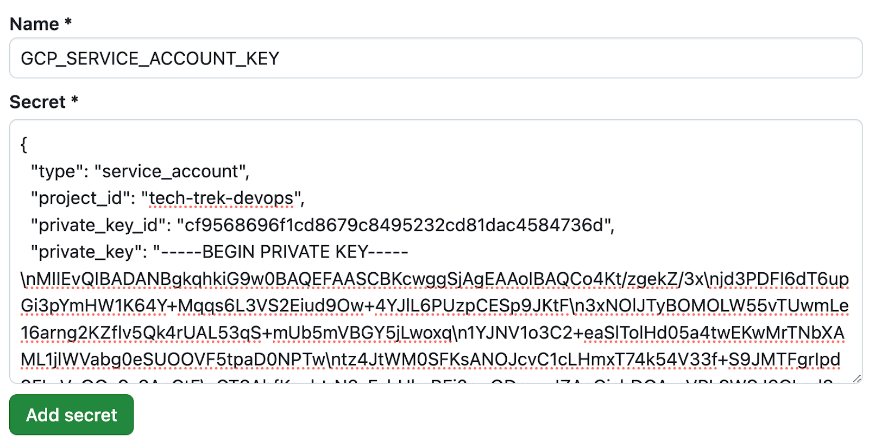
Session-3

Session 3 Lab: Deploy to Cloud Run with GitHub Actions

Prerequisites

1. To complete this lab, you must have done the prior two labs.   
     
   If you have not already done so, complete it using the following link: [Session 1 Lab: Creating DevOps Pipelines with GitHub Actions](https://docs.google.com/document/d/1o5hoIHLMLNlnZdPZ8YY5_MXBPtGc5FpgdkPRkel9T8I/edit?usp=sharing)   
     
   Also, if you have not already done so, complete it using the following link: [Session 2 Lab: Building Docker Images with GitHub Actions](https://docs.google.com/document/d/1uiIaLGE--pXvwo8C27Mief_IUULMi-bWOBHDKYXDyYw/edit?usp=sharing)

Adding Google Cloud Credentials

1. Go to <https://github.com>, sign in, and open the repository you created in the previous labs.
2. Create a new branch called session-3. Make sure to spell the branch exactly because the name will be used later in your code.
3. On the GitHub website, click the **Settings** link.
4. From the Security section on the left, select **Secrets and variables**, and then **Actions**.
5. Click the **New repository secret** button. Create a secret called GCP\_SERVICE\_ACCOUNT\_KEY. For the value of the secret, you will need to paste in the service account key provided by your instructor. The screen will look similar to the screenshot below.   
     
     
     
   **Note:** If you want to use your own Google Cloud project, you can create a Google Cloud Service account with the permissions to deploy to Cloud Run and generate a JSON key for that account. However, this is beyond the scope of this training.

Extending the CI/CD Pipeline to Deploy to Cloud Run

1. Return to your GitHub repository **Code** view. Make sure you are using the session-3 branch and navigate to the file .github/workflows/run-tests.yml. Open the file in **Edit** mode by clicking on the pencil icon. Replace the current contents of the file with the following code.

name: Test-Build-Deploy Pipeline

on:

  push:

    branches:

      - session-3

      - main

jobs:

  test:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Python

        uses: actions/setup-python@v5

        with:

          python-version: '3.11'

      - name: Install dependencies

        run: |

          python -m pip install --upgrade pip

          pip install -r requirements.txt

      - name: Run tests

        run: |

          pytest --maxfail=1 --disable-warnings

  build\_and\_push:

    runs-on: ubuntu-latest

    needs: test

    steps:

      - name: Checkout code

        uses: actions/checkout@v3

      - name: Set up Docker Buildx

        uses: docker/setup-buildx-action@v2

      - name: Log in to Docker Hub

        uses: docker/login-action@v2

        with:

          username: ${{ secrets.DOCKER\_HUB\_USERNAME }}

          password: ${{ secrets.DOCKER\_HUB\_ACCESS\_TOKEN }}

      - name: Build and push Docker image

        uses: docker/build-push-action@v5

        with:

          context: .

          push: true

          tags: ${{ secrets.DOCKER\_HUB\_USERNAME }}/tech-trek:${{ github.sha }}

  deploy:

    runs-on: ubuntu-latest

    needs: build\_and\_push

    steps:

      - name: Authenticate to Google Cloud

        uses: google-github-actions/auth@v1

        with:

          credentials\_json: ${{ secrets.GCP\_SERVICE\_ACCOUNT\_KEY }}

      - name: Deploy to Cloud Run

        uses: google-github-actions/deploy-cloudrun@v1

        with:

          service: ${{ secrets.DOCKER\_HUB\_USERNAME }}-tech-trek

          image: docker.io/${{ secrets.DOCKER\_HUB\_USERNAME }}/tech-trek:${{ github.sha }}

          region: us-central1

          flags: "--allow-unauthenticated"

      - name: Get Cloud Run Service URL

        run: |

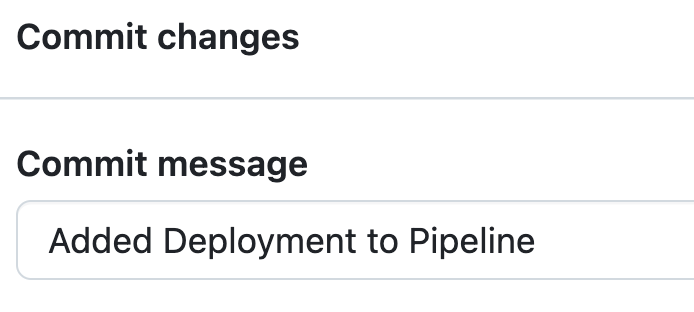
          URL=$(gcloud run services describe "${{ secrets.DOCKER\_HUB\_USERNAME }}-tech-trek" --region us-central1 --format 'value(status.url)')

          echo "Cloud Run service URL: $URL"

          echo "SERVICE\_URL=$URL" >> $GITHUB\_ENV

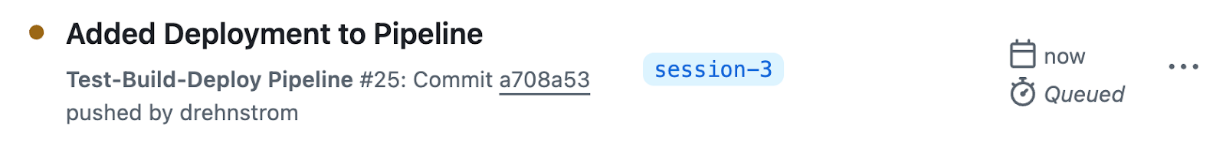
**Note:** This workflow now has three jobs: test, build\_and\_push, and deploy. We are interested in the third job for this session. The first two jobs were covered in previous sessions.

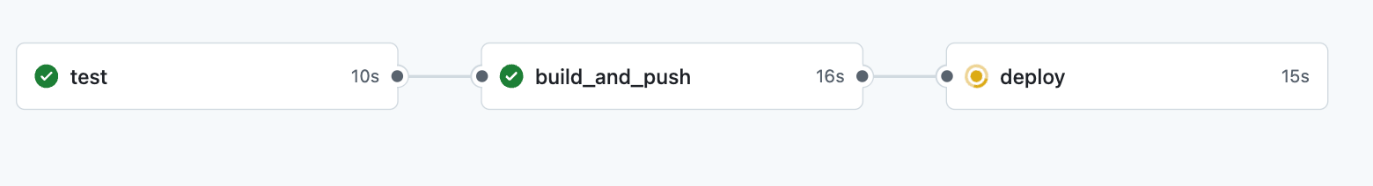
1. The first step of the deploy job uses the GitHub Google Cloud authorization action. Note that it uses the secret you just created, which includes the service account key.
2. The second step of the deploy job uses the Deploy to Cloud Run GitHub action. The name of the service will be your docker username plus the string “-tech-trek.” Also, note that you are deploying the Docker image created in the previous job.
3. The third step is just to output the URL of the Cloud Run service that is being deployed. You will see the results after the pipeline executes.
4. Click the **Commit changes** button near the top of the page to save the changes to the pipeline code.   
     
   Set the commit message to Added Deployment to Pipeline as shown in the screenshot below.



Monitoring the Pipeline

1. Click the **Actions** menu at the top. Your pipeline will appear as shown below.

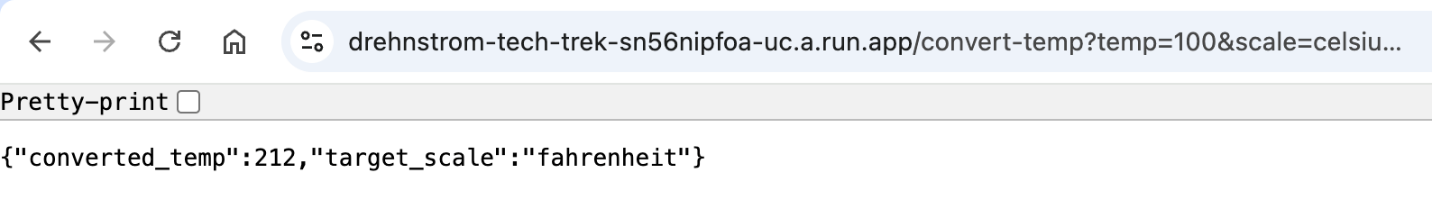


1. Click on the pipeline and you will be able to monitor the job progress.   
     
   
2. Click the third job, deploy. Expand the steps and read the output. When the job is completed, the step **Get Cloud Run Service URL** will contain the link to your service. Note that your Docker username is masked with asterisks, as shown below. This is a security feature to prevent sensitive data from being added to any logs.   
     
   Copy and paste the URL into a text file, replacing the asterisks with your Docker hub username. Then, test the URL in your browser. It should return a version number.



1. Add the following string to the end of the URL and it should return a temperature conversion, as shown in the screenshot.

/convert-temp?temp=100&scale=celsius&target\_scale=fahrenheit



Merge your Changes

1. Create a **Pull request** and merge the changes you made on the session-3 branch with the main branch.
2. After you have merged the changes, go to the **Actions** menu. You should see that your pipeline runs again.