Circle CI Quick Guide

CircleCI is a service that detects when your codebase changes and runs jobs (that you define) on the code. It's used for CI/CD.

- ✓ Continuous Integration (CI) - When pushing a code change, you want to know if the state of the code is still okay. (Ex. Does the code compile (build), and do all the tests pass.)
- ✓ Continuous Deployment (CD) - After making a code change (that hopefully passes your CI checks), deploy that code to whever it's meant to be. (Ex. Deploy a server to Heroku/AWS, or run a one-time job (ex. database migration).)

Our Sample Python Project

Let's create a file named "main.py" and add the following code to it:

```
def Add(a, b):
    return a + b

def SayHello():
    print("sup world from srcmake")

if __name__ == '__main__':
    SayHello()
```

We can run this file with the following command (in your terminal):

```
python3 main.py
```

Let's also create a file named "main-test.py" to test our main file.

```
# Import the Add function, and assert that it works correctly.
from main import Add

def TestAdd():
    assert Add(2,3) == 5
    print("Add Function works correctly")

if __name__ == '__main__':
    TestAdd()
```

We can run the test using the following command (in your terminal):

```
python3 main-test.py
```

Push The Project To Github

Let's create a github repository for this project. Go to github and make a new repo named "circleci-demo".

Now let's push our project (which is currently on our local computer) to the github repo.

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://github.com/xxxxxx/circleci-demo.git
git push -u origin master
```

Setting Up CI With CircleCI

We need to create a configuration file so that CircleCI will know what we want it to do .

Create a folder named ".circleci" and inside of it create a file named "config.yml".

```
mkdir .circleci
touch .circleci/config.yml
```

Open ".circleci/config.yml" and add the following code:

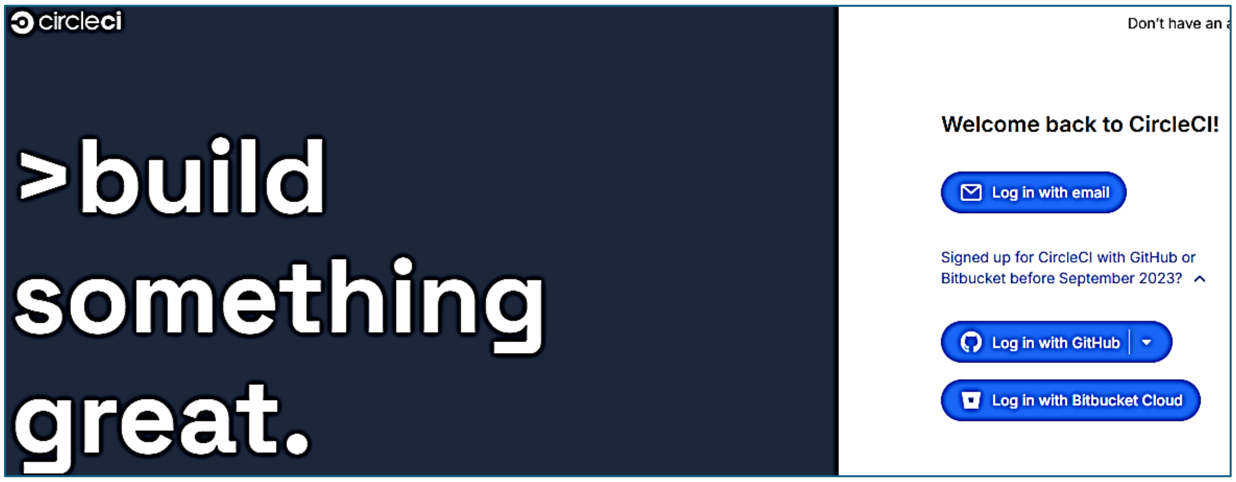
```
version: 2.1

jobs:
 build:
  working_directory: ~/circleci-python
  docker:
   - image: "circleci/python:3.6.4"
  steps:
   - checkout
   - run: python3 main.py
 test:
  working_directory: ~/circleci-python
  docker:
   - image: "circleci/python:3.6.4"
  steps:
   - checkout
   - run: python3 main-test.py

workflows:
 build_and_test:
  jobs:
   - build
   - test:
     requires:
      - build
```

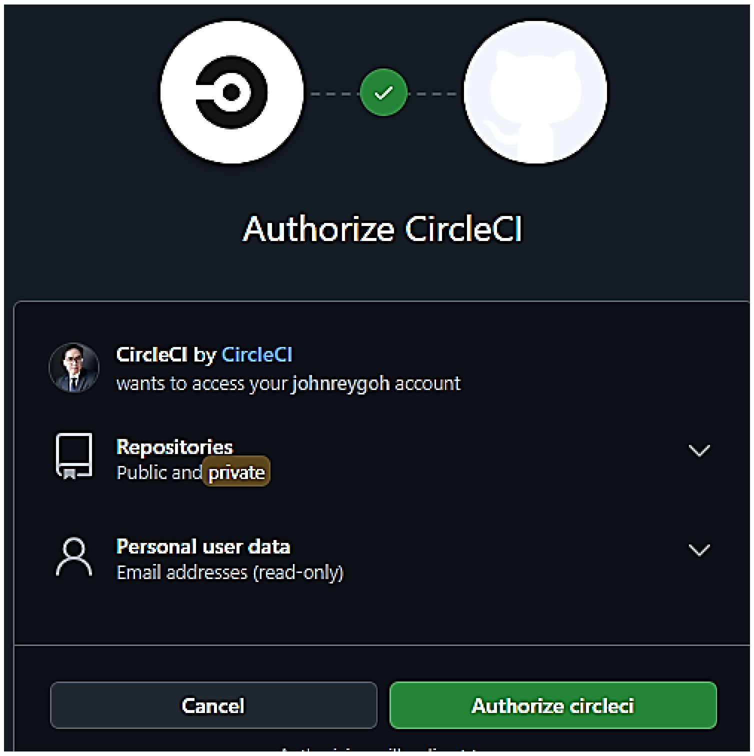
Notice that we have two types of tasks (jobs) that we want CircleCI to do: We want to run main.py, and we want to run our tests. We're also specifying that for every branch we want to perform the build job, and the test job.

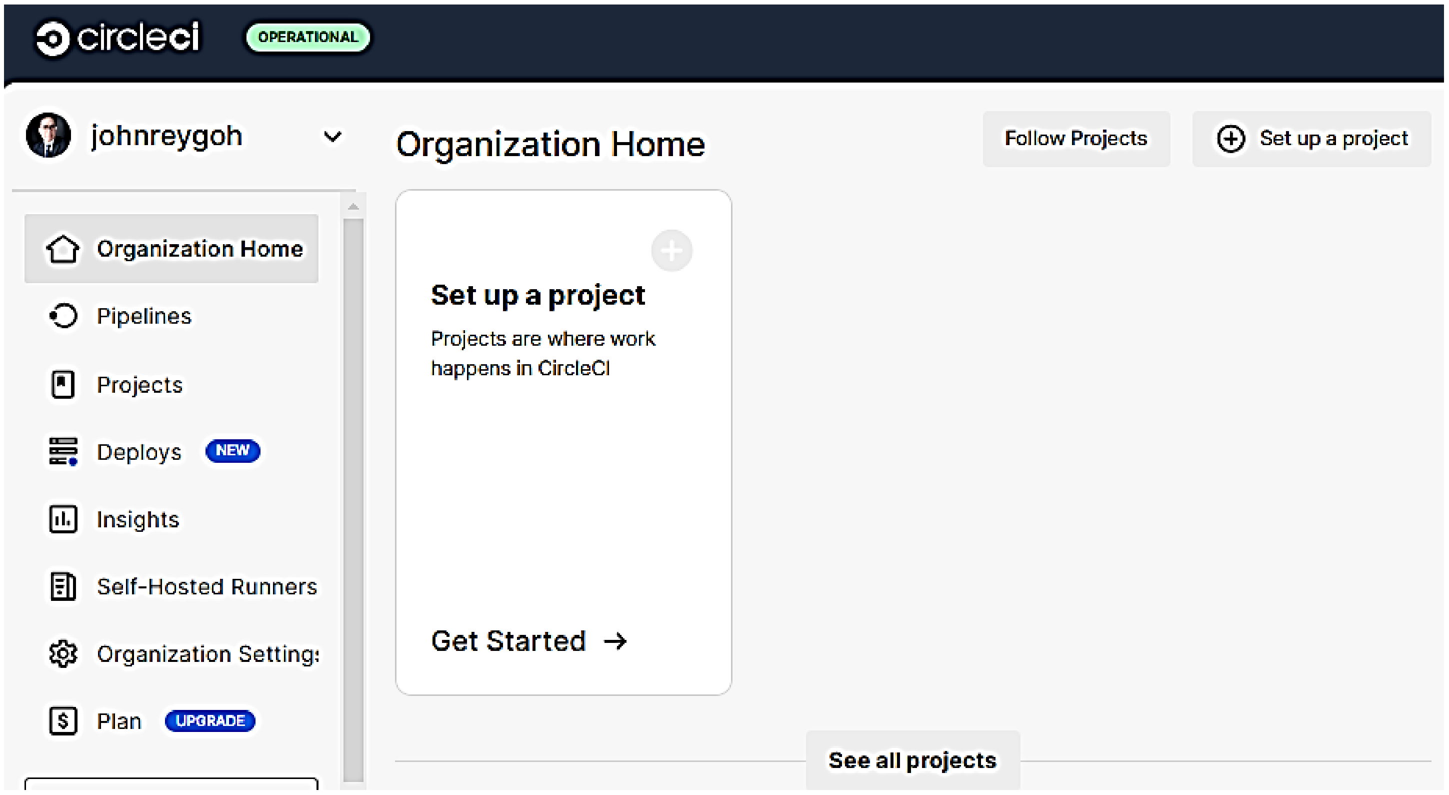
The official CircleCI Docs can show how to create more complex configurations:
https://circleci.com/docs/2.0/hello-world/

We also need to give CircleCI access to our repo so that they're authorized to run these jobs. Go to CircleCI's website:
https://circleci.com/

Login in with your github account, and authorize them to access your circleci-demo repo that we just created.

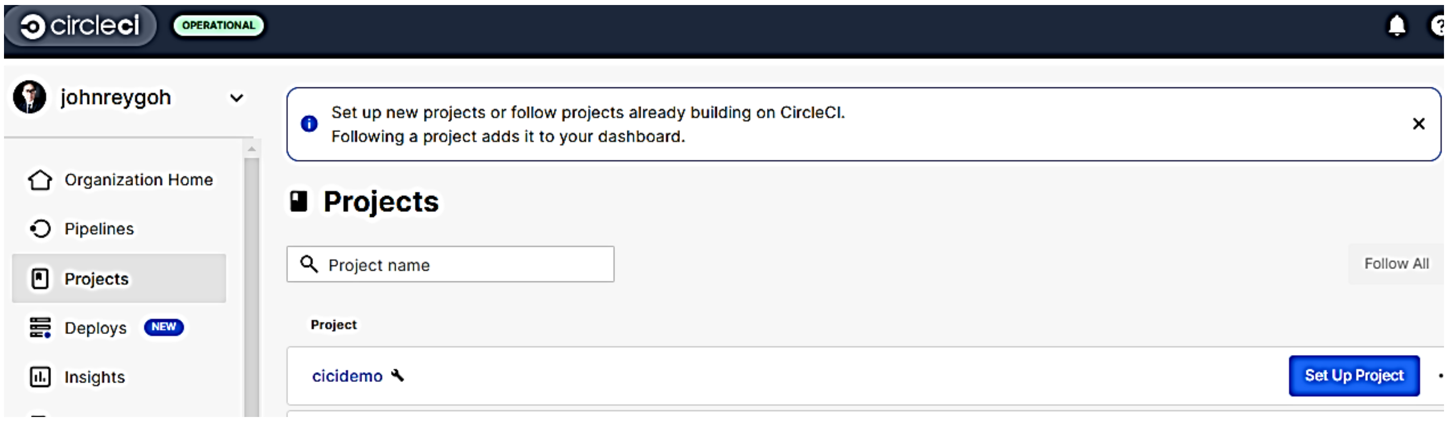

circleci
>build something great.
Don't have an
Welcome back to CircleCI!
Log in with email
Signed up for CircleCI with GitHub or Bitbucket before September 2023?
Log in with GitHub
Log in with Bitbucket Cloud



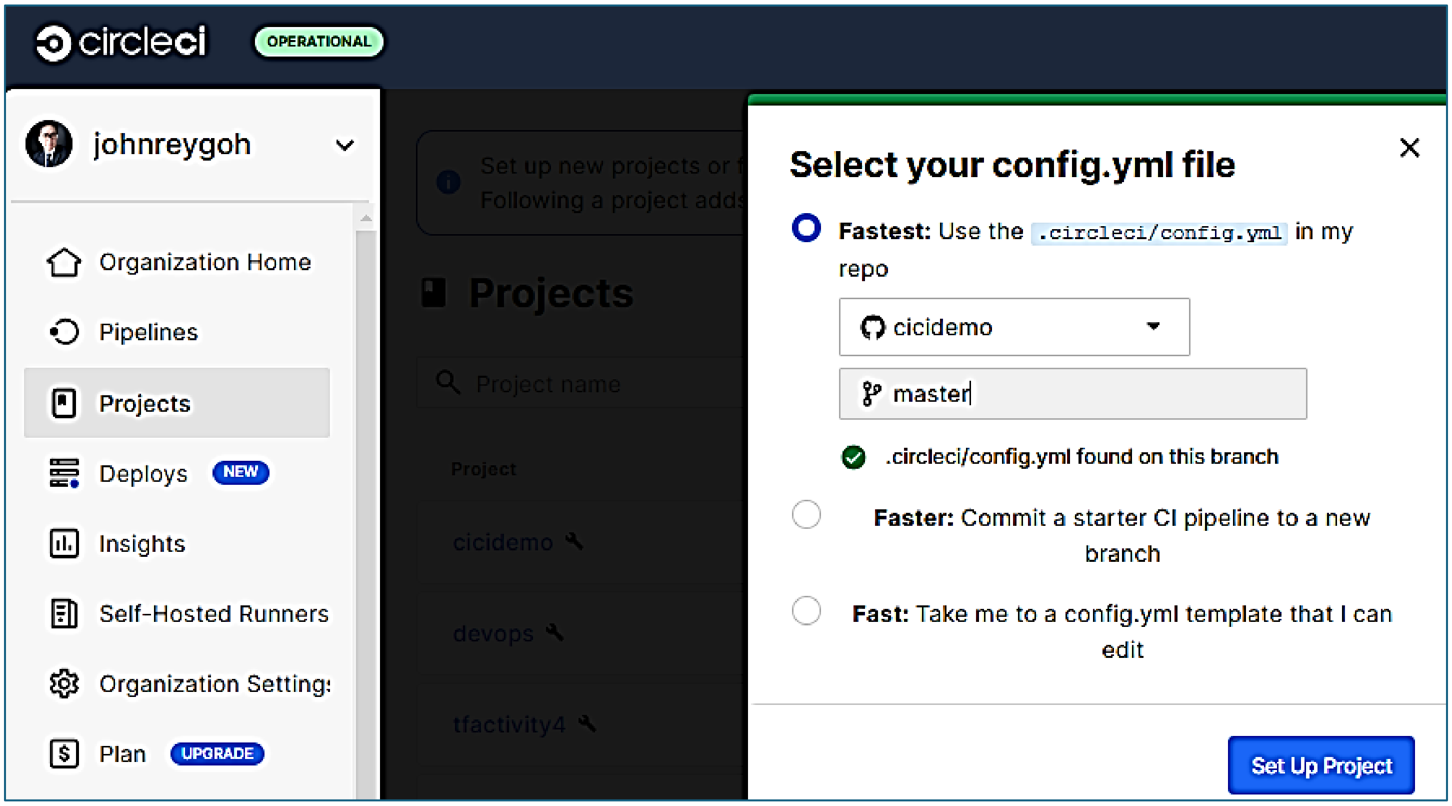


Set up a project

Projects are where work happens in CircleCI



Set up new projects or follow projects already building on CircleCI.
Following a project adds it to your dashboard.



Select your config.yml file

[✓ **Fastest:** Use the `.circleci/config.yml` in my repo]

cicidemo

master

.circleci/config.yml found on this branch

Faster: Commit a starter CI pipeline to a new branch

Fast: Take me to a config.yml template that I can edit

Now let's push our new config file to github.

```
git add .
git commit -m "Adds CircleCI config file"
git push
```

CircleCI should detect the code changes and run our build_and_test workflow for our branch. **We should be able to see the jobs run on CircleCI's website.**

(Pro-tip: If there are errors then make sure your config.yml is linted correctly. Use 2 spaces instead of tabs.)

The full source code for this project:
https://github.com/johnreygoh/cicidemo

Testing That CircleCI Is Working
We have CI running now so let's test it out. Let's add a line to our unit test for our Add function to make sure 5 + 5 is equal to 10.

Check out a new branch:

```
git checkout -b add-test
```

Add the following line of code to our TestAdd() function in "main-test.py":

```
assert Add(5,5) == 10
```

Push this branch up to github.

```
git add .
git commit -m "Adds test to make sure 5 + 5 is equal to 10"
git push --set-upstream origin add-test
```

Let's go to our github repo and make a Pull Request for this branch to see if CircleCI is running our jobs.

We should see a build job and a test job with green checkmarks since they pass. We have CI working properly.

Of course, the useful part of CI (and unit tests) is making sure our codebase is working properly. Let's see what happens if we mess up old code by making our Add function do multiplication instead of tradition. In "main.py" change our Add function to multiply instead of add.
return a * b

Commit this change and push our updated branch to github.

```
git add .
git commit -m "Add function now multiplies instead of adds"
git push
```

If we look at our Pull Request, we should see that the build step passes since the code is still functional, but now our test fails since 2 * 3 isn't 5. So now we'd know the code change being introduced by this branch is harmful and we shouldn't merge this PR (and CircleCI can use that info to stop other jobs, like making sure not to deploy if the tests fail).

Sample CircleCI yml files (docs)
https://circleci.com/docs/sample-config/

Activity: Sample Circle CI Workflow

Here's a complete .circleci/config.yml file that does the following:

1. Builds and tests a Flask application.
2. Creates a Docker image.
3. Pushes the Docker image to Docker Hub.
4. Uses Terraform to create an EC2 Amazon Linux 2023 instance.
5. Uses Ansible to pull the Docker image into the EC2 instance and run it in a container.

Prerequisites

- ✓ You have a Docker Hub account.
- ✓ Your repository contains:
- ✓ A Flask app (app.py).
- ✓ A Dockerfile.
- ✓ A Terraform configuration (terraform/ directory).
- ✓ An Ansible playbook (ansible/ directory).

Environment variables are set in CircleCI:

- DOCKERHUB_USERNAME
- DOCKERHUB_PASSWORD
- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY
- EC2_SSH_KEY (for Ansible)

.circleci/config.yml

```
version: 2.1

executors:
 python-executor:
  docker:
   - image: circleci/python:3.10
  working_directory: ~/project

 docker-executor:
  docker:
   - image: circleci/docker:latest
  working_directory: ~/project

 terraform-executor:
  docker:
   - image: hashicorp/terraform:latest
  working_directory: ~/project

 ansible-executor:
  docker:
   - image: williamyeh/ansible:latest
  working_directory: ~/project

jobs:
 build-and-test:
  executor: python-executor
  steps:
   - checkout
   - run:
     name: Install dependencies
     command: |
      python -m venv venv
      source venv/bin/activate
      pip install -r requirements.txt
   - run:
     name: Run tests
     command: |
        source venv/bin/activate
        pytest tests/

 build-docker-image:
  executor: docker-executor
  steps:
   - checkout
   - setup_remote_docker
   - run:
      name: Build Docker Image
      command: docker build -t $DOCKERHUB_USERNAME/flask-app:latest .
   - run:
      name: Log in to Docker Hub
      command: echo "$DOCKERHUB_PASSWORD" | docker login -u "$DOCKERHUB_USERNAME" --
password-stdin
   - run:
      name: Push Docker Image
      command: docker push $DOCKERHUB_USERNAME/flask-app:latest

 deploy-terraform:
  executor: terraform-executor
  steps:
   - checkout
   - run:
      name: Initialize Terraform
      command: |
       cd terraform
       terraform init
   - run:
      name: Apply Terraform
      command: |
       cd terraform
       terraform apply -auto-approve

 deploy-ansible:
  executor: ansible-executor
  steps:
   - checkout
   - run:
      name: Install SSH key
      command: |
       echo "$EC2_SSH_KEY" > ~/.ssh/id_rsa
       chmod 600 ~/.ssh/id_rsa
   - run:
      name: Run Ansible Playbook
      command: |
        cd ansible
        ansible-playbook -i inventory.ini deploy.yml

workflows:
 version: 2
 deploy-app:
  jobs:
   - build-and-test
   - build-docker-image:
      requires:
       - build-and-test
   - deploy-terraform:
      requires:
       - build-docker-image
   - deploy-ansible:
      requires:
       - deploy-terraform
```

Supporting Files

1. Dockerfile (Build Flask App)

```
FROM python:3.10
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

2. terraform/main.tf (Create EC2 Instance)

```
provider "aws" {
 region = "us-east-1"
}

resource "aws_instance" "flask_app" {
 ami         = "ami-0c55b159cbfafe1f0"  # Amazon Linux 2023 AMI (update if needed)
 instance_type = "t2.micro"
 key_name     = "my-key"

 tags = {
  Name = "flask-app-instance"
 }

 provisioner "file" {
  source     = "../ansible"
  destination = "/home/ec2-user/ansible"
 }
}

output "ec2_public_ip" {
  value = aws_instance.flask_app.public_ip
}
```

3. ansible/inventory.ini (Inventory File)

```
[web]
ec2-instance ansible_host=<EC2_PUBLIC_IP> ansible_user=ec2-user ansible_ssh_private_key_file=~/.ssh/id_rsa
```

4. ansible/deploy.yml (Ansible Playbook)

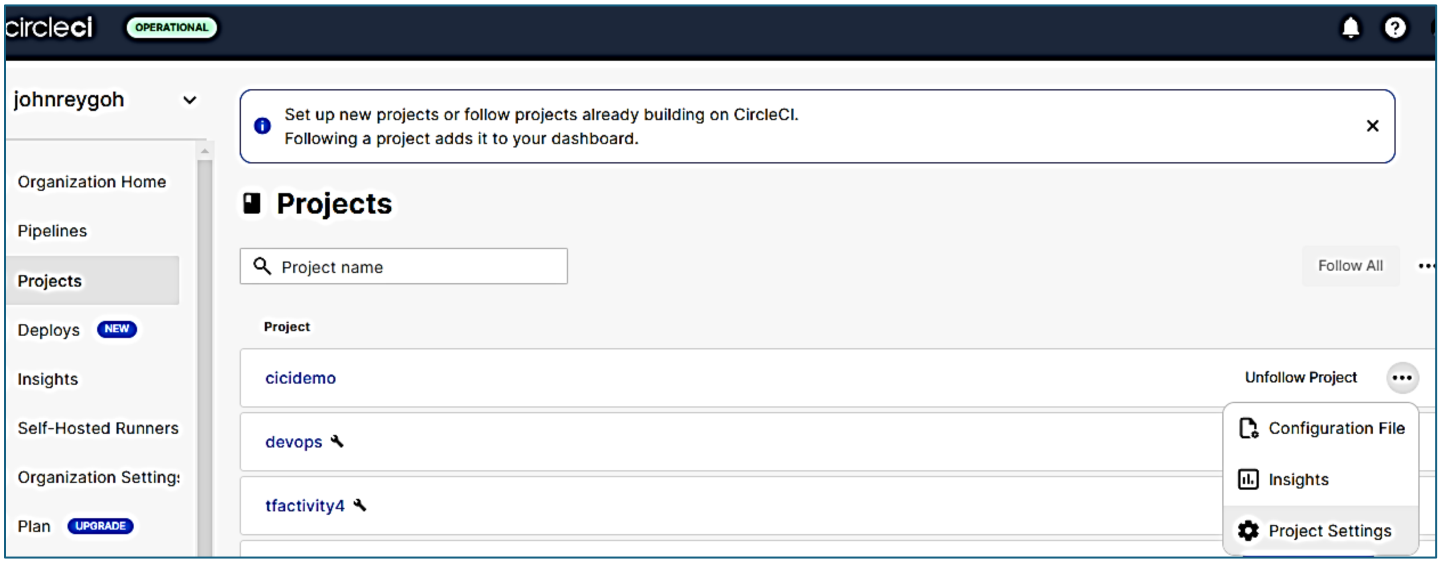
```
- name: Deploy Flask App
  hosts: web
  become: true
  tasks:
    - name: Install Docker
      yum:
        name: docker
        state: present

    - name: Start Docker Service
      service:
        name: docker
        state: started
        enabled: true

    - name: Pull Docker Image
      command: docker pull {{ lookup('env', 'DOCKERHUB_USERNAME') }}/flask-app:latest

    - name: Run Docker Container
      command: docker run -d -p 5000:5000 --name flask-app {{ lookup('env', 'DOCKERHUB_USERNAME')
}}/flask-app:latest
```

Workflow Breakdown

build-and-test	Installs dependencies and runs tests on the Flask app.
build-docker-image	Builds a Docker image and pushes it to Docker Hub.
deploy-terraform	Creates an EC2 instance using Terraform.
deploy-ansible	Uses Ansible to: 1. Install Docker. 2. Pull the Flask app image from Docker Hub. 3. Run the Flask app inside a container.

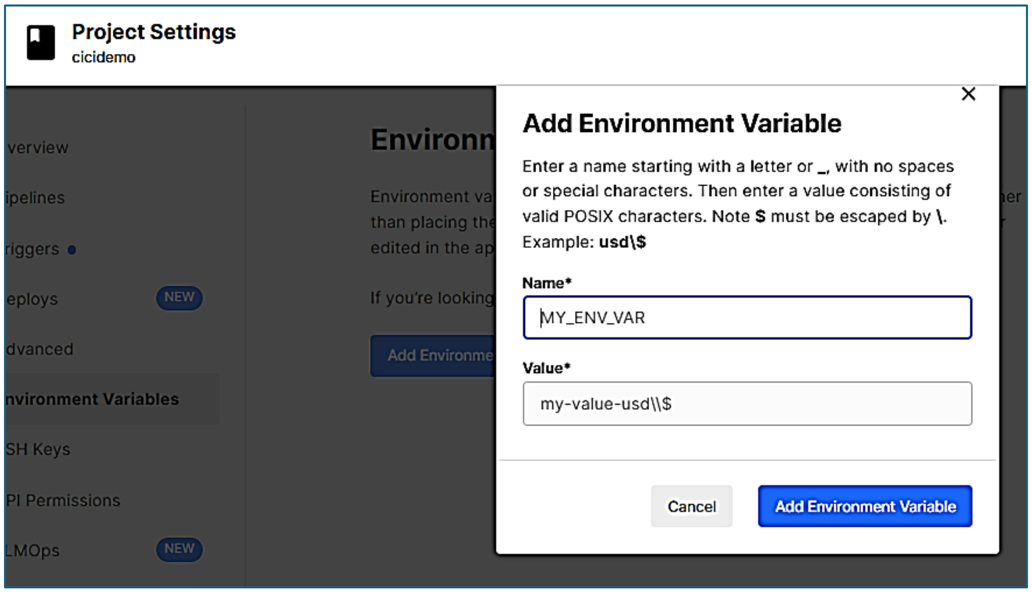
How to Set Up Environment Variables in CircleCI

- ✓ DOCKERHUB_USERNAME
- ✓ DOCKERHUB_PASSWORD
- ✓ AWS_ACCESS_KEY_ID
- ✓ AWS_SECRET_ACCESS_KEY
- ✓ EC2_SSH_KEY (Private SSH key for EC2)



Set up new projects or follow projects already building on CircleCI.
Following a project adds it to your dashboard.

Project
cicidemo
devops
tfactivity4



Add Environment Variable

Enter a name starting with a letter or _, with no spaces or special characters. Then enter a value consisting of valid POSIX characters. Note **$** must be escaped by **\\**. Example: **usd\\$**

Name*: MY_ENV_VAR

Value*: my-value-usd\\\\$

CircleCI will automatically start the pipeline on github push

Final Outcome

- ✅ Flask app is built and tested
- ✅ Docker image is created and pushed to Docker Hub
- ✅ EC2 instance is provisioned via Terraform
- ✅ Ansible pulls and runs the Docker container in EC2
- ✅ Your Flask app is running on http://your-ec2-ip:5000