

## GENERAL

# How to Integrate Pulumi with GitHub Actions



Flavius Dinu

14 Feb 2024 · 13 min read



Pulumi is one of the best infrastructure-as-code (IaC) tools available today, letting you leverage different programming languages for managing your infrastructure. As with every IaC tool, you need to leverage continuous integration/continuous deployment (CI/CD) pipelines to streamline the integration and deployment workflows for your infrastructure.

We will cover:

1. [What is Pulumi?](#)
2. [What is GitHub Actions?](#)
3. [Why Pulumi and GitHub Actions?](#)
4. [Key benefits of using Pulumi with GitHub Actions](#)
5. [Setting up Pulumi with GitHub Actions](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## What is Pulumi?

**Pulumi** is an open-source IaC tool that allows developers to define and manage cloud infrastructure using programming languages such as JavaScript, TypeScript, Python, Go, and .NET. It enables developers to apply familiar practices such as loops, functions, and classes to define cloud resources, and it supports a wide range of cloud providers, including AWS, Microsoft Azure, Google Cloud, Kubernetes, and others, facilitating the creation, deployment, and management of cloud infrastructure.

## What is GitHub Actions?

GitHub Actions is a CI/CD platform that enables developers to automate their workflows directly within GitHub. With GitHub Actions, you can set up workflows to build, test, and deploy your code, based on events within your GitHub repositories, such as pull requests, merges, and even manual interventions.

Apart from defining your custom tasks, GitHub Actions leverages a vast ecosystem of reusable tasks called actions that are built and supported by the community and GitHub itself, allowing for highly customizable pipelines that can integrate with various tools and services.

## Why Pulumi and GitHub Actions?

### Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)



build, test, and deployment increasingly popular, and many Combining it with Pulumi can it more automated, reproducible,

Check out also: [How to manage Terraform with GitHub Actions](#)

## Key benefits of using Pulumi with GitHub Actions



directly from your CI/CD pipeline, reducing manual effort and the potential for error.

- **Version Controlled Infrastructure** – Manage your IaC, allowing you to track changes, review history, and easily revert when necessary.
- **Enhanced Collaboration** – Increase the collaboration inside the team by easily reviewing infrastructure changes in pull requests.
- **Streamlined Workflow** – Keep your infrastructure and application code in sync, ensuring that your deployments are consistent and reliable.

Here are some limitations when leveraging the integration:

- **Complex workflow configuration** – Setting up a workflow for Pulumi in GitHub Actions can be complex, and you will need to have a good understanding of both tools to build it properly.
- **Concurrency and state management** – [Pulumi relies on state files](#) to manage and track the state of cloud resources. When using Pulumi with GitHub Actions in a team environment, concurrent executions might lead to conflicts or race conditions if not carefully managed, especially in scenarios where multiple team members trigger deployments simultaneously.
- **Hard to share secrets, files, and configurations between multiple workflows** – This can be challenging and requires careful handling to avoid exposing sensitive data.
- **Hard to share outputs between multiple workflows** – It's very hard to share outputs from multiple repositories defined for your Pulumi workflows.

After demonstrating how to configure Pulumi with GitHub Actions, we will present an alternative that addresses all these limitations – [Spacelift](#). With Spacelift, you won't need to define the workflow

## Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

- [Why DevOps Engineers Love and Recommend Spacelift](#)
- [How to Improve Your Infrastructure as Code](#)



## Setting up Pulumi with GitHub Actions

To set up Pulumi with GitHub Actions, we will need to take the following steps:

1. Create a GitHub repository and clone it.
2. Prepare the Pulumi code.
3. Prepare the GitHub actions workflows.
4. Configure the GitHub environment for manual approvals.
5. Configure the GitHub secrets.
6. Push the configuration to GitHub.
7. Run the pipelines.

### 1. Create a GitHub repository and clone it

Go to your GitHub account and create a repository:

#### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

#### Repository template

No template ▾

Start your repository with a template repository's contents.

Owner \*



saturnhead ▾

Repository name \*

pulumi\_example

✔ pulumi\_example is available.

Great repository names are short and memorable. Need inspiration? How about **fluffy-funicular** ?

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

#### Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

#### Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

## Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)





in my case, I've selected an  
md file.

```
git clone <repo_link>
```

You can get the repo\_link by clicking on the code tab inside your newly created repository:

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)




 Clone 

HTTPS

SSH

GitHub CLI

`git@github.com:saturnhead/pulumi_examples.0` 

Use a password-protected SSH key.

 Open with GitHub Desktop

 Download ZIP

## 2. Prepare the Pulumi code

### Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

an EC2 instance and a security

the following command inside your

# Output

This command will walk you through creating a new Pulumi project.

Enter a value or leave blank to accept the (default), and press <ENTER>.

Press ^C at any time to quit.

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



```
stack name (dev):
Created stack 'dev'
Enter your passphrase to protect config/secrets:
Re-enter your passphrase to confirm:

aws:region: The AWS region to deploy into (us-east-1): eu-west-1
Saved config
```

This will generate the following files:

```
ls
# Output
Pulumi.dev.yaml  Pulumi.yaml  __main__.py  requirements.txt  venv
```

We will make changes only in the `__main__.py` file for now. When this file is generated, by default it will have a configuration that creates an S3 bucket and exports the bucket name and the bucket id:

```
"""An AWS Python Pulumi program"""

import pulumi
```

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

As mentioned before, we want to create an EC2 instance, a security group, and an attachment. We will need to modify the code, remove the import to S3, and import the class that takes care of EC2 resources:



Now that we've handled the imports, let's create a security group:

```
# Create a new security group
security_group = ec2.SecurityGroup('my-security-group',
    description='Enable SSH access',
    ingress=[
        {'protocol': 'tcp', 'from_port': 22, 'to_port': 22, 'cidr_blocks': ['0.0.0.0/0']},
    ])
```

If you are accustomed to OpenTofu or Terraform, you will see the syntax doesn't feel that much different, although we are using a programming language here.

Next, let's create an EC2 instance with this security group attached to it:

```
# Create a new EC2 instance
instance = ec2.Instance('my-instance',
    instance_type='t2.micro',
    ami='ami-0905a3c97561e0b69',
    security_groups=[security_group.name])
```

## Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

way. You just have to specify the

The configuration is now ready, and we can go on to the next step.





For this example, we will prepare two GitHub actions workflows:

- Workflow 1: plan + manual intervention for apply
- Workflow 2: manually run destroy

### Workflow 1: plan + manual intervention for apply

In the first part, we will add a name to the workflow, specify when to run it, and declare a couple of environment variables that we will need for authenticating to AWS:

```
# pulumi_deploy.yaml

name: Deploy EC2 Instance
on: [push]
env:
  AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
  AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
  AWS_SESSION_TOKEN: ${ secrets.AWS_SESSION_TOKEN }
  AWS_REGION: ${ secrets.AWS_REGION }
  PULUMI_STACK: aws_ec2
  AWS_KMS_KEY_ID: ${ secrets.AWS_KMS_KEY_ID }
```

You don't necessarily require a session token, but I always try to use short-lived credentials, even if this requires extra work.

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

```
run:
  working-directory: ./aws_ec2

steps:
- uses: actions/checkout@v2
```



```
- name: Install dependencies
  run: |
    pip install pulumi pulumi_aws

- name: Login to S3 for state management
  run: |
    pulumi login s3://pulumi-state-saturnhead

- name: Create Pulumi Stack
  run: pulumi stack init ${PULUMI_STACK} --secrets-provider="awskms://${ env.AWS_KMS_KEY_ID }"
  continue-on-error: true

- name: Select Pulumi Stack
  run: pulumi stack select ${PULUMI_STACK} --secrets-provider="awskms://${ env.AWS_KMS_KEY_ID }"

- name: Run a plan
  run: |
    pulumi stack change-secrets-provider "awskms://${ env.AWS_KMS_KEY_ID }"
    pulumi preview
```

Let's break down the above configuration:

- First, we define what will be our runner and what is the working directory we will use.
- Then we check out the code.

## Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

to the bucket (this bucket should

this step can fail if the stack exists.

this KMS key has to exist) and run

Now let's take a look at the apply job:

```
deploy:
  needs: setup-and-preview
```



```
environment: production
steps:
- uses: actions/checkout@v2

- name: Set up Python
  uses: actions/setup-python@v2
  with:
    python-version: '3.x'

- name: Install dependencies
  run: |
    pip install pulumi pulumi_aws

- name: Login to S3 for state management
  run: |
    pulumi login s3://pulumi-state-saturnhead

- name: Select Pulumi Stack
  run: pulumi stack select ${PULUMI_STACK} --secrets-provider="awskms://${{ env.AWS_KMS_KEY_ID }}"

- name: Deploy
  run: |
    pulumi stack change-secrets-provider "awskms://${{ env.AWS_KMS_KEY_ID }}"
    pulumi up --yes
```

## Struggling to balance developer velocity with control?

Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

and Pulumi AWS, stack selection,

is finished successfully.

annual approval, only after we are

can use any name you want.

- Finally, we've changed the Pulumi command to *pulumi up*, which is similar to a tofu/terraform apply.

This workflow is now complete, so we can take a look at the second one.

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)

```
# pulumi_destroy.yaml

name: Destroy EC2 Instance
on: workflow_dispatch
env:
  AWS_ACCESS_KEY_ID: ${ secrets.AWS_ACCESS_KEY_ID }
  AWS_SECRET_ACCESS_KEY: ${ secrets.AWS_SECRET_ACCESS_KEY }
  AWS_SESSION_TOKEN: ${ secrets.AWS_SESSION_TOKEN }
  AWS_REGION: ${ secrets.AWS_REGION }
  PULUMI_STACK: aws_ec2
  AWS_KMS_KEY_ID: ${ secrets.AWS_KMS_KEY_ID }

jobs:
  destroy:
    runs-on: ubuntu-latest
    defaults:
      run:
        working-directory: ./aws_ec2
    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'

      - run: pulumi stack select ${PULUMI_STACK} --secrets-provider="awskms://${ env.AWS_KMS_KEY_ID }
```

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

```
run: pulumi stack select ${PULUMI_STACK} --secrets-provider="awskms://${ env.AWS_KMS_KEY_ID }
```

```
- name: Destroy
  run: |
```

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



The above workflow is pretty similar to the ones that do plan and apply, but there are two notable differences:

- This workflow will always run manually.
- The command, in the end, is a *pulumi down*, which is pretty similar to tofu/terraform destroy.

## 4. Configure the GitHub environment for manual approvals

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

environment.

Add a name for your environment; it should be the same one you used in the GitHub Actions workflow. In my case, this will be *production*.

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Next, click on **Required reviewers**, and add the person(s) that you want to be able to deploy your Pulumi code.

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

We now need to define the GitHub Actions secrets. For that, we need to go to **Settings** → **Secrets and Variables** and select **Actions**. Then, to add a secret you need to select the **New Repository** secret option and provide the values specific to your account.

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## 6. Push the configurations to GitHub

As for any git provider, you need to add/commit/push your changes for them to be available:

```
git add .  
git commit -m 'Initial version of the Pulumi workflow'  
git push
```

## 7. Run the pipelines

As we did the push, and the first workflow has an “on: push” option on it, the pipeline will be triggered automatically:

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Now, we have reached the deploy job, but this cannot continue until we approve it:

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)



Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



After you click on **Approve and deploy**, the code is applied successfully:

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



workflow.

After a couple of seconds, we can see that all resources have been destroyed:

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



We now have a working GitHub Actions pipeline for Pulumi, and if you don't want to follow all of these steps to configure everything, you can take the code from [here](#), create a repo based on it, and just add your environment and environment secrets.

Now that we've seen the lengthy process for configuring a GitHub Actions workflow for Pulumi, let's take a look at how simple it is with Spacelift and the advantages of leveraging it.

## Building a Pulumi pipeline with Spacelift

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

...ing an ec2 instance and security  
..., and select **Create a Stack:**

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



In the following step, select Pulumi as your IaC management tool and provide a Login URL (we will

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Next, you will need to select a runner image that has Pulumi installed, one example is:

`public.ecr.aws/spacelift/runner-pulumi-base-alpine:latest`

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Now we can simply create the stack.

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

In the Environment tab, define a variable called `AWS_REGION` in which we will specify the region



Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Going back to the stack view, we can trigger a run. As I didn't select the AutoDeploy option in the view where I've added the runner image, this run will do a pulumi preview, then wait for my input to apply the changes:

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



Tearing down everything, couldn't be simpler – you just need to go to tasks, and run a *pulumi down* – yes:

## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## Key points

Although GitHub Actions is very powerful for many use cases, there are simpler and less error-prone ways to manage infrastructure.

If you are searching for the right product to manage your OpenTofu, Terraform, Pulumi, Ansible, CloudFormation, Kubernetes, and Terragrunt, Spacelift is the answer.

You can [create a free account today](#), or [book a demo](#) with one of our engineers to learn more.

## The Most Flexible CI/CD Automation Tool

Spacelift is an alternative to using homegrown solutions on top of a generic CI. It helps overcome common state management issues and adds several must-have capabilities for infrastructure management.

[Start free trial](#)

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



experience based on his exposure



## Read also

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

13 min read

[2024]

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



OPENTOFU

16 min read

## OpenTofu: The Open-Source Alternative to Terraform

### Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)

TERRAFORM

11 min read

## Pulumi vs. Terraform : Key Differences and Comparison

Solving the DevOps Infrastructure Dilemma: Enabling developer velocity with control [Register for the webinar here](#)



## Product

[Documentation](#)

[How it works](#)

[Spacelift Tutorial](#)

[Pricing](#)

[Customer Case Studies](#)

[Integrations](#)

[Security](#)

[System Status](#)

[Product Updates](#)

[Test Pilot Program](#)

## Company

[About Us](#)

[Careers](#)

[Contact Sales](#)

[Partners](#)

[Media resources](#)

## Learn

[Blog](#)

[Atlantis Alternative](#)

[Terraform Cloud Alternative](#)

[Terraform Enterprise Alternative](#)

[Spacelift for AWS](#)

[Terraform Automation](#)

Subscribe



## Struggling to balance developer velocity with control?



Attend the June 25 webinar:  
Solving the DevOps Infrastructure Dilemma

[Register for the webinar](#)