Pulumi & AWS get started tutorial

Pulumi's infrastructure-as-code SDK helps you create, deploy, and manage AWS containers, serverless functions, and infrastructure using familiar programming languages.

This tutorial takes you through the steps to easily deploy a static website.

Install Pulumi

macOS | Windows | Linux

Copy

```
$ curl -fsSL https://get.pulumi.com | sh
```

Or explore more installation options.

Install language runtime

TypeScript | JavaScript | Python | Go | C# | Java | YAML

Install Node.js.

Configure Pulumi to access AWS

Pulumi requires cloud credentials to manage and provision resources. Use an IAM user account that has **programmatic access** with rights to deploy and manage resources.

If you have previously installed and configured the AWS CLI, Pulumi will respect and use your configuration settings.

If you do not have the AWS CLI installed or plan on using Pulumi from within a CI/CD pipeline, retrieve your access key ID and secret access key and then set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables on your workstation:

macOS | Windows | Linux

Copy

```
$ export AWS_ACCESS_KEY_ID=<YOUR_ACCESS_KEY_ID>
$ export AWS_SECRET_ACCESS_KEY=<YOUR_SECRET_ACCESS_KEY>
```

Create project

Let's create your first Pulumi project, stack, and program. Pulumi projects and stacks organize Pulumi code. Projects are similar to GitHub repos and stacks are an instance of code with separate configuration. Projects can have multiple stacks for

different development environments or for different cloud configurations.

TypeScript | JavaScript | Python | Go | C# | Java | YAML

Copy

```
$ mkdir quickstart && cd quickstart
$ pulumi new aws-typescript
```

The `pulumi new` command creates a Pulumi project with basic scaffolding.

You will be asked for a **project name** and **project description**.

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

Enter a value or leave blank to accept the (default), and press <ENTER>.
Press ^C at any time to quit.

project name: (quickstart)
project description: (A minimal AWS Pulumi program)
Created project 'quickstart'
```

Then you will be asked for a **stack name**.

```
Please enter your desired stack name.
To create a stack in an organization, use the format <org-name>/<stack-na
stack name: (dev)
```

```
Created stack 'dev'
```

Finally, you will be prompted for a configuration value for the stack, specifically the AWS region.

```
aws:region: The AWS region to deploy into: (us-west-2)
Saved config
```

After the command completes, the project and stack will be ready.

Review project

Let's review some of the generated project files:

- `Pulumi.yaml` defines the project.
- `Pulumi.dev.yaml` contains configuration values for the stack you just initialized.
- `index.ts` is the Pulumi program that defines your stack resources.

TypeScript | JavaScript | Python | Go | C# | Java | YAML

Copy

```
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";
import * as awsx from "@pulumi/awsx";

// Create an AWS resource (S3 Bucket)
```

```
const bucket = new aws.s3.Bucket("my-bucket");

// Export the name of the bucket
export const bucketName = bucket.id;
```

This Pulumi program creates a new S3 bucket and exports the name of the bucket.

Copy

```
export const bucketName = bucket.id;
```

Deploy stack

Let's deploy the stack:

Copy

```
$ pulumi up
```

This command evaluates the program and determines what resources need updates. A preview is shown that outlines the changes that will be made when you run the update:

```
Previewing update (dev):

     Type                 Name            Plan
 +   pulumi:pulumi:Stack  quickstart-dev  create
 +   └─ aws:s3:Bucket     my-bucket       create
```

```
Resources:
    + 2 to create

Do you want to perform this update?
> yes
  no
  details
```

Once the preview has finished choose `yes` to create your new S3 bucket in AWS.

```
Do you want to perform this update? yes
Updating (dev):

     Type                 Name            Status
 +   pulumi:pulumi:Stack  quickstart-dev  created (4s)
 +   └─ aws:s3:Bucket     my-bucket       created (2s)

Outputs:
    bucketName: "my-bucket-58ce361"

Resources:
    + 2 created

Duration: 5s
```

Copy

```
$ pulumi stack output bucketName
```

Modify program

Now that your S3 bucket is provisioned, let's add a file to it. In the project directory, create a new file called `index.html` along with some content:

macOS | Windows | Linux

Copy

```
echo '<html>
    <body>
        <h1>Hello, Pulumi!</h1>
    </body>
</html>' > index.html
```

Open the program and add this file to the S3 bucket. This uses Pulumi's `FileAsset` resource to assign the content of the file to a new `BucketObject`:

TypeScript | JavaScript | Python | Go | C# | Java | YAML

In `index.ts`, create the `BucketObject` right after creating the bucket itself:

Copy

```
// Create an S3 Bucket object
```

```
const bucketObject = new aws.s3.BucketObject("index.html", {
    bucket: bucket.id,
    source: new pulumi.asset.FileAsset("./index.html")
});
```

This bucket object is part of the `Bucket` that we deployed earlier because we *reference* the bucket name in the properties of the bucket object.

We refer to this relationship as the `BucketObject` being a *child* resource of the S3 `Bucket` that is the *parent* resource.

Deploy changes

Deploy the changes by running `pulumi up` again.

Copy

```
$ pulumi up
```

Pulumi will run the `preview` step of the update, which computes the minimally disruptive change to achieve the desired state described by the program.

```
Previewing update (dev):

     Type                     Name            Plan
     pulumi:pulumi:Stack      quickstart-dev
 +   └─ aws:s3:BucketObject   index.html      create

Resources:
```

```
    + 1 to create
    2 unchanged

Do you want to perform this update?
> yes
  no
  details
```

Choose `yes` to proceed with the update and upload the `index.html` file to your bucket:

```
Do you want to perform this update? yes
Updating (dev):

     Type                    Name            Status
     pulumi:pulumi:Stack     quickstart-dev
 +   └─ aws:s3:BucketObject  index.html      created (0.98s)


Outputs:
    bucketName: "my-bucket-58ce361"

Resources:
    + 1 created
    2 unchanged

Duration: 3s
```

Verify the object was created in your bucket by checking the AWS Console or by running the following AWS CLI command:

Copy

```
$ aws s3 ls $(pulumi stack output bucketName)
```

Notice that your `index.html` file has been added to the bucket:

Copy

```
2023-04-20 17:01:06        118 index.html
```

Modify program again

Update the `Bucket` declaration to add a `website` property and make `index.html` the home page of the website:

TypeScript | JavaScript | Python | Go | C# | Java | YAML

Copy

```
const bucket = new aws.s3.Bucket("my-bucket", {
    website: {
        indexDocument: "index.html",
    },
});
```

Lastly, you'll make a few adjustments to make these resources accessible on the Internet.

For the bucket itself, you'll need two new resources: a `BucketOwnershipControls` resource, to define the bucket's file-ownership settings, and a `BucketPublicAccessBlock` resource to allow the bucket to be accessed publicly.

For the `BucketObject`, you'll need an access-control (ACL) setting of `public-read` to allow the page to be accessed anonymously (e.g., in a browser) and a content type of `text/html` to tell AWS to serve the file as a web page.

TypeScript | JavaScript | Python | Go | C# | Java | YAML

```
const ownershipControls = new aws.s3.BucketOwnershipControls("ownership-c
    bucket: bucket.id,
    rule: {
        objectOwnership: "ObjectWriter"
    }
});

const publicAccessBlock = new aws.s3.BucketPublicAccessBlock("public-acce
    bucket: bucket.id,
    blockPublicAcls: false,
});

const bucketObject = new aws.s3.BucketObject("index.html", {
    bucket: bucket.id,
    source: new pulumi.asset.FileAsset("./index.html"),
    contentType: "text/html",
    acl: "public-read",
}, { dependsOn: publicAccessBlock });
```

The `BucketObject` includes the Pulumi resource *option* `dependsOn`. This setting tells Pulumi that the `BucketObject` relies indirectly on the `BucketPublicAccessBlock`, which is responsible for enabling public access to its contents.

At the end of the program, export the bucket's endpoint URL:

Copy

```
export const bucketEndpoint = pulumi.interpolate`http://${bucket.websiteE
```

Deploy website

Update your stack to deploy these changes to AWS:

Copy

```
$ pulumi up
```

Review the preview of the changes before deploying:

```
Previewing update (dev):

     Type                                Name                Plan       I
     pulumi:pulumi:Stack                 quickstart-dev
 ~   ├─ aws:s3:Bucket                    my-bucket           update     [
 +   ├─ aws:s3:BucketOwnershipControls   ownership-controls  create
 +   ├─ aws:s3:BucketPublicAccessBlock   public-access-block create
 ~   └─ aws:s3:BucketObject              index.html          update     [

Outputs:
  + bucketEndpoint: output<string>

Resources:
    + 2 to create
    ~ 2 to update
    4 changes. 1 unchanged

Do you want to perform this update?
```

```
> yes
  no
  details
```

Choose `yes` to perform the deployment:

```
Do you want to perform this update? yes
Updating (dev):

     Type                                Name                Status
     pulumi:pulumi:Stack                 quickstart-dev
 ~   ├─ aws:s3:Bucket                    my-bucket           updated (3s)
 +   ├─ aws:s3:BucketOwnershipControls   ownership-controls  created (0.8
 +   ├─ aws:s3:BucketPublicAccessBlock   public-access-block created (1s)
 ~   └─ aws:s3:BucketObject              index.html          updated (0.5

Outputs:
  + bucketEndpoint: "http://my-bucket-dfd6bd0.s3-website-us-east-1.amazon
    bucketName    : "my-bucket-dfd6bd0"

Resources:
    + 2 created
    ~ 2 updated
    4 changes. 1 unchanged

Duration: 8s
```

Check out your new website at the URL or make a `curl` request:

```
$ curl $(pulumi stack output bucketEndpoint)
```

```
<html>
    <body>
        <h1>Hello, Pulumi!</h1>
    </body>
</html>
```

Destroy stack

Now that you've seen how to deploy changes to our program, let's clean up and tear down the resources that are part of your stack.

To destroy resources, run the following:

```
$ pulumi destroy
```

```
Previewing destroy (dev):

     Type                              Name                Plan
 -   pulumi:pulumi:Stack               quickstart-dev      delete
```

```
 -   ├─ aws:s3:BucketObject             index.html           delete
 -   ├─ aws:s3:BucketOwnershipControls  ownership-controls   delete
 -   ├─ aws:s3:BucketPublicAccessBlock  public-access-block  delete
 -   └─ aws:s3:Bucket                   my-bucket            delete

Outputs:
  - bucketEndpoint: "http://my-bucket-dfd6bd0.s3-website-us-east-1.amazon
  - bucketName    : "my-bucket-dfd6bd0"

Resources:
    - 5 to delete

Do you want to perform this destroy? yes
Destroying (dev):

     Type                               Name                 Status
 -   pulumi:pulumi:Stack                quickstart-dev       deleted
 -   ├─ aws:s3:BucketObject             index.html           deleted (1s)
 -   ├─ aws:s3:BucketPublicAccessBlock  public-access-block  deleted (0.2
 -   ├─ aws:s3:BucketOwnershipControls  ownership-controls   deleted (0.4
 -   └─ aws:s3:Bucket                   my-bucket            deleted (0.3

Outputs:
  - bucketEndpoint: "http://my-bucket-dfd6bd0.s3-website-us-east-1.amazon
  - bucketName    : "my-bucket-dfd6bd0"

Resources:
    - 5 deleted
```

```
Duration: 4s
```

To delete the stack itself, run `pulumi stack rm`. This removes the stack and the update history from Pulumi Cloud.

Next steps

Congrats! You've deployed your first project on AWS with Pulumi. Try a next step!

- Dive into Learn Pulumi for a comprehensive walkthrough of key Pulumi concepts in the context of a real-life application.
- Explore how-to guides: static websites, EC2 virtual machines, EKS clusters, Fargate containers, and serverless applications.
- Learn how Pulumi works from its architecture to key concepts, including stacks, state, configuration, and secrets.
- Read through the latest blog posts about using Pulumi with AWS.