# ***Pulumi - The new IAC Tool***

*Define cloud apps and infrastructure in your favorite language and deploy to any cloud.*

If you search the Internet for "**infrastructure-as-code**", it's pretty easy to come up with a list of the most popular tools: Chef, Ansible, Puppet, Terraform and the Freshman to IAC is **PULUMI**.

*It's 4am and the production server has gone down, You cannot keep calm ?*

Sure, how tough is it? Except that you'll probably need to recall what you've done a year ago to set up your environment, desperately trying to figure out what you've installed or implemented or configured. Finally you've gathered all your findings to closely replicate the environment.

Wouldn't it be nice to have something that manages all this configuration for you?
No, there are no robots coming to take over the devops team yet. I'm talking about using **Infrastructure-as- Code** to automatically and consistently manage infrastructure configuration.

## What is Infrastructure as Code (IAC)

Infrastructure as code simply means to convert your infrastructure into code, where it is managed by some kind of version control system, e.g., Git, and stored into a repository where you can manage it similar to your application.

## Pulumi - The new IAC Tool

From the last couple of days I'm trying my hands on Pulumi and the experience was very enlightening. Sharing some of the important and interesting findings with you all.

**Pulumi is a multi-language and multi-cloud development platform.** It lets you create all aspects of cloud programs using real languages ( Pulumi currently supports JavaScript, TypeScript, and Python, with more languages supported in the future) and real code, from infrastructure on up to the application itself. Just write programs and run them, and Pulumi figures out the rest.

Using real languages unlocks tremendous benefits:

- **Familiarity**: no need to learn new bespoke DSLs or YAML-based templating languages
- **Abstraction**: build bigger things out of smaller things.
- **Sharing and reuse**: we leverage existing language package managers to share and reuse these abstractions, either with the community, within your team, or both
- **Full control** : use the full power of your language, including async, loops, and conditionals

## Favorite things about Pulumi

1. **Multi-Language and Real language** - Using general purpose programming languages reduces the learning curve and makes it easier to express your configuration requirements.
2. **Developer Friendly:** Pulumi bridges the gap between Development and operations team not treating application code and infrastructure as separate things.
3. **Reusable Components**. Thanks to having a real language, we can build higher level abstractions.

   Below is one of my example code snippet using **Pulumi component** that create an instance of the Azure Resource Group to be used in other programs. You can find the full source code  that provisions Azure Load Balancer GitHub Code

```
class ResourceGroup extends pulumi.ComponentResource {
    constructor(resourceGroupName, location,path, opts)
    {
       super("az-pulumi-createstorageaccount:ResourceGroup",
resourceGroupName,location, {}, opts);

         console.log(`Resource Group ${resourceGroupName} : location
${location} `);
       // Create an Azure Resource Group
            const resourceGroup = new
azure.core.ResourceGroup(resourceGroupName,
             {
                 location:location,
             },

           {
              parent: this
           }
        );

         // Create a property for the resource group name that was created
        this.resourceGroupName = resourceGroup.name,
        this.location = location


         // For dependency tracking, register output properties for this
component
        this.registerOutputs({
            resourceGroupName: this.resourceGroupName,

        });

    }

}


module.exports.ResourceGroup = ResourceGroup;
```

This class can be instantiated as below:

```
// import the class
const resourceGroup = require("./create-resource-group.js");


// Create an Azure Resource Group
// Arguments : Resource group name and location
let azureResouceGroup = new resourceGroup.ResourceGroup("rgtest","EastUS");
```

**4. Multi Cloud -** Pulumi supports all major clouds — including AWS, Azure and Google Cloud, as well as Kubernetes clusters. This delivers a consolidated programming model and tools for managing cloud software anywhere. There’s no need to learn three different YAML dialects, and five different CLIs, just to get a simple container-based application stood up in production.

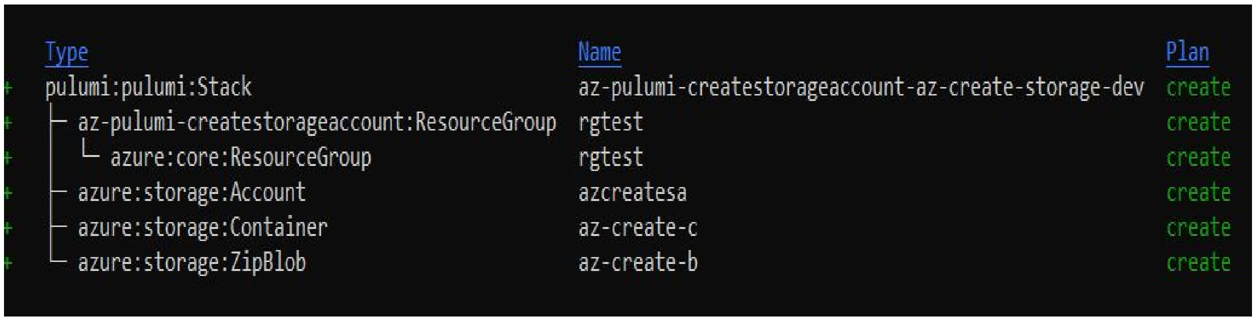
Let us create an storage account in AWS and Azure and upload objects using pulumi interface and compare them at length, how resources are allocated and provisioned in both the cloud platforms.

| Azure Storage Account | AWS Storage Account |
|---|---|
| // import packages and declare variables<br>….<br>… | *// import packages and declare variables*<br>….<br>… |
| // Create an Azure Resource Group<br>let azureResouceGroup = new resourceGroup.ResourceGroup("rgtest","EastUS");<br><br>// Create an Azure resource (Storage Account)<br>const storageAccountName = `${prefix.toLowerCase().replace(/-/g, "")}sa`;<br>const account = new azure.storage.Account(storageAccountName, {<br>    resourceGroupName: azureResouceGroup.resourceGroupName,<br>    location: azureResouceGroup.location,<br>    accountTier: "Standard",<br>    accountReplicationType: "LRS",<br>});<br><br>// Create a storage container<br>const storageContainer = new azure.storage.Container(`${prefix}-c`, {<br>    resourceGroupName: azureResouceGroup.resourceGroupName,<br>    storageAccountName: account.name,<br>    containerAccessType: "private",<br>}); | *// Create an AWS resource (S3 Bucket)*<br>**const** siteBucket = **new** aws.s3.Bucket("my-bucket",{<br>        website: {<br>    indexDocument: "index.html",<br>  }<br>});<br><br>**let** siteDir = "www"; *// directory for content files* |
| //For each file in the directory, create a blob | *// For each file in the directory, create an S3 object stored in `siteBucket`* |

```
for (let item of
require("fs").readdirSync("wwwroot")) {
  let filePath = require("path").join("www",
item);

 let  blob = new
azure.storage.Blob(item.name, {
    resourceGroupName:
azureResouceGroup.resourceGroupName,
    storageAccountName: account.name,
    storageContainerName:
storageContainer.name,
    source: new
pulumi.asset.FileAsset(filePath),
    contentType:mime.getType(filePath) ||
undefined
  });
}



// Create Signed URL
const blobUrl = signedBlobReadUrl(blob,
account, storageContainer);
```

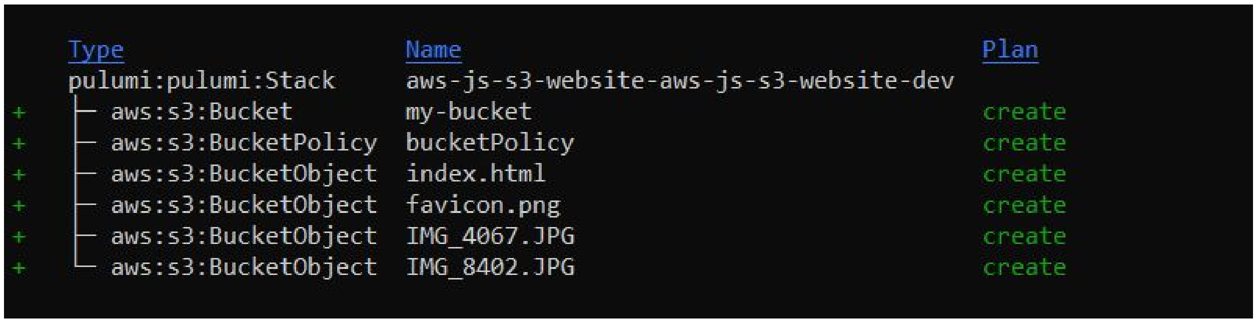
```
for (let item of
require("fs").readdirSync(siteDir)) {
  let filePath =
require("path").join(siteDir, item);
  let object = new aws.s3.BucketObject(item,
{
    bucket: siteBucket,
    source: new
pulumi.asset.FileAsset(filePath),     // use
FileAsset to point to a file
    contentType: mime.getType(filePath) ||
undefined, // set the MIME type of the file
  });
}

// Create an S3 Bucket Policy to
allow public read of all objects
in bucket
function
publicReadPolicyForBucket(bucketName) {
    return JSON.stringify({
        Version: "2012-10-17",
        Statement: [{
            Effect: "Allow",
            Principal: "*",
            Action: [
                "s3:GetObject"
            ],
            Resource: [

`arn:aws:s3:::${bucketName}/*` // policy
refers to bucket name explicitly
            ]
        }]
    })
}

// Set the access policy for the
bucket so all objects are
readable
let bucketPolicy = new
aws.s3.BucketPolicy("bucketPolicy", {
    bucket: siteBucket.bucket, // refer to
the bucket created earlier
    policy:
siteBucket.bucket.apply(publicReadPolicyForBu
cket) // use output property
```

|  |  |
|---|---|
|  | `` `siteBucket.bucket` `` <br> `});` <br><br> ***// Export the DNS name of the bucket*** <br> `exports.bucketName = siteBucket.bucketDomainName;` <br> `exports.websiteUrl = siteBucket.websiteEndpoint;` |
| Pulumi Plan for Azure Storage Account <br> **See Fig 1.1** | Pulumi Plan for S3 <br> **See Fig. 1.2** |



```
     Type                                             Name                                             Plan
 +   pulumi:pulumi:Stack                              az-pulumi-createstorageaccount-az-create-storage-dev create
 +   ├─ az-pulumi-createstorageaccount:ResourceGroup  rgtest                                           create
 +   │  └─ azure:core:ResourceGroup                   rgtest                                           create
 +   ├─ azure:storage:Account                         azcreatesa                                       create
 +   ├─ azure:storage:Container                       az-create-c                                      create
 +   └─ azure:storage:ZipBlob                         az-create-b                                      create
```

Fig. 1.1



```
     Type                      Name                                     Plan
     pulumi:pulumi:Stack       aws-js-s3-website-aws-js-s3-website-dev
 +   ├─ aws:s3:Bucket          my-bucket                                create
 +   ├─ aws:s3:BucketPolicy    bucketPolicy                             create
 +   ├─ aws:s3:BucketObject    index.html                               create
 +   ├─ aws:s3:BucketObject    favicon.png                              create
 +   ├─ aws:s3:BucketObject    IMG_4067.JPG                             create
 +   └─ aws:s3:BucketObject    IMG_8402.JPG                             create
```

Fig 1.2

Evident from the figure that almost similar provisioning  plan for the resources in different clouds, Azure and AWS and almost similar syntax for provisioning resources in  the multi cloud environment.

## Closing thoughts

I would like to close this a bit long post with a statement “**Cloud Renaissance for DevOps and Developers**” as called by the whole internet community. Building powerful cloud software will be more enjoyable, more productive, and more collaborative for the developers. Ofcourse, everything comes with a cost, exploring pulumi I find that Pulumi has a sort of  lack in documentation. Besides this, for developers to write IAC, the understanding of infrastructure is must.

I hope that this post has given you a better idea of the overall platform, approach, and unique strengths.

Happy Puluming :) !!!