Infrastructure as Code

Doing the same thing over and over again

• So far what we've done in AWS has been done “by hand”
• This is fine for development and experimentation
• Once you have things figured out however, you want to codify your infrastructure
  • AWS CLI
  • CloudFormation
  • Python SDK (boto3)
  • TerraForm

aws-cli

• On your EC2 instance, the AWS CLI is pre-installed
• You can install it on your laptop too
  • https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
You need IAM credentials from your AWS account to use the CLI
- Log in to AWS Academy
  - https://www.awsacademy.com/LMS_Login
- Start your AWS environment

Under AWS Details
- Click on the “Show” button for AWS CLI

Copy the contents of the expanded box into a new file in your user’s home directory, inside the hidden ~/.aws/ folder named credentials.
- See the following link for details on where to store these credentials.
  - https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
Infrastructure as Code

**aws-cli**

- Get some basic info about your credentials and make sure everything is working

  ```bash
  aws sts get-caller-identity
  ```

- Default output is JSON
- Can change to text or table

  ```bash
  aws sts get-caller-identity --output table
  ```
The aws-cli is a command line interface to the core AWS API.
Everything you can do with the Web Console, you can do with the API and CLI.

If you've already created an EC2 instance, you have a security group already configured. Let's find it's ID.

```
aws ec2 describe-security-groups --region us-east-1
```
Infrastructure as Code

CloudFormation
AWS CloudFormation
Amazon's first party Infrastructure as Code service

• Refers to both the templating syntax as well as the AWS service
• Create text file templates which can be repeatedly deployed
• A deployment is called a “stack”

AWS CloudFormation
Amazon's first party Infrastructure as Code service

• Templates can be JSON or YAML formatted text files
• Top level sections: Parameters, Resources, Outputs and others
• Most data is basic key/value pairs
• YAML doesn't require you to quote every string

AWS CloudFormation
Infrastructure as Code service

• Templates can be uploaded to the AWS web console and deployed
<table>
<thead>
<tr>
<th>AWS CloudFormation</th>
<th>Infrastructure as Code service</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>• Stack changes can be previewed before deployment to see what resources will be created or modified</strong></td>
<td></td>
</tr>
<tr>
<td><strong>• Can watch the progress of the stack deployment</strong></td>
<td></td>
</tr>
<tr>
<td><strong>• If anything fails, CloudFormation can either leave things in place and broken so you can examine things, or it can roll back all your changes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>• Stacks can be updated over time</strong></td>
<td></td>
</tr>
<tr>
<td><strong>• Stacks can be completely deleted when you’re finished with it</strong></td>
<td></td>
</tr>
</tbody>
</table>
### AWS Language SDKs

**Software Development Kit**

- AWS Provides many ways to interact with its API
- RAW REST API
- AWS Web Console
- AWS CLI
- Programming Language SDKs

---

### AWS Language SDKs

**Programming Language SDKs**

- Python
- JavaScript
- Node.js
- Java
- Go
- C++
- .NET
- Ruby
- Rust
- Swift

https://aws.amazon.com/developer/tools/
Python SDK - boto3

Authentication

- Just like the aws-cli, if you’re making AWS API calls from outside of an AWS account, you need credentials
- The boto3 SDK knows to look for your [default] credentials from your ~/.aws/credentials file
- If you got the aws-cli working, then running python code from your laptop will also work
- If you want to run your python code inside of a container, you need to get credentials in to the container

Python SDK - boto3

Create an EC2 Instance

- The SDK documentation is essential


Python SDK - boto3

Two SDK Models

- Each Service in the boto3 library presents two different interface models
  - client model
    - Closely maps directly to the AWS API itself / aws-cli
    - Returns dictionary mappings of the raw JSON responses
  - resource model
    - More object oriented
    - Returns python objects
We want the `boto3.client` for EC2 to start.

Documentation provides a comprehensive list of all the properties and methods available.

Many examples

I almost always start here first, then go off to more broad searches if I need to.

```python
import boto3
from botocore.config import Config

conf = Config(region_name='us-east-1')
ec2 = boto3.client('ec2', config=conf)
call_result = ec2.run_instances(
    ImageId='ami-026b57f3c383c2eec',
    InstanceType='t2.micro',
    MinCount=1,
    MaxCount=1,
    KeyName='vockey',
    NetworkInterfaces=[
        {
            'DeviceIndex': 0,
            'SubnetId': 'subnet-0cea5865199d0595c',
            'Groups': ["sg-07f090fb54ae76532"],
            'AssociatePublicIpAddress': True,
        }
    ],
)
print(call_result)
```

Response is a generic python dictionary with key/value pairs.

Useful if you only need cursory interaction with the resource after you create it.

```
{'Groups': [], 'Instances': [{'AmiLaunchIndex': 0, 'ImageId': 'ami-026b57f3c383c2eec', 'InstanceId': 'i-0aafad17c8d49bf7a', 'InstanceType': 't2.micro', 'KeyName': 'vockey', 'LaunchTime': datetime.datetime(2022, 10, 23, 20, 45, 33, tzinfo=tzutc()), 'Monitoring': {'State': 'disabled'}, 'Placement': {'AvailabilityZone': 'us-east-1e', 'GroupName': '', 'Tenancy': 'default'}, 'PrivateDnsName': 'ip-172-31-63-12.ec2.internal', 'PrivateIpAddress': '172.31.63.12', 'ProductCodes': [], 'PublicDnsName': '', 'State': {'Code': 0, 'Name': 'pending'}, 'StateTransitionReason': '', 'SubnetId': 'subnet-0cea5865199d0595c', 'VpcId': 'vpc-0b1989c3c4cd0263a', 'Architecture': 'x86_64', 'BlockDeviceMappings': [], 'ClientToken': 'c259d26c-0056-41bb-96ec-a5b3cb42857d', 'EbsOptimized': False, 'EnaSupport': True, 'Hypervisor': 'xen', 'NetworkInterfaces': [{'Attachment': {'AttachTime': datetime.datetime(2022, 10, 23, 20, 45, 33, tzinfo=tzutc()), 'AttachmentId': 'eni-attach-0d2727e02df2c2ea0', 'DeleteOnTermination': True, 'DeviceIndex': 0, 'Status': 'attaching', 'NetworkCardIndex': 0}, 'Description': '', 'Groups': [{'GroupName': 'launch-wizard-1', 'GroupId': 'sg-07f090fb54ae76532'}], 'Ipv6Addresses': [], 'MacAddress': '06:3d:1a:e8:79:37', 'NetworkInterfaceId': 'eni-0a8b52f5531047feb', 'OwnerId': '561707296892', 'PrivateDnsName': 'ip-172-31-63-12.ec2.internal', 'PrivateIpAddress': '172.31.63.12', 'PrivateIpAddresses': [{'Primary': True, 'PrivateDnsName': 'ip-172-31-63-12.ec2.internal', 'PrivateIpAddress': '172.31.63.12'}], 'SourceDestCheck': True, 'Status': 'in-use', 'SubnetId': 'subnet-0cea5865199d0595c', 'VpcId': 'vpc-0b1989c3c4cd0263a', 'InterfaceType': 'interface'}], 'RootDeviceName': '/dev/xvda', 'RootDeviceType': 'ebs', 'SecurityGroups': [{'GroupName': 'launch-wizard-1', 'GroupId': 'sg-07f090fb54ae76532'}], 'SourceDestCheck': True, 'StateReason': {'Code': 'pending', 'Message': 'pending'}, 'VirtualizationType': 'hvm', 'CpuOptions': {'CoreCount': 1, 'ThreadsPerCore': 1}, 'CapacityReservationSpecification': {'CapacityReservationPreference': 'open'}, 'MetadataOptions': {'State': 'pending', 'HttpTokens': 'optional', 'HttpPutResponseHopLimit': 1, 'HttpEndpoint': 'enabled', 'call_result["InstanceId"]'}}
```
**Python SDK - boto3**

Terminate an EC2 Instance

- The resource model allows us to manipulate objects
- Here we first create an EC2 instance object in our code
- Because it is a python object, we can easily inspect attributes and call methods

```python
import boto3
from botocore.config import Config

conf = Config(region_name='us-east-1')
ec2 = boto3.resource('ec2', config=conf)
instance = ec2.Instance('i-0aafad17c8d49bf7a')

print(instance.state)
instance.terminate()
instance.wait_until_terminated()
print(instance.state)
```

```bash
$ python3 ec2-terminate.py
{'Code': 16, 'Name': 'running'}
{'Code': 48, 'Name': 'terminated'}
```

**Terraform**

Open-Source Multi-Provider Templating System

- Open-source tool spooned by HashiCorp
- Supports multiple cloud providers
- Has its own language that is similar to JSON, but supports comments, and built-in references and functions
- Install the `terraform` CLI tool

```terraform
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }
  required_version = ">= 1.2.0"
}

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

# Create a basic EC2 Instance
resource "aws_instance" "app_server" {
  ami                        = "ami-026b57f3c383c2eec"
  instance_type              = "t2.micro"
  associate_public_ip_address = true
  subnet_id                   = "subnet-0cea5865199d0595c"
  security_groups             = ["sg-07f090fb54ae76532"]
  key_name                    = "vockey"
}
```

[https://www.terraform.io/downloads](https://www.terraform.io/downloads)
Comparison
So what should you use?

• “It depends”
• Each method presented here has advantages and disadvantages
• Significant overlap between tools
• Can always start simple with a shell script running aws-cli commands. As that becomes cumbersome move to either boto3 or CloudFormation/Terraform depending on needs

Version Control Systems
Basicly git

Version Control Systems
It’s just git these days

• A version control system aims to keep track of all the changes made to any of your project files
• Mostly focused on text files
  • Binary files can be versioned, but they are harder to look at differences
• If you’re dealing with text files that might change, you should probably use a version control system
Version Control Systems
It's just git these days

- Years ago there used to be several competing version control systems
- These days the industry has basically settled on git
- Originally developed to manage the Linux kernel.
- Designed as a distributed version control system with direct peer-to-peer capabilities
  - Very rarely used in practice
- Hub & spoke model of older version control systems gave rise to GitHub
  - GitHub ≠ git!

The git Version Control System

- A git repository is basically a folder with a hidden .git directory in it which contains state and history
- Files added to the folder can then be added to change sets and committed to the repository
- All of this can happen locally on your computer without needing a server
- If you want to use a service like GitHub, your local repository can be pushed to a remote repository hosted on GitHub.

git basics
Setup

- https://git-scm.com/downloads
- Many platforms have git installed by default
  - macOS has git as part of Xcode
  - Windows installer
  - Linux package managers
git basics
Setup

- Initial setup commands
- Set your default branch name
- Set your user.name
- Set your user.email

Create some files
- `git init` to initialize your current folder as a repository

Use `git status` to show what changes are not in your repository
git basics
Setup
• Use `git add` to stage new or changed files

• Use `git commit` to commit all staged changes to the repository along with a change log message
  
  • Message can be provided inline with the `-m` option, or with a CLI text editor like `vim`

• Tools like VS Code have built-in support for `git`
  
  • Add and commit changed files directly in VS Code GUI
Committing changes to files that are already tracked can be done with the -a option on the `git commit` command.

VS Code also has built-in support for showing differences between files as you work.

Can see a history of commits with the `git log` command.

Also shows up in the VS Code Timeline pane.