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
Infrastructure as Code
aws-cli

- You need IAM credentials from your AWS account to use the CLI

- Log in to AWS Academy
  - https://awsacademy.instructure.com/login/canvas
  - 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 lecture slides 07-aws for walkthrough of setting up credentials in VS Code

http://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
Who are you?

- 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
```
Infrastructure as Code

aws-cli

- 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

```
aws ec2 describe-security-groups --region us-east-1
```

Infrastructure as Code

aws-cli

- If you've already created an EC2 instance, you have a security group already configured. Let's find its ID
• Looking up information is fine, but can we make things?
• Let's deploy a new EC2 instance from the command line.
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”

- 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

---

# EC2 Basic CloudFormation Deployment
# -----------------------------------------

# This CloudFormation template will deploy a single EC2 instance with
# its own security group.

AWSTemplateFormatVersion: "2010-09-09"

Parameters:
  HostName:
    Type: String
    Description: "Enter the name of the host or service, ie 'Civil Engineering Structures App', or 'UITS Cloud Services Testing', etc."

Resources:
  Ec2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !Ref AmazonLinuxAmi
      KeyName: !Ref KeyName
      InstanceType: !Ref InstanceType
      IamInstanceProfile: !Ref InstanceProfile

  InstanceSecurityGroup:
    Type: "AWS::EC2::SecurityGroup"
    Properties:
      GroupDescription: "Allow ssh to client host"
      VpcId: !Ref VPCID
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: "0.0.0.0/0"

Outputs:
  InstancePublicIP:
    Condition: AssignPublicIPCondition
    Description: "The Public IP address of the instance"
    Value: !GetAtt Ec2Instance.PublicIp
AWS CloudFormation
Infrastructure as Code service

- Templates can be uploaded to the AWS web console and deployed

- Stack changes can be previewed before deployment to see what resources will be created or modified

- Can watch the progress of the stack deployment
  - 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
AWS CloudFormation
Infrastructure as Code service

- Stacks can be updated over time
- Stacks can be completely deleted when you’re finished with it

AWS Python SDK - boto3

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

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

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 `~/.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

https://aws.amazon.com/developer/tools/

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

Create an EC2 Instance

- 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="t3.micro",
    MinCount=1,
    MaxCount=1,
   KeyName="vockey",
    NetworkInterfaces=
    [
        {
            "DeviceIndex": 0,
            "SubnetId": "subnet-0cea5865199d0595c",
            "Groups": ["sg-07f090fb54ae76532"],
            "AssociatePublicIpAddress": True
        }
    ],
)
print(call_result)
```
Python SDK - boto3
Create an EC2 Instance

- Response is a generic python dictionary with key/value pairs
- Useful if you only need cursory interaction with the resource after you create it

```python
response = call_result
print(response)
```
**Terraform**
Create an EC2 Instance

- 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 = "~> 4.1.16"
    }
  }
  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**
Basically `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

- 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
- Use `git status` to show what changes are not in your repository
- 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`
git basics
Setup
• 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
**git basics**

**Setup**

- Can see a history of commits with the `git log` command
- Also shows up in the VS Code Timeline pane