3. A. IAC-Getting Started

Getting Started with CloudFormation

Cloud services are broadly categorized as *Software as a Service (SaaS)*, *Platform as a Service (PaaS)*, or *Infrastructure as a Service (IaaS)*.

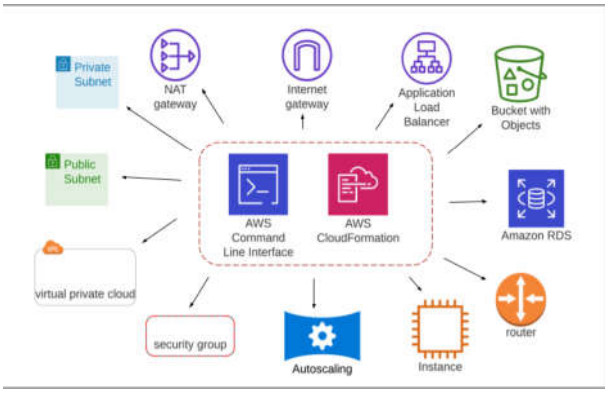
Cloud is a *collection of geographically distributed data centers that are logically grouped into regions and availability-zones*.

The IaaS allows a user to:

- provision Virtual Machines (VMs),
- set up networks (VPC),
- create subnets, and associate necessary security features. Further,
- VMs can be attached to storage volumes, different networks, or servers.

All the resources mentioned above are referred to as Infrastructure.

The diagram below shows various cloud infrastructure resources that we will learn to provision on the AWS cloud using the *AWS command line* and *CloudFormation tools*. These are the primary resources that are required to build a web app architecture.



Cloud Devops is a topic of culture shifts where we can have code that actually deploys physical servers instead of software - or a little bit of both, it’s the combination of development and operation.

1. Setup environment and establish foundation
2. Understand Architecture Diagram
3. Turn Diagram into Script (Infrastructure as Code) and Networking Infrastructure
4. Deploy servers on top of networking infrastructure (Security Groups)
5. Databases and Internal storages

Top

What is DevOps?

In the past, we have `development` and `operations` as separate entities. This is less than ideal. Developers develop code that needs to be migrated to Production. Therefore, (IT) operations have to communicate with developers. There usually communication gaps between these 2 groups of people.
The job of `development` and `operations` now, in the world of `DevOps` is to integrate each other and get everything working together as a team and not as two separate entities.

`DevOps` is the combination of industry best practices, and set of tools that improve an organization's ability to:

- Increase the *speed of software delivery*
- Increases the *speed of software evolution*
- Have *better reliability* of the software
- Have *scalability using automation*,
- *Improved collaboration* among teams.

In other words, these tools enable a company to showcase industry best practices in software development.

Top

Why we need DevOps

DevOpos tries to solve the following issues:

- Unpredictable deployments
  - Deployment done by IT Operations who are not involved in the creation of the software.
  - They receive specifications from developers and they do their best to get things running.
  - This can often be a problem as there may be gaps and they do their best to get it up and running. Some configurations may not be documented and therefore, this process is not repeatable.
- Mismatched environments (development doesn't match production)
  - There are often scenarios where software works perfectly in the development environment but when it comes to actual deployment, there are deviations (performance, working of the software)
  - Ideally we want perfectly matching environments.

- Configuration Drift
  - configurations change over time and are no longer the same between production and development.
  - changes may be due to temporary fixes put in place by IT Operations (Support Team) that are not fedback to version control.

Top

Benefits of DevOps

DevOps offers a set of `best-practices` and `tools` to solve problems meantioned above.

`Configuration Management` tools such as `Chef`, `Puppet` and `Ansible` are tools designed to deploy and manage configuration changes to servers, could be 1, 100 or 1000 at the same time. These changes are written as code.

- You write this code
- You put it in a repository just like source code from developers
- This allowed the operational people to take this code and make a `patch` or a necessary configuration change to servers.
- Once these are tested and are working, you can apply this same change to the development environment avoiding `configuration-drift`

`Deployments` are done in a very *automated* and *predictable* fashion. You could have development team along with operations create a *deployment automation task* and this task is going to deploy not only to `test` but also to `production` - is just going to be different paramters for `credentials` and `environment variables` but the process is necessary is going to be the same.
The predictability of this deployment makes everything a world of difference because now you have code that you know that you don't care which person deploys it, it's going to work consistently regardless who deploys it.

`Continuous Integration / Continuous Deployment (CI/CD)` leverages tools such as `Jenkins` or `Circle CI` helps developers to push features out a lot faster.
When we have a `continuous-integration-pipeline`, we continously monitor the repository. Every time the developers check-in new code, this will trigger an automatic process that's going to continuously build, perform unit test, and possible deploy as well to a development environment or a test environment the latest version of the software.
Now, once this process goes on - because usually it takes serveral days or weeks for developers to create new features and test it properly - once this is done we end up running the exact same continuous deployment pipeline all the way to production. The deployment will trigger those necessary steps to get the software properly running in production, repeatable consistently (predictability).

In the DevOps model, `development` and `operations` teams are *merged into a single team*. These DevOps teams use a few tools and best practices that deploy and manage configuration changes to servers. Stackexchange has a discussion post detailing the difference between DevOps tools vs. Software Configuration Tools.

The most important practices are:

- `Continuous Integration / Continuous Delivery (CI/CD)` - new features are automatically deployed with all the required dependencies.
- `Infrastructure as Code (IaaC)` - configuration and management of cloud infrastructure using re-usable scripts.

Other prevalent practices are:

- Microservices
- Monitoring and Logging
- Communication and Collaboration

Glossary

1. `Continuous Integration Continuous Deployment (CI/CD)`: Tracks the development workflow from testing through production. Continuous integration is the process flow of testing any change made to your development flow, while continuous deployment tracks those changes through to staging and production systems. You may like to read this article by Atlassian.com that describes CI/CD in detail.
2. `Infrastructure as code (IaaC)`: Provision and manages the cloud-infrastructure by using scripts. These scripts can be written in YAML or JSON format. These scripts ensure that the same architecture can be re-built multiple numbers of times. These scripts are particularly useful in enterprise applications and different environments - dev, prod, or test. Read more here.

Top

Setup Tools

- AWS `CloudFormation` is a tool to:
  - configure
  - deploy
  - manage

  the necessary infrastructure to do: pushing code along with the necessary configurations.
- We need a code editor (because CloudFormation supports scripting)
- GIT support for version control:
  - need the ability to roll-back some changes
- IAM user, providing us with API access - we won't need to use the Console.
- AWS CLI tool
- Use of us-west-2 (Oregon) for consistency
- Use JSON and YAML (Yet Another Markup Language)

Top

Creating Access Key ID for IAM User

Notes:
For Dev and Prod user accounts:

- In practice, Dev and DevOps members may have separate user accounts for the dev environment as opposed to the production environment.
- This makes it easier for developers by giving them wider privileges in the dev environment that would normally only be reserved for DevOps members in the production environment.

To setup the AWS CLI:
On command prompt:

```
aws configure
AWS Access Key ID [None]: [Enter here]
AWS Secret Access Key [None]: [Entery here]
Default region name [None]: us-west-2 (Enter region here)
Default output format [None]:
```

Test by using command:

```
aws s3 ls
```

You can have max 2 keys at any one time per user. You therefore can add an additional key to the same user.

Regarding the access keys, it's always best to:

- rotate them (change them) frequently
- Make them inactive if they won't be used for a while

Top

Adding Additional Key

Additional Access Keys

Note that each user can have up to 2 access keys at the same time.

Why Making Keys Inactive is a Better Choice

You may make your access key temporarily inactive rather than destroying it and creating a new one. This may be helpful if you want to stop an automated process that uses that key (for example, a CI/CD process).

Top

Understanding CloudFormation

- CloudFormation is a declarative language, not an imperative language
  - You declare the resources that you want
  - You allow the 'logic' behind it to take care of everything

- CloudFormation handles `resource dependencies` so that you don't have to specify which resource to start up before another.
  - There are cases where you can specify that a resource depends on another resource, but *ideally, you'll let CloudFormation take care of dependencies*.
- `VPC` is the *smallest unit of resource*.

Declarative languages: These languages specify what you want, without requiring you to specify how to get it. An example of a popular declarative language is `SQL`.
Imperative languages: These languages use statements to change the state of the program.

Additional resources:
Here is the Wikipedia page describing the differences between the two.
Here is a Stack Overflow thread describing imperative languages.

Top

Getting Started with CloudFormation

CloudFormation supports either YAML or JSON format. YAML is the preferred format.
Visual Studio Code allows for creation of properly formatted YAML files. YAML very picky about the use of tabs, indentation and spaces.

```
Description: >
    Martin Simanjuntak / Udacity
    This template deploys a VPC
Resources:
    VPC:    # VPC is just a name or label here
        Type: AWS::EC2::VPC    # Type is what matters
        Properties:
            CidrBlock: 10.0.0.0/16
            EnableDnsHostnames: true
```

We can add more resources to the same file. However it can easily become a very long / big declaration file.
What we want is to create several files and group them based on either:

- logic, e.g. one for servers, one for networks, or by
- ownership, e.g. development environment files created by software developers and production resources (production databases and configurations) created by IT Operations.
- Everything will come together in the `deployment-pipeline`.

So, it's a best practice to group resources based on what works best for your company.

Notes:

- Having scripts specific for `networking` and other scripts specific to `EC2 Servers` or `Databases` keep your scripts *small*, and *easily shared across teams with different skill sets*, such as *database administrators* and *network experts*.

YAML and JSON

- `YAML` and `JSON` file formats are both supported in CloudFormation, but *YAML is the industry preferred version* that's used for `AWS` and other cloud providers (`Azure`, `Google Cloud Platform`).
- An important note about YAML files: the whitespace indentation matters! We recommend that you use four white spaces for each indentation.

Glossary in CloudFormation scripts

- `Name`: A name you want to give to the resource (does this have to be unique across all resource types?)
- `Type`: Specifies the actual hardware resource that you're deploying.
- `Properties`: Specifies configuration options for your resource. Think of these as all the drop-down menus and checkbox options that you would see in the AWS console if you were to request the resource manually.
- `Stack`: A stack is a group of resources. These are the resources that you want to deploy, and that are specified in the YAML file.

Best practices

Coding best practice: Create separate files to organize your code. You can either create separate files for similar resources or create files for each developer who uses those resources.

Documentation for CloudFormation syntax

You don't need to memorize the code that you need for each resource. You can find sample code in the documentation for CloudFormation for examples of how to write your CloudFormation scripts.

Top

Testing CloudFormation

- In the terminal, use your yml code to request the resources:

```
aws cloudformation create-stack
--stack-name myfirsttest                # stack-name
--region us-west-2                      # region name
--template-body file://testcfn.yml      # yml file-name
```

- Alternatively, a `shell-script` (.sh) can be used:

```
aws cloudformation create-stack
--stack-name $1
--template-body file://$2
--parameters file://$3
--capabilities "CAPABILITY_IAM" "CAPABILITY_NAMED_IAM"
--region=us-west-2
```

where `$1`, `$2`, and `$3` can be replaced with actual values.

Note: The `--parameters` and `--capabilities` options will be learnt in the upcoming lesson.

- You can also try a `batch script` (.bat) with a similar syntax, except that the actual values can be written as `%1` instead as `$1`.
- You may also want to use `update-stack` when you want to update an existing stack instead of destroying your stack and creating a new one every time changes are required. The syntax is similar:

```
aws cloudformation update-stack
--stack-name $1
--template-body file://$2
--parameters file://$3
--capabilities "CAPABILITY_IAM" "CAPABILITY_NAMED_IAM"
--region=us-west-2
```

Top