**Infrastructure as Code (IaC)** is the practice of managing and provisioning computing infrastructure through machine-readable configuration files rather than through physical hardware configuration or interactive configuration tools. Here are examples of how IaC can be applied:

**Example 1: Terraform (for AWS Infrastructure)**

Terraform is a popular open-source IaC tool that allows you to define cloud resources such as servers, databases, and networking using configuration files.

**Terraform Example (Creating an AWS EC2 instance):**

hcl

Copy code

# Define the provider (in this case, AWS)

provider "aws" {

region = "us-west-2"

}

# Define an AWS EC2 instance

resource "aws\_instance" "my\_server" {

ami = "ami-0c55b159cbfafe1f0" # Amazon Machine Image (AMI)

instance\_type = "t2.micro" # Instance type (small, low-cost)

tags = {

Name = "MyTerraformServer"

}

}

# Outputs the public IP of the instance after creation

output "instance\_ip" {

value = aws\_instance.my\_server.public\_ip

}

In this Terraform script:

* The AWS provider is specified.
* An EC2 instance (a virtual server) is provisioned with a specific Amazon Machine Image (AMI) and instance type.
* The instance is tagged with a name, and once it's deployed, its public IP is output.

**Example 2: AWS CloudFormation (for AWS Infrastructure)**

AWS CloudFormation is a service that provides IaC by using YAML or JSON files to describe AWS resources.

**CloudFormation Example (Creating an S3 Bucket):**

yaml

Copy code

Resources:

MyS3Bucket:

Type: "AWS::S3::Bucket"

Properties:

BucketName: "my-example-bucket"

In this CloudFormation YAML:

* An AWS S3 bucket is created with the name "my-example-bucket."

**Example 3: Azure Resource Manager (ARM) Template (for Azure Infrastructure)**

ARM templates are JSON files that define resources and services in Azure.

**ARM Template Example (Creating a Virtual Machine):**

json

Copy code

{

"$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",

"contentVersion": "1.0.0.0",

"resources": [

{

"type": "Microsoft.Compute/virtualMachines",

"apiVersion": "2019-03-01",

"name": "myVM",

"location": "[resourceGroup().location]",

"properties": {

"hardwareProfile": {

"vmSize": "Standard\_DS1\_v2"

},

"storageProfile": {

"imageReference": {

"publisher": "Canonical",

"offer": "UbuntuServer",

"sku": "18.04-LTS",

"version": "latest"

}

},

"osProfile": {

"computerName": "myVM",

"adminUsername": "azureuser",

"adminPassword": "Password123!"

},

"networkProfile": {

"networkInterfaces": [

{

"id": "[resourceId('Microsoft.Network/networkInterfaces', 'myNIC')]"

}

]

}

}

}

]

}

This ARM template creates a Virtual Machine in Azure with the following:

* An Ubuntu server image.
* An instance size of **Standard\_DS1\_v2**.
* A network interface.

**Example 4: Ansible Playbook (for Configuration Management)**

Ansible uses YAML-based playbooks to define configurations or manage infrastructure.

**Ansible Example (Installing NGINX on a Server):**

yaml

Copy code

---

- name: Install NGINX on Ubuntu

hosts: webservers

become: yes

tasks:

- name: Update apt repo and cache

apt: update\_cache=yes

- name: Install NGINX

apt: name=nginx state=present

In this Ansible playbook:

* The **apt** package manager is used to install **NGINX** on servers in the "webservers" group.
* Ansible handles the configuration remotely by executing these commands on each server in the group.

**Example 5: Docker Compose (for Container Orchestration)**

Docker Compose allows you to define and run multi-container Docker applications using YAML files.

**Docker Compose Example (Defining a Web Application and Database):**

yaml

Copy code

version: '3'

services:

web:

image: nginx:latest

ports:

- "80:80"

volumes:

- ./html:/usr/share/nginx/html

db:

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: password

In this **Docker Compose** file:

* An NGINX web server and a MySQL database are defined as services.
* The **web** service binds the host's port 80 to the container's port 80 and mounts a local directory for serving static content.
* The **db** service sets up a MySQL container with a root password through environment variables.

**Benefits of Infrastructure as Code (IaC):**

1. **Version Control:** Infrastructure configurations can be stored in version control systems like Git, providing a history of changes and enabling collaboration.
2. **Consistency:** IaC ensures that environments (development, testing, production) are consistent and reproducible.
3. **Scalability:** You can easily scale infrastructure up or down based on predefined templates, enabling rapid deployment.
4. **Automation:** IaC removes the need for manual infrastructure provisioning, reducing errors and speeding up deployment.
5. **Cost Efficiency:** Automating infrastructure allows you to deploy only what you need, scaling resources dynamically and avoiding the cost of over-provisioning.

Student Information System

Create three services, Profile, Course and Attendance.

Activity (Create S3, Create a Virtual Env Elastic BeanStalk, Application setup, URL allocations…)

To do this in AWS you need a scripting called CloudFormation . This is called Infra as a code.

AWSTemplateFormatVersion: '2010-09-09'

Description: CloudFormation template to deploy a Spring Boot application using AWS Elastic Beanstalk.

AWSTemplateFormatVersion: '2010-09-09'

Description: CloudFormation template to deploy three microservices: Student Profile, Student Courses, and Student Attendance using Elastic Beanstalk.

Resources:

# S3 bucket to store the microservice JAR files

MicroserviceAppBucket:

Type: AWS::S3::Bucket

Properties:

BucketName: student-microservices-bucket

# Elastic Beanstalk Application for Student Profile Microservice

StudentProfileApp:

Type: AWS::ElasticBeanstalk::Application

Properties:

ApplicationName: "StudentProfileService"

Description: "Elastic Beanstalk application to deploy the Student Profile microservice"

# Elastic Beanstalk Application for Student Courses Microservice

StudentCoursesApp:

Type: AWS::ElasticBeanstalk::Application

Properties:

ApplicationName: "StudentCoursesService"

Description: "Elastic Beanstalk application to deploy the Student Courses microservice"

# Elastic Beanstalk Application for Student Attendance Microservice

StudentAttendanceApp:

Type: AWS::ElasticBeanstalk::Application

Properties:

ApplicationName: "StudentAttendanceService"

Description: "Elastic Beanstalk application to deploy the Student Attendance microservice"

# Elastic Beanstalk Environment for Student Profile

StudentProfileEnv:

Type: AWS::ElasticBeanstalk::Environment

Properties:

EnvironmentName: "StudentProfileEnv"

ApplicationName: !Ref StudentProfileApp

SolutionStackName: "64bit Amazon Linux 2 v3.3.3 running Corretto 11" # Java environment

OptionSettings:

- Namespace: "aws:autoscaling:launchconfiguration"

OptionName: "InstanceType"

Value: "t3.medium"

- Namespace: "aws:elasticbeanstalk:application:environment"

OptionName: "SPRING\_PROFILES\_ACTIVE"

Value: "prod"

VersionLabel: !Ref StudentProfileVersion

# Elastic Beanstalk Environment for Student Courses

StudentCoursesEnv:

Type: AWS::ElasticBeanstalk::Environment

Properties:

EnvironmentName: "StudentCoursesEnv"

ApplicationName: !Ref StudentCoursesApp

SolutionStackName: "64bit Amazon Linux 2 v3.3.3 running Corretto 11" # Java environment

OptionSettings:

- Namespace: "aws:autoscaling:launchconfiguration"

OptionName: "InstanceType"

Value: "t3.medium"

- Namespace: "aws:elasticbeanstalk:application:environment"

OptionName: "SPRING\_PROFILES\_ACTIVE"

Value: "prod"

VersionLabel: !Ref StudentCoursesVersion

# Elastic Beanstalk Environment for Student Attendance

StudentAttendanceEnv:

Type: AWS::ElasticBeanstalk::Environment

Properties:

EnvironmentName: "StudentAttendanceEnv"

ApplicationName: !Ref StudentAttendanceApp

SolutionStackName: "64bit Amazon Linux 2 v3.3.3 running Corretto 11" # Java environment

OptionSettings:

- Namespace: "aws:autoscaling:launchconfiguration"

OptionName: "InstanceType"

Value: "t3.medium"

- Namespace: "aws:elasticbeanstalk:application:environment"

OptionName: "SPRING\_PROFILES\_ACTIVE"

Value: "prod"

VersionLabel: !Ref StudentAttendanceVersion

# Application version for Student Profile

StudentProfileVersion:

Type: AWS::ElasticBeanstalk::ApplicationVersion

Properties:

ApplicationName: !Ref StudentProfileApp

Description: "Version 1.0 of the Student Profile microservice"

SourceBundle:

S3Bucket: !Ref MicroserviceAppBucket

S3Key: student-profile-service.jar # JAR file in the S3 bucket

# Application version for Student Courses

StudentCoursesVersion:

Type: AWS::ElasticBeanstalk::ApplicationVersion

Properties:

ApplicationName: !Ref StudentCoursesApp

Description: "Version 1.0 of the Student Courses microservice"

SourceBundle:

S3Bucket: !Ref MicroserviceAppBucket

S3Key: student-courses-service.jar # JAR file in the S3 bucket

# Application version for Student Attendance

StudentAttendanceVersion:

Type: AWS::ElasticBeanstalk::ApplicationVersion

Properties:

ApplicationName: !Ref StudentAttendanceApp

Description: "Version 1.0 of the Student Attendance microservice"

SourceBundle:

S3Bucket: !Ref MicroserviceAppBucket

S3Key: student-attendance-service.jar # JAR file in the S3 bucket

Outputs:

StudentProfileURL:

Description: "URL of the Student Profile microservice running on Elastic Beanstalk"

Value: !Sub "http://${StudentProfileEnv.EndpointURL}"

StudentCoursesURL:

Description: "URL of the Student Courses microservice running on Elastic Beanstalk"

Value: !Sub "http://${StudentCoursesEnv.EndpointURL}"

StudentAttendanceURL:

Description: "URL of the Student Attendance microservice running on Elastic Beanstalk"

Value: !Sub "http://${StudentAttendanceEnv.EndpointURL}"