Lab 2 :

Author: Badr TAJINI - DevOps Data for SWE - ESIEE - 2025

Lab: Managing Infrastructure as Code (IaC)

Introduction

In this lab, you will learn how to manage your infrastructure as code (IaC) using various tools and techniques. Instead of manually configuring and deploying your infrastructure, you will automate the process using code, making deployments faster, more consistent, and less error-prone.

You will explore four categories of IaC tools:

1. **Ad hoc scripts**: Using scripts to automate infrastructure tasks.
2. **Configuration management tools**: Using tools like Ansible to configure servers.
3. **Server templating tools**: Using tools like Packer to create server images.
4. **Provisioning tools**: Using tools like OpenTofu to provision infrastructure.

By the end of this lab, you will have hands-on experience with each of these tools and understand their strengths and weaknesses.

Objectives

- Understand the benefits of managing infrastructure as code.
- Deploy an EC2 instance using a Bash script.
- Deploy and configure an EC2 instance using Ansible.
- Create an Amazon Machine Image (AMI) using Packer.
- Deploy, update, and destroy an EC2 instance using OpenTofu.
- Use OpenTofu modules to manage infrastructure.
- Compare different IaC tools and understand when to use each.

Prerequisites

- AWS account with appropriate permissions.

- AWS CLI installed and configured.
- Ansible installed.
- Packer installed.
- OpenTofu installed.
- Basic knowledge of the command line and scripting.
- Node.js sample app (provided in the lab resources).
- Git installed (for module usage from GitHub).

> [!IMPORTANT]
>
> Reminder : You can create your own script on the basis of the lab or make use of the code examples shared with you throughout each chapter in DevOps Data for SWE course.

Section 1: Authenticating to AWS on the Command Line

Before you can interact with AWS services programmatically, you need to authenticate using AWS access keys.

Steps

1. **Create an Access Key**
    - Log in to the AWS Management Console.
    - Navigate to the **IAM Console**.
    - Click on **Users** and select your IAM user.
    - Click on the **Security credentials** tab.
    - Scroll down to **Access keys** and click **Create access key**.
    - Select **Command Line Interface (CLI)** as the use case.
    - Provide a description and click **Create access key**.
    - Follow the prompts to generate an access key. AWS will provide you with two pieces of information:
        - **Access Key ID**: A unique identifier for the key.
        - **Secret Access Key**: The actual key used for authentication.
    - **Important:** Ensure you securely store the **Secret Access Key** immediately, as it will not be accessible again. AWS recommends downloading the key details or storing them in a password manager.

- Use the access key credentials in your command-line interface or program to authenticate to AWS.

This text-only guide ensures users can follow along without needing the image.

2. **Set Environment Variables**

   On your local machine, set the following environment variables using the values from the previous step:

   ```
   export AWS_ACCESS_KEY_ID=YOUR_ACCESS_KEY_ID
   export AWS_SECRET_ACCESS_KEY=YOUR_SECRET_ACCESS_KEY
   ```

   Note: These environment variables apply only to the current terminal session. You'll need to set them again if you open a new terminal.

Section 2: Deploying an EC2 Instance Using a Bash Script

In this section, you'll write a Bash script to automate the deployment of an EC2 instance running a simple Node.js app.

Steps

1. **Set Up the Directory Structure**

   ```
   mkdir -p devops_base/td2/scripts/bash
   cd devops_base/td2/scripts/bash
   ```

2. **Create the User Data Script**

   Copy the user data script from the previous chapter :

   ```
   $ cp td1/scripts/ec2-user-data-script/user-data.sh td2/bash/
   ```

   Or create a new file called `user-data.sh` with the following content:

```
#!/usr/bin/env bash
# Install updates and Node.js, then run the app
yum update -y
curl -sL https://rpm.nodesource.com/setup_21.x | bash -
yum install -y nodejs

# Download the sample app (assuming it's hosted somewhere accessible)
curl -o /home/ec2-user/app.js https://raw.githubusercontent.com/your-repo/sample-app/main/app.js

# Start the Node.js app
nohup node /home/ec2-user/app.js &
```

3. **Create the Bash Script**

   Create a file called `deploy-ec2-instance.sh` with the following content:

```
#!/usr/bin/env bash

set -e

export AWS_DEFAULT_REGION="us-east-2"
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
user_data=$(cat "$SCRIPT_DIR/user-data.sh")

# Create a security group
security_group_id=$(aws ec2 create-security-group \
  --group-name "sample-app" \
  --description "Allow HTTP traffic into the sample app" \
  --output text \
  --query GroupId)

# Allow inbound HTTP traffic
aws ec2 authorize-security-group-ingress \
  --group-id "$security_group_id" \
  --protocol tcp \
  --port 80 \
  --cidr "0.0.0.0/0" > /dev/null

# Launch the EC2 instance
instance_id=$(aws ec2 run-instances \
  --image-id "ami-0900fe555666598a2" \
  --instance-type "t2.micro" \
  --security-group-ids "$security_group_id" \
  --user-data "$user_data" \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=sample-app}]'
\
  --output text \
  --query Instances[0].InstanceId)

# Wait for the instance to be in running state
aws ec2 wait instance-running --instance-ids "$instance_id"

# Get the public IP address
public_ip=$(aws ec2 describe-instances \
  --instance-ids "$instance_id" \
  --output text \
  --query 'Reservations[*].Instances[*].PublicIpAddress')

echo "Instance ID = $instance_id"
echo "Security Group ID = $security_group_id"
echo "Public IP = $public_ip"
```

4. **Make the Script Executable**

```
chmod u+x deploy-ec2-instance.sh
```

5. **Run the Script**

   ```
   ./deploy-ec2-instance.sh
   ```

   Wait for the script to complete. It will output the `Instance ID`, `Security Group ID`, and `Public IP`.

6. **Test the Deployment**

   - Wait a couple of minutes for the EC2 instance to initialize.
   - Open a web browser and navigate to `http://<Public IP>`.
   - You should see `Hello, World!` or the output of your Node.js app.

Exercise

- **Exercise 1**: What happens if you run the script a second time? Try it and observe the output. Explain why this happens.

  Hint: Consider resource names and uniqueness in AWS.

- **Exercise 2**: Modify the script to deploy multiple EC2 instances. How would you adjust the script to handle this?

Cleanup

- **Important**: To avoid incurring charges, terminate the EC2 instance when you're done.

```
aws ec2 terminate-instances --instance-ids <Instance ID>
aws ec2 delete-security-group --group-id <Security Group ID>
```

Section 3: Deploying an EC2 Instance Using Ansible

In this section, you'll use Ansible to deploy and configure an EC2 instance running the Node.js sample app.

Steps

1. **Set Up the Directory Structure**

```
mkdir -p devops_base/td2/scripts/ansible
cd devops_base/td2/scripts/ansible
```

2. **Create the Ansible Playbook for EC2 Deployment**

   Create a file called `create_ec2_instance_playbook.yml` with the following content:

```yaml
- name: Deploy an EC2 instance in AWS
  hosts: localhost
  gather_facts: no
  environment:
    AWS_REGION: us-east-2
  tasks:
    - name: Create security group
      amazon.aws.ec2_security_group:
        name: sample-app-ansible
        description: Allow HTTP and SSH traffic
        rules:
          - proto: tcp
            ports: [8080]
            cidr_ip: 0.0.0.0/0
          - proto: tcp
            ports: [22]
            cidr_ip: 0.0.0.0/0
      register: aws_security_group

    - name: Create a new EC2 key pair
      amazon.aws.ec2_key:
        name: ansible-ch2
        file_name: ansible-ch2.key
      no_log: true
      register: aws_ec2_key_pair

    - name: Create EC2 instance with Amazon Linux 2
      amazon.aws.ec2_instance:
        name: sample-app-ansible
        key_name: "{{ aws_ec2_key_pair.key.name }}"
        instance_type: t2.micro
        security_group: "{{ aws_security_group.group_id }}"
        image_id: ami-0900fe555666598a2
        wait: yes
        wait_timeout: 500
        tags:
          Ansible: ch2_instances
      register: ec2_instance
```

3. **Run the Playbook**

```
ansible-playbook -v create_ec2_instance_playbook.yml
```

4. **Set Up the Ansible Inventory**

   Create a file called `inventory.aws_ec2.yml` with the following content:

```yaml
plugin: amazon.aws.aws_ec2
regions:
  - us-east-2
keyed_groups:
  - key: tags.Ansible
leading_separator: ''
```

5. **Create Group Variables**

   Create a directory called `group_vars` and inside it, create a file called `ch2_instances.yml` with the following content:

```yaml
ansible_user: ec2-user
ansible_ssh_private_key_file: ansible-ch2.key
ansible_host_key_checking: false
```

6. **Create the Ansible Playbook for Configuration**

   Create a file called `configure_sample_app_playbook.yml` with the following content:

```yaml
- name: Configure the EC2 instance to run a sample app
  hosts: ch2_instances
  gather_facts: true
  become: true
  roles:
    - sample-app
```

7. **Create the Ansible Role**

   - Create the role directory structure:

```
mkdir -p roles/sample-app/{tasks,files}
```

   - Copy your `app.js` (Node.js sample app) into `roles/sample-app/files/`.

- Create `roles/sample-app/tasks/main.yml` with the following content:

```
- name: Add Node packages to yum
  shell: curl -fsSL https://rpm.nodesource.com/setup_14.x | bash -
  args:
    warn: false

- name: Install Node.js
  yum:
    name: nodejs

- name: Copy sample app
  copy:
    src: app.js
    dest: /home/ec2-user/app.js
    owner: ec2-user
    group: ec2-user
    mode: 0755

- name: Start sample app
  shell: nohup node /home/ec2-user/app.js &
  args:
    chdir: /home/ec2-user/
    creates: /tmp/node-app.pid
```

Note: The `creates` parameter helps make the task idempotent.

8. **Run the Configuration Playbook**

```
ansible-playbook -v -i inventory.aws_ec2.yml configure_sample_app_playbook.yml
```

9. **Test the Deployment**

- Retrieve the public DNS or IP of the EC2 instance from the Ansible output or AWS Console.
- Open a web browser and navigate to `http://<Public DNS>:8080`.
- You should see `Hello, World!` or the output of your Node.js app.

Exercise

- **Exercise 3**: What happens if you run the configuration playbook a second time? Observe and explain.

  Hint: Consider idempotency in Ansible tasks.

- **Exercise 4**: Modify the playbook to deploy and configure multiple EC2 instances. How would you adjust the playbook and inventory?

Cleanup

- **Important**: To avoid incurring charges, terminate the EC2 instance when you're done.

```
aws ec2 terminate-instances --instance-ids <Instance ID>
aws ec2 delete-key-pair --key-name ansible-ch2
aws ec2 delete-security-group --group-name sample-app-ansible
rm ansible-ch2.key
```

Section 4: Creating a VM Image Using Packer

In this section, you'll use Packer to create an Amazon Machine Image (AMI) with your Node.js sample app pre-installed.

Steps

1. **Set Up the Directory Structure**

```
mkdir -p devops_base/td2/scripts/packer
cd devops_base/td2/scripts/packer
```

2. **Copy the Node.js Sample App**

```
cp ../../td1/scripts/sample-app/app.js .
```

3. **Create the Packer Template**

   Create a file called `sample-app.pkr.hcl` with the following content:

```
packer {
  required_plugins {
    amazon = {
      version = ">= 1.3.1"
      source  = "github.com/hashicorp/amazon"
    }
  }
}

source "amazon-ebs" "amazon_linux" {
  ami_name        = "sample-app-packer-${uuidv4()}"
  ami_description = "Amazon Linux 2 AMI with a Node.js sample app."
  instance_type   = "t2.micro"
  region          = "us-east-2"
  source_ami      = "ami-0900fe555666598a2"
  ssh_username    = "ec2-user"
}

build {
  sources = ["source.amazon-ebs.amazon_linux"]

  provisioner "file" {
    source      = "app.js"
    destination = "/home/ec2-user/app.js"
  }

  provisioner "shell" {
    inline = [
      "curl -fsSL https://rpm.nodesource.com/setup_14.x | sudo bash -",
      "sudo yum install -y nodejs"
    ]
    pause_before = "30s"
  }
}
```

4. **Initialize Packer**

```
packer init sample-app.pkr.hcl
```

5. **Build the AMI**

```
packer build sample-app.pkr.hcl
```

Wait for the build to complete. Note the AMI ID outputted at the end.

Exercise

- **Exercise 5**: What happens if you run `packer build` a second time? Explain why.

  Hint: Consider the uniqueness of the `ami_name`.

- **Exercise 6**: Modify the Packer template to create images for another cloud provider or for local use (e.g., VirtualBox).

Cleanup

- **Note**: The AMI created will incur minimal charges if not used. If you wish, you can deregister the AMI from the AWS Console.

Section 5: Deploying, Updating, and Destroying an EC2 Instance Using OpenTofu

In this section, you'll use OpenTofu to deploy, update, and destroy an EC2 instance using the AMI created with Packer.

Steps

1. **Set Up the Directory Structure**

   ```
   mkdir -p devops_base/td2/scripts/tofu/ec2-instance
   cd devops_base/td2/scripts/tofu/ec2-instance
   ```

2. **Create the Main Configuration File**

   Create a file called `main.tf` with the following content:

```
provider "aws" {
  region = "us-east-2"
}

resource "aws_security_group" "sample_app" {
  name        = "sample-app-tofu"
  description = "Allow HTTP traffic into the sample app"
}

resource "aws_security_group_rule" "allow_http_inbound" {
  type              = "ingress"
  protocol          = "tcp"
  from_port         = 8080
  to_port           = 8080
  security_group_id = aws_security_group.sample_app.id
  cidr_blocks       = ["0.0.0.0/0"]
}

resource "aws_instance" "sample_app" {
  ami                    = var.ami_id
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.sample_app.id]
  user_data              = file("${path.module}/user-data.sh")

  tags = {
    Name = "sample-app-tofu"
  }
}
```

3. **Create the Variables File**

   Create a file called `variables.tf` with the following content:

```
variable "ami_id" {
  description = "The ID of the AMI to run."
  type        = string
}
```

4. **Create the User Data Script**

   Create a file called `user-data.sh` with the following content:

```
#!/usr/bin/env bash
nohup node /home/ec2-user/app.js &
```

5. **Create the Outputs File**

   Create a file called `outputs.tf` with the following content:

```
output "instance_id" {
  description = "The ID of the EC2 instance"
  value       = aws_instance.sample_app.id
}

output "security_group_id" {
  description = "The ID of the security group"
  value       = aws_security_group.sample_app.id
}

output "public_ip" {
  description = "The public IP of the EC2 instance"
  value       = aws_instance.sample_app.public_ip
}
```

6. **Initialize OpenTofu**

```
tofu init
```

7. **Apply the Configuration**

```
tofu apply
```

   - When prompted for `ami_id`, enter the AMI ID from the Packer build in the previous section.

8. **Test the Deployment**

   - Wait for the EC2 instance to initialize.
   - Retrieve the `public_ip` from the output.
   - Open a web browser and navigate to `http://<Public IP>:8080`.
   - You should see `Hello, World!` or the output of your Node.js app.

9. **Update the Configuration**

   - Modify `main.tf` to add a new tag:

```
tags = {
  Name = "sample-app-tofu"
  Test = "update"
}
```

- Run `tofu apply` again to apply the changes.

10. **Observe the Plan and Changes**
    - Before applying, OpenTofu will show you a plan detailing what will change.
    - Notice that it plans to update the EC2 instance in place to add the new tag.

11. **Destroy the Resources**

```
tofu destroy
```

- Confirm the destruction when prompted.

Exercise

- **Exercise 7**: What happens if you run `tofu apply` after the resources have been destroyed? Explain the behavior.
- **Exercise 8**: How would you modify the OpenTofu code to deploy multiple EC2 instances? Implement this change using a loop or by defining multiple resources.

Section 6: Deploying an EC2 Instance Using an OpenTofu Module

In this section, you'll refactor your OpenTofu configuration to use modules for better code reuse and organization.

Steps

1. **Set Up the Modules Directory**

```
mkdir -p ../../modules
mv ../ec2-instance ../../modules/ec2-instance
```

2. **Create the Root Module**

```
mkdir -p ../live/sample-app
cd ../live/sample-app
```

3. **Create the Main Configuration File**

   Create a file called `main.tf` with the following content:

```
provider "aws" {
  region = "us-east-2"
}

module "sample_app_1" {
  source = "../../modules/ec2-instance"

  ami_id = "YOUR_AMI_ID"  # Replace with your AMI ID
  name   = "sample-app-tofu-1"
}

module "sample_app_2" {
  source = "../../modules/ec2-instance"

  ami_id = "YOUR_AMI_ID"  # Replace with your AMI ID
  name   = "sample-app-tofu-2"
}
```

4. **Modify the Module to Use Variables**

   In `../../modules/ec2-instance/variables.tf`, add:

```
variable "name" {
  description = "The base name for the instance and all other resources"
  type        = string
}
```

5. **Update `main.tf` in the Module to Use `var.name`**

   Replace hardcoded names with `var.name`:

```
resource "aws_security_group" "sample_app" {
  name        = var.name
  description = "Allow HTTP traffic into ${var.name}"
}

resource "aws_instance" "sample_app" {
  # ...
  tags = {
    Name = var.name
  }
}
```

6. **Initialize OpenTofu**

```
tofu init
```

7. **Apply the Configuration**

```
tofu apply -var ami_id=YOUR_AMI_ID
```

   - Since the `ami_id` is an input variable in the module, you may need to pass it or define it accordingly.

8. **Test the Deployment**

   - After deployment, you will have two EC2 instances.
   - Retrieve their public IPs from the outputs.
   - Test both instances by navigating to `http://<Public IP>:8080`.

Exercise

- **Exercise 9**: Modify the module to accept additional parameters like `instance_type` and `port`. Update the root module to pass these parameters.
- **Exercise 10**: Use OpenTofu's `count` or `for_each` to deploy multiple instances without duplicating code in the root module.

Section 7: Using OpenTofu Modules from GitHub

In this section, you'll learn how to use OpenTofu modules hosted on GitHub.

Steps

1. **Modify the Module Source to Point to GitHub**

   In `main.tf`, update the module source:

   ```
   module "sample_app" {
     source = "github.com/your-github-username/your-repo.git//path/to/module"

     ami_id = "YOUR_AMI_ID"  # Replace with your AMI ID
     name   = "sample-app-from-github"
   }
   ```

   Note: Replace `your-github-username`, `your-repo`, and `path/to/module` with the actual values.

2. **Initialize OpenTofu**

   ```
   tofu init
   ```

3. **Apply the Configuration**

   ```
   tofu apply
   ```

4. **Test the Deployment**

   - Retrieve the public IP from the outputs.
   - Navigate to `http://<Public IP>:8080` to test the app.

Exercise

- **Exercise 11**: Explore the use of versioning with modules by specifying a specific Git tag or commit in the module source.
- **Exercise 12**: Find an OpenTofu module in the Terraform Registry or another public repository and use it in your configuration.

Cleanup

- **Important**: To avoid incurring charges, destroy the resources when you're done.

```
tofu destroy
```

Conclusion

In this lab, you explored different tools and techniques for managing infrastructure as code. You deployed and configured EC2 instances using Bash scripts, Ansible, Packer, and OpenTofu. You learned how to update and destroy resources using OpenTofu and how to use modules for better code reuse and organization. You also learned how to use modules from GitHub, which allows you to reuse and share infrastructure code.

By comparing these tools, you should now have a better understanding of their strengths and weaknesses and when to use each one in your infrastructure management workflow.
