SLIIT | DEPARTMENT OF COMPUTER SCIENCE & SOFTWARE ENGINEERING | FACULTY OF COMPUTING

Module - Current Trends in Software Engineering (SE4010) | 2025 | Semester 1

DevOps Lab – 2

1 - Kubernetes

Installing Kubernetes

1 – Docker Desktop – Download and install the version required for your OS (https://docs.docker.com/desktop/)
2 - https://kubernetes.io/docs/tasks/tools/ (Kind, Minikube)

If you are going with option 2 for installation, ensure that you also install **kubectl**

Tasks:

Part 1: Deploying Nginx using kubectl

1. **Create a Deployment**:
    - Command: **kubectl create deployment nginx --image=nginx:latest**
    - This command creates a deployment named 'nginx' using the **nginx:latest** Docker image.
2. **Verify the Deployment**:
    - Command: **kubectl get deployments**
    - This shows the status of the deployment.
3. **Expose the Deployment**:
    - Command: **kubectl expose deployment nginx --port=80 --type=NodePort**
    - This exposes Nginx on a dynamically assigned NodePort.
4. **Access Nginx**:
    - Command: **kubectl get service nginx**
    - Note the NodePort and access Nginx through **<NodeIP>:<NodePort>**.
5. **Remove resources**
    - kubectl delete deployment nginx
    - kubectl delete service nginx

Part 2: Using a Kubernetes Manifest File

1. **Create a Manifest File**:
    - Filename: **nginx-deployment.yaml**
    - Content:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
```

```
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

2. **Deploy using the Manifest File**:
   - Command: **kubectl apply -f nginx-deployment.yaml**
   - This creates the deployment defined in the manifest.
3. **Expose using a Service Manifest**:
   - Create a **nginx-service.yaml** with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      nodePort: 30007
```

   - Deploy the service: **kubectl apply -f nginx-service.yaml**.
4. **Access the Nginx Service**:
   - Find the service URL: **kubectl get service nginx**.
   - Access Nginx at **<NodeIP>:30007**. NodeIP can be localhost if you are running locally.

2 – HELM

Installing HELM

1 - https://helm.sh/docs/intro/install/

Tasks:

1. **Add the Helm Chart Repository**:
   - Command: **helm repo add bitnami https://charts.bitnami.com/bitnami**
   - This adds the Bitnami chart repository, which contains the Nginx chart.

2. **Install Nginx using Helm**:
    - Command: **helm install my-nginx bitnami/nginx**
    - This installs Nginx using the Bitnami Nginx chart with the release name 'my-nginx'.
3. **Verify the Deployment**:
    - Command: **helm list**
    - This will list all Helm releases including 'my-nginx'.
4. **Access Nginx**:
    - Command: **kubectl get svc my-nginx**
    - This shows the service created by Helm. Access Nginx using the external IP and port if available.
5. **Update the Deployment**:
    - Command: **helm upgrade my-nginx bitnami/nginx --set service.type=LoadBalancer**
    - This upgrades the release to change the service type to LoadBalancer.
6. **Uninstall the Release**:
    - Command: **helm uninstall my-nginx**
    - This removes the Nginx deployment from Kubernetes.
7. Refer ArtifactHub to view any publicly available HELM charts
    - https://artifacthub.io/

3 – Review and discuss

Refer the repository and review the CI/CD flow of the application. Discuss the issues and drawbacks of this approach and how GitOps approach tries to solve the drawbacks - https://github.com/rav94/devops-in-practice

4 – Terraform Basics

4.1 - Prerequisites

1. **Install Terraform**:
    - Download Terraform from the official Terraform downloads page.
    - Install it by following instructions specific to your operating system (Windows, macOS, Linux).
2. **Create or have access to a Cloud Provider account**: (Select which ever cloud provider that you have access to)
    - **AWS**: AWS Free Tier
    - **Azure**: Azure Free Account
    - **GCP**: GCP Free Tier
3. **A code editor** (VS Code, IntelliJ, etc.) with Terraform plugin support (optional but recommended).

4.2 - Basic Terraform Workflow

Regardless of the provider, the workflow usually follows these steps:

1. **Write configuration** (in .tf files).
2. **Initialize the working directory**: terraform init
3. **Review the plan**: terraform plan
4. **Apply the plan**: terraform apply
5. **Destroy resources**: terraform destroy (use with caution).

4.3 - AWS Example

4.3.1 - Setting Up AWS Access

1. **Create an AWS user** with programmatic access (using AWS IAM).
2. **Download or note the Access Key ID and Secret Access Key**.
3. **Configure AWS CLI (optional but convenient)**:

   ```
   aws configure
   ```

   Provide the AWS Access Key, Secret Access Key, default region (e.g., `us-east-1`), and default output format (e.g., `json`).

4.3.2 - Example Directory Structure

```
aws-terraform-tutorial/
├── main.tf
└── variables.tf (optional)
```

4.3.3 - main.tf for AWS

Below is a simple Terraform configuration that launches an EC2 instance in AWS. Make sure to replace placeholders with actual values.

```
# main.tf

terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
  # If not set up via CLI or environment variables, add:
  # access_key = "YOUR_ACCESS_KEY"
  # secret_key = "YOUR_SECRET_KEY"
}

# Use a data source to fetch the latest Amazon Linux 2 AMI
data "aws_ami" "amazon_linux" {
  most_recent = true
```

```
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

# Create a security group that allows SSH
resource "aws_security_group" "sg_main" {
  name        = "terraform-quickstart-sg"
  description = "Allow SSH inbound traffic"
  vpc_id      = data.aws_vpc.default.id

  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Use a data source to fetch the default VPC
data "aws_vpc" "default" {
  default = true
}

# Launch an EC2 instance using the dynamically fetched AMI
resource "aws_instance" "my_ec2_vm" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"

  vpc_security_group_ids = [aws_security_group.sg_main.id]
}
```

4.3.4 - Running the AWS Example

1. **Initialize** Terraform (download providers, set up the workspace):

   ```
   terraform init
   ```

2. **Review the plan** (what Terraform will do):

   ```
   terraform plan
   ```

3. **Apply the changes** (create the resources):

   ```
   terraform apply
   ```

Type yes when prompted.

You should see a new EC2 instance in the AWS console.

4. **Destroy** the resources when done (to avoid costs):

   ```
   terraform destroy
   ```

4.4 - Azure Example

4.4.1 - Setting Up Azure Access

1. **Create an Azure account** (or use an existing one).
2. **Install Azure CLI**: Azure CLI Installation
3. **Log in** to Azure via CLI:

   ```
   az login
   ```

4. (Optionally) **Create a Service Principal** if you need a programmatic approach:

   ```
   az ad sp create-for-rbac --role="Contributor" --
   scopes="/subscriptions/<SUBSCRIPTION_ID>"
   ```

   This command will give you the `appId`, `password`, and `tenant`.

4.4.2 - Example Directory Structure

```
azure-terraform-tutorial/
├── main.tf
└── variables.tf (optional)
```

4.4.3 - main.tf for Azure

```
# main.tf

terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"  # Check latest version
    }
  }
}

provider "azurerm" {
  features {}
  # If using Service Principal, specify:
  # subscription_id = "<your-subscription-id>"
  # client_id       = "<appId>"
  # client_secret   = "<password>"
  # tenant_id       = "<tenant>"
}
```

```
# Create a resource group
resource "azurerm_resource_group" "rg" {
  name     = "rg-terraform-quickstart"
  location = "East US"
}

# Create a virtual network
resource "azurerm_virtual_network" "vnet" {
  name                = "vnet-terraform"
  resource_group_name = azurerm_resource_group.rg.name
  location            = azurerm_resource_group.rg.location
  address_space       = ["10.0.0.0/16"]
}

# Create a subnet
resource "azurerm_subnet" "subnet" {
  name                 = "subnet1"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name = azurerm_virtual_network.vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

# Create a public IP
resource "azurerm_public_ip" "public_ip" {
  name                = "publicip-terraform"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}

# Create a network interface
resource "azurerm_network_interface" "nic" {
  name                = "nic-terraform"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "ipconfig1"
    subnet_id                     = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.public_ip.id
  }
}

# Create a simple VM
resource "azurerm_linux_virtual_machine" "vm" {
  name                = "vm-terraform"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  size                = "Standard_B1ls"  # Low-cost VM size

  admin_username       = "azureuser"
  admin_password       = "P@ssw0rd12345!"  # Not recommended for production
  network_interface_ids = [
    azurerm_network_interface.nic.id
  ]

  # Ubuntu 20.04 LTS image
  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-focal"
```

```
    sku       = "20_04-lts"
    version   = "latest"
  }
}
```

4.4.4 - Running the Azure Example

1. **Initialize**:

   ```
   terraform init
   ```

2. **Plan**:

   ```
   terraform plan
   ```

3. **Apply**:

   ```
   terraform apply
   ```

   Type yes when prompted.

Check the **Azure Portal** to see your new resources.

4. **Destroy**:

   ```
   terraform destroy
   ```

4.5 - GCP Example

4.5.1 - Setting Up GCP Access

1. **Create a GCP account** or use an existing one.
2. **Enable the Compute Engine API** in your Google Cloud Console.
3. **Install and configure the gcloud CLI**:

   ```
   gcloud init
   ```

   This will guide you through logging in and selecting a project.

4. **Create a Service Account** (optional for more controlled programmatic access):

   ```
   gcloud iam service-accounts create terraform-sa --display-
   name="Terraform Service Account"
   ```

   Then create and download a key file (JSON) for the service account:

   ```
   bash
   CopyEdit
   gcloud iam service-accounts keys create keyfile.json \
       --iam-account=terraform-
   sa@<your_project_id>.iam.gserviceaccount.com
   ```

You can use this JSON file in your Terraform provider configuration.

5.2. Example Directory Structure

```
gcp-terraform-tutorial/
├── main.tf
└── variables.tf (optional)
```

5.3. main.tf for GCP

```
# main.tf

terraform {
  required_version = ">= 1.0.0"
  required_providers {
    google = {
      source  = "hashicorp/google"
      version = "~> 4.0"
    }
  }
}

# Configure the GCP provider
provider "google" {
  project = "<YOUR_PROJECT_ID>"
  region  = "us-central1"
  # Provide the path to your service account key if not using gcloud CLI
default
  # credentials = file("path/to/keyfile.json")
}

# Create a simple VM instance
resource "google_compute_instance" "vm_instance" {
  name         = "terraform-quickstart-vm"
  machine_type = "f1-micro"
  zone         = "us-central1-a"

  # Use a public image for the VM (Debian 10 for example)
  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-10"
    }
  }

  network_interface {
    # Creates or uses the "default" network
    network = "default"
    # Assigns ephemeral public IP
    access_config {
    }
  }
}
```

5.4. Running the GCP Example

1. **Initialize**:

   ```
   terraform init
   ```

2. **Plan**:

   ```
   terraform plan
   ```

3. **Apply**:

   ```
   terraform apply
   ```

   Type yes when prompted.

Check the **GCP Console** (Compute Engine -> VM instances) to see your new VM.

4. **Destroy**:

   ```
   terraform destroy
   ```

4.6 - Key Takeaways & Tips

1. **Terraform State**: Terraform maintains a state file (`terraform.tfstate`) to keep track of your deployed resources. Protect this file or store it remotely (e.g., in an S3 bucket, Terraform Cloud, etc.).
2. **Version Control**: Always put your `.tf` files into a Git repository. Do **not** commit `terraform.tfstate` or sensitive credentials.
3. **Managing Credentials**:
   - Use environment variables or secrets managers rather than hardcoding credentials.
   - For AWS, you can set `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`.
   - For Azure, you might rely on the Azure CLI logged-in session.
   - For GCP, you can use the `GOOGLE_CREDENTIALS` environment variable to store JSON credentials.
4. Costs**: Although the examples use low-cost or free-tier resources, remember to** destroy **resources to avoid incurring charges once you're done experimenting.**

5 - References

- https://kubernetes.io/docs/concepts/overview/components/
- https://kubernetes.io/docs/tutorials/
- https://helm.sh/docs/
- https://www.terraform.io/use-cases/infrastructure-as-code