SCRIPTS

1. **Ansible Script:**

Playbook to Install Nginx on Ubuntu Servers:

```
---
- name: Install Nginx on Ubuntu Servers
  hosts: ubuntu_servers
  become: yes
  tasks:
    - name: Update apt package cache
      apt:
        update_cache: yes

    - name: Install Nginx
      apt:
        name: nginx
        state: present

    - name: Start Nginx service
      service:
        name: nginx
        state: started
```

2. **Terrafrorm Script:**

Terraform Configuration to Create VPC, Subnet, and EC2 Instance:

```
# Provider Configuration
provider "aws" {
  region = "us-east-1"
}

# VPC Configuration
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
}

# Subnet Configuration
```

1. Save the above Terraform configuration in a file with a `.tf` extension, for example, `ec2_vpc_subnet.tf`.
2. Initialize Terraform in the directory containing the configuration file: `terraform init`.
3. Apply the configuration to create the VPC, subnet, and EC2 instance: `terraform apply`.
4. Follow the prompts to confirm the changes.

This script will create a VPC with the CIDR block `10.0.0.0/16`, a subnet within the VPC with the CIDR block `10.0.1.0/24`, and an EC2 instance launched in the specified subnet. Adjust the parameters such as region, instance type, AMI, CIDR blocks, etc., according to your requirements.

```
resource "aws_subnet" "my_subnet" {
  vpc_id            = aws_vpc.my_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "us-east-1a" # Change the AZ as per your preference
}

# EC2 Instance Configuration
resource "aws_instance" "my_instance" {
  ami           = "ami-0c55b159cbfafe1f0" # Change to your desired AMI
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.my_subnet.id
  tags = {
    Name = "my-instance"
  }
}
```

3. Kubernetes:

Kubernetes YAML file to create a Nginx deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```

```
kubectl apply -f nginx-deployment.yaml
kubectl apply -f my-service.yaml
```

```
image: nginx:latest
ports:
- containerPort: 80
```

4. **Docker:**

Java application:

```
# Use an official OpenJDK runtime as the base image
FROM openjdk:11

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the JAR file into the container
COPY target/my-java-app.jar .

# Expose port 8080 to the outside world
EXPOSE 8080

# Command to run the application
CMD ["java", "-jar", "my-java-app.jar"]
```

```
docker build -t my-java-app .
docker run -p 8080:8080 my-java-app
```