1) Upload a text file to a AWS S3 bucket

```
# service provider AWS Cloud

Proivder "aws"
  access_key = "access_key"
  secret_key = "secret_key"
  region     = "ap-south-1"
# create a bucket
resource "aws_s3_bucket" "mybucket"{
  bucket = "my_test-bucket"
  acl    =  private"
  tag = {
    Name="bucket created by terraform_anish"
    Environment =Dev"
   }

}

# Upload an object
resource "aws_s3_bucket_object" "file upload" {
  bucket = "my_bukket"
  key    = my bucket_key"
  acl    = private"
  source ="home/anish/terraform/interview.txt
```

2) On upload, trigger some type of alert (SNS, Email, Slack, text, etc)

```
# Creating Lambda IAM resource
resource "aws_iam_role" "lambda_iam" {
  name = var.lambda_role_name

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
}
```

```
resource "aws_iam_role_policy" "revoke_keys_role_policy" {
  name = var.lambda_iam_policy_name
  role = aws_iam_role.lambda_iam.id

  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:*",
        "ses:*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
EOF
}

# Creating Lambda resource
resource "aws_lambda_function" "test_lambda" {
  function_name    = var.function_name
  role             = aws_iam_role.lambda_iam.arn
  handler          = "src/${var.handler_name}.lambda_handler"
  runtime          = var.runtime
  timeout          = var.timeout
  filename         = "../src.zip"
  source_code_hash = filebase64sha256("../src.zip")
  environment {
    variables = {
      env            = var.environment
      SENDER_EMAIL   = var.sender_email
      RECEIVER_EMAIL = var.receiver_email
    }
  }
}

# Creating s3 resource for invoking to lambda function
resource "aws_s3_bucket" "bucket" {
  bucket = my_test-bucket
  acl    = "private"

  tags = {
    Environment = var.environment
  }
}

# Adding S3 bucket as trigger to my lambda and giving the permissions
```

```
resource "aws_s3_bucket_notification" "aws-lambda-trigger" {
  bucket = my_test-bucket
  lambda_function {
    lambda_function_arn = aws_lambda_function.test_lambda.arn
    events              = ["s3:ObjectCreated:*", "s3:ObjectRemoved:*"]

  }
}
resource "aws_lambda_permission" "test" {
  statement_id  = "AllowS3Invoke"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.test_lambda.function_name
  principal     = "s3.amazonaws.com"
  source_arn    = "arn:aws:s3:::${aws_s3_bucket.bucket.id}"
}
```

4) Create the following through Terraform:
   1 VPC, IGW, Route table, subnet & security group

```
 # Create VPC using terraform

resource "aws_vpc" "ownvpc" {
  cidr_block       = "0.0.0.0/16"
  instance_tenancy = "default"
}
# Two subnet created one public and other is private.

// public subnet

resource "aws_subnet" "public" {
  vpc_id     = aws_vpc.ownvpc.id
  cidr_block = "192.168.0.0/24"
  availability_zone = "ap-south-1a"
}

// private subnet

resource "aws_subnet" "private" {
    vpc_id = aws_vpc.ownvpc.id
    cidr_block = "0.0.1.0/24"
    availability_zone = "ap-south-1b"
}
# Create a public facing internet gateway for connecting VPC/Network with the
internet world and also attach gateway to VPC

resource "aws_internet_gateway" "mygateway" {
```

```
  vpc_id = aws_vpc.ownvpc.id
}
# Create routing table for internet gateway

resource "aws_route_table" "my_table" {
  vpc_id = aws_vpc.ownvpc.id


  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.mygateway.id
  }

}
resource "aws_route_table_association" "rta_subnet_public" {
  subnet_id      = aws_subnet.public.id
  route_table_id = aws_route_table.my_table.id
}

#  Creating security groups

resource "aws_security_group" "mywebsecurity" {
  name        = "my_web_security"
  description = "Allow http,ssh,icmp"
   vpc_id      =  aws_vpc.ownvpc.id

  ingress {
    description = "HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
 ingress {
    description = "MYSQL"
    from_port   = 3306
    to_port     = 3306
    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"]
  }

  tags = {
    Name = "mywebserver_sg"
  }
}

 # Create an EC2 instance running a webserver (e.g apache or tomcat) associated with
the above components.

resource "aws_instance" "webserver"{
  ami = "ami-07a8c73a650069cf3"
   instance_type="t2.micro"
    availability_zone = "ap-south-1b"
     key_name=var.mykey
security_groups = ["my_web_security"]
user_data=<<-EOF
    #! /bin/bash
    sudo yum install git -y
    sudo yum install httpd -y
    sudo systemctl start httpd
    sudo systemctl enable httpd
    sudo mkfs -t ext4 /dev/sdd
    sudo mount /dev/sdd /var/www/html
    EOF
tags={
Name="webserver"
}
}



open the terminal
comand
terraform init
terraform plan
terraform apply


6) Write helm chart/k8s manifests to deploy a microservice
If possible demonstrate using minikube/kind cluster on your local laptop



kuectl install demo steps

Install Docker
$  sudo apt update && apt -y install docker.io
```

```
 Install kubectl
$  curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd6
4/kubectl &&   chmod +x ./kubectl && sudo mv ./kubectl /usr/local/bin/kubectl

 Install Minikube
$  curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 &&
chmod +x minikube && sudo mv minikube /usr/local/bin/

 Start Minikube
$  apt install conntrack
$  minikube start --vm-driver=none
$  minikube status

To download helm
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh

-------------------------------task work demo------------------
/eksinterview
├── charts/
├── Chart.yaml
├── templates/
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

----------------------------------------
We'll follow this template, and create a new chart called eksinterviewcd
~/environment
helm create eksinterviewcd
cd eksinterviewcd
//Run the following code block to create a new Chart.yaml file

cat <<EoF > ~/environment/eksinterview/Chart.yaml
apiVersion: v2
name: eksinterviewcd
description: A Helm chart for EKS Workshop Microservices application
version: 0.1.0
```

```
appVersion: 1.0
EoF
//Next we'll copy the manifest files for each of our microservices into the
templates directory as servicename.yaml

mkdir -p ~/environment/eksinterview/templates/deployment
mkdir -p ~/environment/eksinterview/templates/service

# Copy and rename frontend manifests
cp ~/environment/eksinterview-frontend/kubernetes/deployment.yaml
~/environment/eksinterview/templates/deployment/frontend.yaml
cp ~/environment/eksinterview-frontend/kubernetes/service.yaml
~/environment/eksinterview/templates/service/frontend.yaml

# Copy and rename crystal manifests
cp ~/environment/eksinterview-crystal/kubernetes/deployment.yaml
~/environment/eksinterview/templates/deployment/crystal.yaml
cp ~/environment/eksinterview-crystal/kubernetes/service.yaml
~/environment/eksinterview/templates/service/crystal.yaml

# Copy and rename nodejs manifests
cp ~/environment/eksinterview-nodejs/kubernetes/deployment.yaml
~/environment/eksinterview/templates/deployment/nodejs.yaml
cp ~/environment/eksinterview-nodejs/kubernetes/service.yaml
~/environment/eksinterview/templates/service/nodejs.yaml

#Under spec, find replicas: 1 and replace with the following:
replicas: {{ .Values.replicas }}

# Under spec.template.spec.containers.image, replace the image with the correct
template value.


#Create a values.yaml file with our template defaults
cat <<EoF > ~/environment/eksdemo/values.yaml
# Default values for eksdemo.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.

# Release-wide Values
replicas: 3
version: 'latest'

# Service Specific Values
nodejs:
  image: brentley/eksinterview-nodejs
crystal:
  image: brentley/eksinterview-crystal
frontend:
  image: brentley/eksinterview-frontend
```

```
  EoF

#Use the dry-run flag to test our templates

helm install --debug --dry-run workshop ~/environment/eksinterview

#Deploy the chart
helm install workshop ~/environment/eksinterview
kubectl get svc,po,deploy

#To test the service our eksinterview Chart created

kubectl get svc ecsdemo-frontend -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}"; echo

# Open values.yaml and modify the image name under nodejs.image

# Deploy the updated demo application chart:
helm upgrade workshop ~/environment/eksinterview
```