**Relazione Tecnica – Burli Giorgia**

**Obiettivo del Progetto**

L'obiettivo dell'esercitazione è stato quello di effettuare il deploy di una semplice applicazione Node.js utilizzando Docker e K3s su una VM creata su Microsoft Azure, gestendo l'infrastruttura con Terraform. L'applicazione doveva rispondere con "Hello, World!" ed essere accessibile via browser all'indirizzo pubblico della macchina.

Inoltre, su indicazione successiva del docente, si è previsto che la VM fosse una sola anziché tre, ma che il pod fosse replicato tre volte (replica di tipo Deployment).

**1. Preparazione Ambiente Terraform**

Si è creata la cartella Verifica\_02 con i file seguenti:

* main.tf: provider, resource group, rete, subnet, NSG, IP pubblico, NIC, macchina virtuale
* variables.tf: contiene variabili riutilizzabili
* outputs.tf: stampa IP pubblico della VM

**Contenuto variables.tf:**

variable "resource\_group\_name" { default = "k3s-rg" }

variable "location" { default = "East US" }

variable "admin\_username" { default = "azureuser" }

variable "ssh\_public\_key\_path" { default = "~/.ssh/id\_ed25519.pub" }

variable "ssh\_private\_key\_path" { default = "~/.ssh/id\_ed25519" }

Nota: si è scelto di usare variables.tf e non terraform.tfvars perché i valori non variavano tra ambienti.

**Comandi eseguiti:**

terraform init

terraform plan

terraform apply

**Errore riscontrato:**

Error: unable to build authorizer for Resource Manager API: could not configure AzureCli Authorizer

**Soluzione:**

az login

Dopo l'autenticazione, terraform apply ha restituito l’IP pubblico:

**Outputs:**

public\_ip\_address = "xxx.xxx.xx.xxx"

**2. Generazione e gestione chiavi SSH**

Poiché la cartella .ssh non esisteva, si è eseguito:

ssh-keygen

Confermato percorso suggerito C:\Users\user\.ssh\id\_ed25519. Verifica:

dir C:\Users\user\.ssh

**3. Connessione alla VM**

Da PowerShell:

ssh -i C:\Users\user\.ssh\id\_ed25519 azureuser@publicIP

Alla richiesta Are you sure you want to continue connecting? si è risposto yes.

**4. Creazione dei file applicativi**

In Visual Studio Code, nella cartella Verifica\_02 si è creata la sottocartella app/ con:

**app.js**

const express = require('express');

const app = express();

const port = 3000;

app.get('/', (req, res) => {

res.send('Hello, World!');

});

app.listen(port, () => {

console.log(`App listening at http://localhost:${port}`);

});

**package.json**

{

"name": "hello-docker",

"version": "1.0.0",

"main": "app.js",

"scripts": {

"start": "node app.js"

},

"dependencies": {

"express": "^4.17.1"

}

}

**Dockerfile**

FROM node:14

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]

**5. Trasferimento file nella VM**

Compressione e invio dell'app:

Compress-Archive -Path .\app\\* -DestinationPath .\app.zip

scp -i ~/.ssh/id\_ed25519 app.zip azureuser@172.publicIP:~

Invio dello script di installazione:

scp -i ~/.ssh/id\_ed25519 install\_k3s.sh azureuser@172.pubblicIP:~

**6. Installazione Docker, K3s e Deploy automatico**

Nella VM:

chmod +x install\_k3s.sh

sudo ./install\_k3s.sh

**Errore riscontrato:**

cd: /home/azureuser/app: No such file or directory

**Soluzione:**

unzip app.zip

mv app ~/app

cd ~/app

**Costruzione immagine Docker:**

sudo docker build -t hello-docker .

sudo docker images

**7. Creazione e deploy del deployment.yaml**

**Nella VM:**

nano ~/deployment.yaml

**Contenuto:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-docker-deployment

spec:

replicas: 3

selector:

matchLabels:

app: hello-docker

template:

metadata:

labels:

app: hello-docker

spec:

containers:

- name: hello-docker

image: hello-docker

imagePullPolicy: IfNotPresent

ports:

- containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

name: hello-docker-service

spec:

type: NodePort

selector:

app: hello-docker

ports:

- protocol: TCP

port: 3000

targetPort: 3000

nodePort: 30080

**Comando:**

sudo kubectl apply -f ~/deployment.yaml

sudo kubectl get pods

**8. Configurazione NSG**

Nel file main.tf, il blocco azurerm\_network\_security\_group conteneva regole per porte 22, 3000 e 30080.

**9. Verifica finale**

**Accesso da browser:**

http://172.190.33.118:30080

**Risultato:**

Hello, World!

**10. Per automatizzare il provisioning della VM, abbiamo inserito nel file main.tf un blocco null\_resource che utilizza i provisioner di Terraform. Questo ci permette di:**

1. Copiare automaticamente lo script install\_k3s.sh all'interno della VM tramite il file provisioner.
2. Eseguire lo script subito dopo, usando il remote-exec provisioner.

**Passaggi effettuati:**

1.Scrittura dello script di installazione

* Creato il file install\_k3s.sh nella cartella Verifica\_02, contenente:
  + aggiornamento pacchetti
  + installazione Docker
  + installazione K3s con flag --docker
  + costruzione immagine Docker
  + creazione file deployment.yaml
  + comando kubectl apply per il deploy

Dopo aver aggiornato main.tf e verificato le chiavi SSH, è bastato eseguire:

terraform apply

Al termine della creazione della VM, Terraform ha:

* caricato lo script nella home dell’utente
* reso lo script eseguibile
* avviato l’installazione completa di Docker, K3s e dell’app

Questo approccio ha permesso di eliminare la necessità di usare scp, ssh o comandi manuali per ogni nuova VM creata, aumentando l’automazione e la replicabilità del progetto.

**Conclusione**

L'applicazione è stata deployata con successo. I pod sono stati replicati 3 volte come da richiesta, e l'app è accessibile via browser. Tutti i problemi incontrati sono stati risolti durante il lavoro. Ogni fase è documentata con i comandi PowerShell e Linux effettivamente utilizzati. Il provisioning automatico tramite provisioner completa un flusso di lavoro cloud-native ripetibile e scalabile.