A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

Tesina di progetto

Virtual Network and Cloud Computing

Corso di Laurea Magistrale in Ingegneria Informatica e Robotica

Curriculum Data Science

A.A. 2023-2024

Dipartimento di Ingegneria

docente

Prof. Gianluca Reali

Utilizzo dello strumento software Terraform per lo sviluppo di un applicativo su cluster Kubernetes

studente

363551 **Andrea Tomassoni** andrea.tomassoni@studenti.unipg.it

Data ultimo aggiornamento: 5 luglio 2024

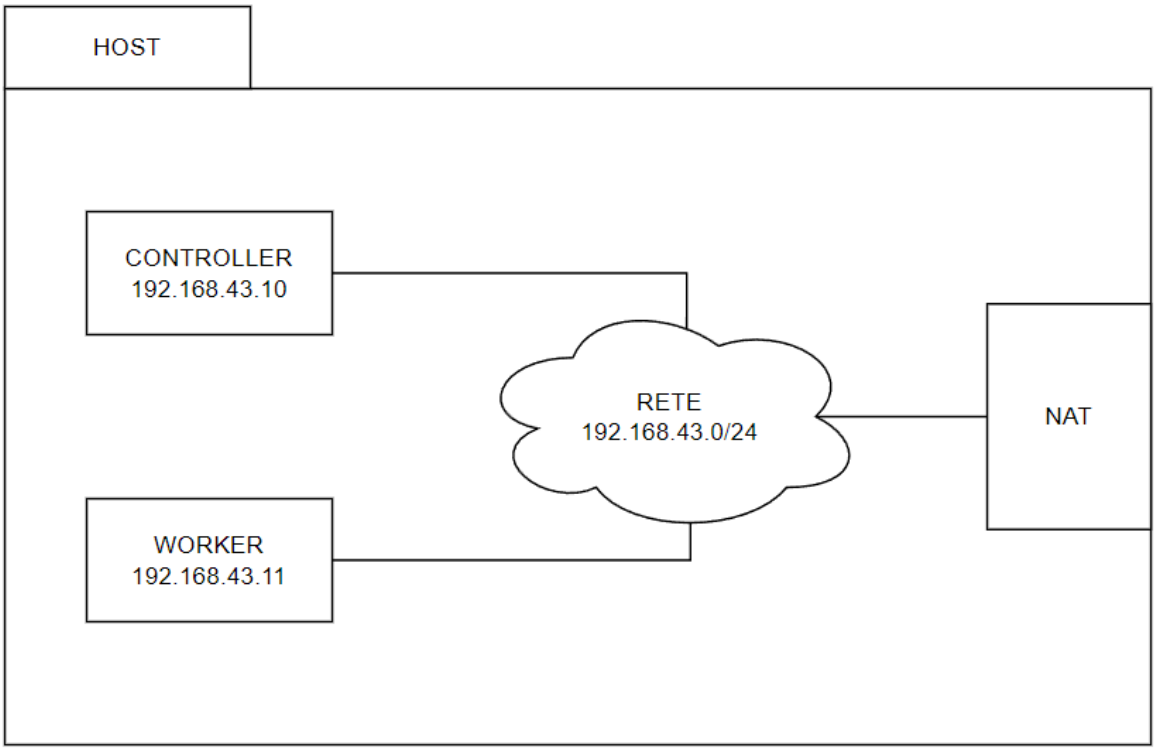
0. Indice

1. Introduzione

Il progetto ha come scopo l'introduzione allo strumento software di Iac (Infrastructure as a Code) Terraform. Nello specifico, si è proceduto a sviluppare e mettere in opera un semplice applicativo web, appoggiato su un cluster Kubernetes precedentemente configurato.

1.1 Cluster

Il cluster kubernetes è costituito da due macchine virtuali con 4GB di RAM e 2 vCore ciascuna, su cui è installato Xubuntu 22. Le due macchine virtuali sono connesse ad una rete con nat gestita da VirtualBox. Per quanto riguarda i nodi kubernetes, una VM ospita il nodo controller e l'altra il nodo worker.



2. Tecnologie utilizzate

2.1 Kubespray

Kubespray è un progetto open source che fornisce una serie di strumenti e per automatizzare la distribuzione e la gestione di cluster Kubernetes; è basato su Ansible.

2.2 Terraform

Per tutto ciò che riguarda la gestione del cluster kubernetes è stato utilizzato Terraform, un software di Infrastructure as a Code che tramite un approccio dichiarativo permette la gestione del cluster e di tutto ciò che viene creato al suo interno.
Terraform è dotato di API che gli permettono di lavorare con tutti i principali provider di servizi in cloud, come AWS, Google e Azure; in questo modo si può lavorare con un infrastruttura eterogenea utilizzando un singolo tool.
Nello specifico, Terraform permette di instanziare le singole componenti dei servizi in cloud, come ad esempio container e volumi, ma anche interi cluster.

3. Setup del cluster

Per quanto riguarda la parte di setup del progetto si è proceduto configurando come prima cosa la rete in modo tale che le due macchine potessero comunicare tra di loro tramite SSH senza bisogno di autenticazione.
In una seconda fase si è proceduto alla creazione del cluster Kubernetes tramite Kubespray e a testare che tutto funzionasse come previsto.
Infine si è proceduto all'installazione di Terraform e alla sua configurazione.

Nota
I passaggi appena descritti sono descritti nel dettaglio e comprensivi di codice nella repository Github del progetto

4. Caso di studio

In questo progetto si è proceduto a realizzare un applicativo web per la condivisione di file statici; per far ciò è stata utilizzata un immagine di un server NGINX modificata in modo tale da avere già in memoria i file e renderli disponibili tramite url statico.
Andando più nel dettaglio, sono state create le seguenti componenti:

- Deployment Kubernetes che instanzia il pod in cui è situato il container contentente l'immagine del server (immagine disponbile su repository docker come andreatms/file-server)
- Service Kubernetes, per rendere il container raggiungibile all'indirizzo *http://file-server*
- Horizontal Pod Autoscaler, per garantire la creazione di un nuovo pod quando il consumo della CPU supera il 50%

Tutte le componenti sopra elencate sono state definite con approccio dichiarativo tramite un unico script terraform. In questo script sono presenti anche le configurazione che riguardano il provider (Kubernetes in questo caso) e le variabili del file *terraform.tfvars*.

Deployment

```
resource "kubernetes_deployment" "file-server-deployment" {
  metadata {
    name = "file-server-deployment"
    labels = {
      App = "fileServer"
    }
  }

  spec {
    selector {
      match_labels = {
        App = "fileServer"
      }
    }

    template {
        metadata {
            labels = {
                App = "fileServer"
            }
        }
        spec {
            container {
                image = "andreatms/file-server:latest"
                name = "nginx"

                port {
                    container_port = 80
                }

                resources {
                  limits = {
                    cpu = "200m"
                    memory = "512Mi"
                  }
                  requests = {
                    cpu = "100m"
                    memory = "256Mi"
                  }
                }
            }
        }
    }
  }
}
```

Service

```
resource "kubernetes_service" "file-server" {
  metadata {
    name = "file-server"
  }
  spec {
    selector = {
      App = kubernetes_deployment.file-server-deployment.spec.0.template.0.meta ...
    }
    port {
      port = 80
    }
  }
}
```

Horizontal Pod Autoscaler

```
resource "kubernetes_horizontal_pod_autoscaler" "file-server-hpa" {
  metadata {
    name = "file-server-hpa"
  }
  spec {
    min_replicas = 1
    max_replicas = 5

    target_cpu_utilization_percentage = 50

    scale_target_ref {
      kind = "Deployment"
      name = kubernetes_deployment.file-server-deployment.metadata[0].name
      api_version = "apps/v1"
    }
  }
}
```

Per testare le funzionalità del sistema messo in opera è stato definito un file manifest, in questo caso definito con la sintassi *yaml* (nativa di Kubernetes), per instanziare alcuni pod che si occupano di generare del carico per il server web.
Per fare ciò eseguono molteplici richieste del tipo

wget -qO- http://file-server/books/dante_alighieri/divina_commedia.txt"

In questo modo sono state testate le funzionalità del deployment e del servizio, ma anche e soprattutto quelle del HPA configurato per gestire da 1 a 5 repliche.

File Downloader

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: file-downloader-dc
spec:
  containers:
  - name: downloader
    image: busybox:latest
    command: ["sh", "-c"]
    args:
      - while true; do
          if wget -qO- http://file-server/books/dante_alighieri/di ...
            echo "OK - File downloaded successfully";
          else
            echo "ERROR - File download failed";
          fi;
          sleep 0.002;
        done;
  restartPolicy: Always
```