Tesina di

Virtual Networks and Cloud Computing

Corso di Laurea in Ingegneria Informatica e Robotica – A.A. 2023-2024

Dipartimento di Ingegneria

docente

Prof. Gianluca Reali

Tesina di Virtual Networks and Cloud Computing

studenti

Alex Ardelean alexnicolae.ardelean@studenti.unipg.it

Data ultimo aggiornamento: 10 gennaio 2025

Indice

Capitolo 1

Installazione Kubernetes

1.1 Configurazione dell'Ambiente

Il primo passo per configurare il cluster Kubernetes è inizializzare due macchine virtuali che rappresentano i nodi del cluster: una per il nodo master e l'altra per il nodo worker. Per implementare i due nodi si è utilizzato il SO Xubuntu-22-02.

Per assicurare una corretta comunicazione di rete tra le due macchine virtuali, è necessario creare una rete NAT specifica per Kubernetes all'interno di VirtualBox. Questa rete permette alle VM di comunicare tra loro in modo isolato rispetto alla rete principale del computer host.

1.2 Configurazione della rete

Ogni VM ha bisogno di un indirizzo IP statico per garantire che i nodi del cluster possano sempre trovarsi e comunicare correttamente.

Per configurare gli indirizzi IP si è modificato il file `/etc/netplan/01-network-manager-all.yaml` utilizzando la seguente configurazione:

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
      enp0s3:
         dhcp4: no
         addresses: [<ip-nodo>]
```

```
        gateway4: 192.168.43.1
        nameservers:
            addresses: [8.8.8.8, 8.8.4.4]
```

File 1.1: /etc/netplan/01-network-manager-all.yaml

con ip `192.168.43.10/24` per il nodo master e `192.168.43.11/24` per il nodo worker.

Per rendere effettiva la configurazione si esegue:

```
    sudo netplan apply
```

É anche necessario modificare il file `/etc/hosts` su entrambe le macchine virtuali per consentire la risoluzione dei nomi host:

```
192.168.43.10 controller controller.example.com
192.168.43.11 worker-1 worker-1.example.com

192.168.43.10 controller.provaspray.local controller node1
192.168.43.11 worker-1.provaspray.local worker-1 node2
```

File 1.2: /etc/hosts

Inoltre si modifica il file `/etc/hostname` su ogni nodo per impostare il nome host corretto. Sul nodo master:

```
controller
```

File 1.3: /etc/hostname

e sul nodo worker:

```
worker-1
```

File 1.4: /etc/hostname

Questi passaggi completano la configurazione iniziale dell'ambiente, assicurando che le macchine virtuali siano correttamente configurate per la comunicazione e la gestione del cluster Kubernetes.

1.3 Installazione del Cluster Kubernetes

1.3.1 Installazione dei Prerequisiti

Prima di poter installare Kubernetes, è necessario preparare i nodi con alcuni software di base.

Per gestire i nodi del cluster da remoto, è necessario installare un server SSH su entrambe le VM tramite il comando

```
sudo apt install openssh-server
```

Per consentire ad Ansible di gestire i nodi senza richiedere una password ogni volta, è necessario configurare l'accesso SSH senza chiave. Per generare una chiave SSH:

```
ssh-keygen -t rsa
```

lasciando il campo password vuoto.

Si copia la chiave pubblica sui nodi master e worker usando:

```
ssh-copy-id 192.168.43.10
ssh-copy-id 192.168.43.11
```

Per consentire ad Ansible di eseguire comandi con privilegi elevati senza richiedere una password, si modifica il file `/etc/sudoers` su entrambe le macchine come segue:

```
root ALL=(ALL) NOPASSWD: ALL
studente ALL=(ALL) NOPASSWD: ALL
```

File 1.5: /etc/sudoers

Ansible richiede Python per funzionare, è quindi necessario installarlo su entrambi i nodi eseguendo:

```
sudo apt install python3-pip
sudo pip3 install --upgrade pip
```

1.3.2 Installazione di Kubespray

Una volta che tutti i prerequisiti sono stati installati, si può procedere con l'installazione di Kubespray, un potente strumento che utilizza Ansible per automatizzare il deployment di cluster Kubernetes.

Sul nodo master, si installa Git se non è già presente e poi si clona il repository di Kubespray:

```
sudo apt install git
git clone https://github.com/kubernetes-sigs/kubespray.
    ↪ git
cd kubespray
```

Si installano le dipendenze di Kubespray utilizzando:

```
sudo pip install -r requirements.txt
```

Kubespray utilizza un inventario Ansible per specificare i nodi del cluster. Per creare l'inventario Ansible `hosts.yaml` si usano i comandi:

```
cp -rfp inventory/sample inventory/mycluster
declare -a IPS=(192.168.43.10 192.168.43.11)
CONFIG_FILE=inventory/mycluster/hosts.yaml python3
    ↪ contrib/inventory_builder/inventory.py ${IPS[@]}
```

É possibile verificare la configurazione nel file `inventory/mycluster/hosts.yaml`.

1.3.3 Configurazione del cluster

I file di configurazione principali per il cluster Kubernetes sono situati all'interno della directory `inventory/mycluster/group_vars`.

É necessario configurare il server DNS all'interno del file `/all/all.yml` che va modificato come segue:

```
upstream_dns_servers:
  - 8.8.8.8
  - 8.8.4.4
```

File 1.6: inventory/mycluster/group_vars/all/all.yml

Kubernetes supporta diversi plugin di rete per gestire la comunicazione tra i pod nel cluster. Nel file `k8s-cluster/k8s-cluster.yml` è possibile specificare quale plugin di rete utilizzare, si modifica quindi come segue:

```
kube_config_dir: /etc/kubernetes
kube_network_plugin: calico
```

File 1.7: inventory/mycluster/group_vars/k8s-cluster/k8s-cluster.yml

Per configurare gli intervalli degli indirizzi IP utilizzati dal cluster per gestire la comunicazione tra i servizi e i pod si modifica sempre il file `k8s-cluster/k8s-cluster.yml` come segue

```
kube_service_addresses: 10.233.0.0/18
kube_pods_subnet: 172.16.0.0/24
kube_network_node_prefix: 24
```

File 1.8: inventory/mycluster/group_vars/k8s-cluster/k8s-cluster.yml

Per pulire un'installazione già esistente è possibile eseguire

```
ansible-playbook -i inventory/mycluster/hosts.yaml --
    ↪ become --become-user=root reset.yml
```

Per configurare e avviare tutti i componenti necessari del cluster Kubernetes, inclusi i nodi master e worker si esegue:

```
ansible-playbook -i inventory/mycluster/hosts.yaml --
    ↪ become --become-user=root cluster.yml
```

1.3.4 Installazione di kubectl

Una volta che il cluster è installato, è necessario installare kubectl, lo strumento da linea di comando per Kubernetes, per gestire il cluster dal nodo master:

```
sudo snap install kubectl --classic
```

Dopo aver installato kubectl, è possibile verificare che il cluster sia funzionante controllando i nodi con il comando:

```
kubectl get nodes
```

Capitolo 2

Deployment Guestbook

2.1 Deployment Redis

2.1.1 Redis leader

Si crea un Deployment Redis che crea un Pod con un unico container che esegue Redis versione 6.0.5 esponendo la porta 6379 (porta default per Redis).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
        role: leader
        tier: backend
    spec:
      containers:
```

```
    - name: leader
      image: "docker.io/redis:6.0.5"
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
      - containerPort: 6379
```

File 2.1: application/guestbook/redis-leader-deployment.yaml

Redis è un database in-memory (memorizza i dati direttamente nella memoria RAM) open-source con supporto di meccanismi di persistenza. Redis supporta la replica dei dati, che consente di avere più istanze di Redis che replicano i dati da un master a uno o più slave. Il Pod è etichettato come `role: leader`, suggerendo che fa parte di un'architettura Redis in cui ci sono probabilmente altri Pods che fungono da replica o da nodi di supporto. Le repliche possono gestire le richieste di lettura, riducendo il carico sul nodo master.

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/redis-leader-deployment.yaml
```

2.1.2 Servizio per Redis leader

Al fine di esporre il Pod Redis leader all'interno del cluster Kubernetes, permettendo ad altri Pods di accedere al servizio Redis tramite il nome del Service e la porta configurata, si definisce il seguente servizio

```
tutorials/guestbook
apiVersion: v1
kind: Service
metadata:
  name: redis-leader
  labels:
    app: redis
    role: leader
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
```

```
    app: redis
    role: leader
    tier: backend
```

File 2.2: application/guestbook/redis-leader-service.yaml

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/redis-leader-service.yaml
```

2.1.3 Redis follower

Al fine di poter gestire un carico maggiore di richieste di lettura è possibile definire dei follower che si sincronizzano con i dati del master (leader) e gestiscono le operazioni di lettura.

Il seguente Deployment crea e gestisce due repliche del Pod Redis follower, assicurando che ci siano due istanze in esecuzione.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-follower
  labels:
    app: redis
    role: follower
    tier: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
        role: follower
        tier: backend
    spec:
      containers:
      - name: follower
        image: us-docker.pkg.dev/google-samples/containers/
            ↪ gke/gb-redis-follower:v2
```

```
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
      - containerPort: 6379
```

File 2.3: application/guestbook/redis-follower-deployment.yaml

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/redis-follower-deployment.yaml
```

2.1.4 Servizio per Redis follower

Per esporre e gestire l'accesso ai Pod Redis follower nel cluster Kubernetes si definisce il seguente servizio

```
apiVersion: v1
kind: Service
metadata:
  name: redis-follower
  labels:
    app: redis
    role: follower
    tier: backend
spec:
  ports:
    # the port that this service should serve on
  - port: 6379
  selector:
    app: redis
    role: follower
    tier: backend
```

File 2.4: application/guestbook/redis-follower-service.yaml

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/redis-follower-service.yaml
```

2.2 Deployment del frontend

2.2.1 Frontend

Il seguente Deployment crea e gestisce tre repliche di un'applicazione PHP configurata per interagire con Redis. Questo Deployment assicura che l'applicazione frontend sia in grado di gestire il traffico HTTP sulla porta 80, mentre si affida a DNS per risolvere gli indirizzi degli altri servizi nel cluster.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
        app: guestbook
        tier: frontend
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: us-docker.pkg.dev/google-samples/containers/
            ↪ gke/gb-frontend:v5
        env:
        - name: GET_HOSTS_FROM
          value: "dns"
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 80
```

File 2.5: application/guestbook/frontend-deployment.yaml

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/frontend-deployment.yaml
```

2.2.2 Servizio per il frontend

Il seguente servizio, di tipo LoadBalancer, espone l'applicazione frontend al di fuori del cluster Kubernetes, fornendo un punto di accesso pubblico tra le repliche dei Pod frontend. Questo consente agli utenti esterni di accedere all'applicazione attraverso la porta 80.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  ports:
    # the port that this service should serve on
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

File 2.6: application/guestbook/frontend-deployment.yaml

Per effettuare il deployment si esegue:

```
kubectl apply -f https://k8s.io/examples/application/
    ↪ guestbook/frontend-service.yaml
```

2.2.3 Port forwarding

Per inoltrare il traffico dalla porta 8080 della macchina locale alla porta 80 del Service frontend nel cluster Kubernetes si esegue:

```
kubectl port-forward svc/frontend 8080:80
```

in questo modo l'applicazione è disponibile alla pagina `http://localhost:8080`.

2.3 Test dell'applicazione

Per verificare che i pod siano in funzione:

```
kubectl get pods
```

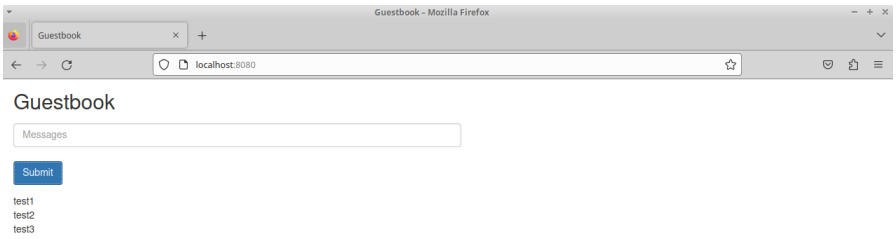
```
NAME                              READY   STATUS    RESTARTS   AGE
frontend-7dc7786f46-lpnlm         1/1     Running   0          25s
frontend-7dc7786f46-mbwvt         1/1     Running   0          94s
frontend-7dc7786f46-v79qj         1/1     Running   0          104s
redis-follower-6bbb6f56b4-cvhj8   1/1     Running   0          41m
redis-follower-6bbb6f56b4-jxmlj   1/1     Running   0          41m
redis-leader-595c97b985-qq8ch     1/1     Running   0          43m
```

Per verificare che i servizi siano in funzione:

```
kubectl get service
```

```
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
frontend         ClusterIP   10.233.20.163   <none>        80/TCP     5m4s
kubernetes       ClusterIP   10.233.0.1      <none>        443/TCP    20h
redis-follower   ClusterIP   10.233.18.239   <none>        6379/TCP   31m
redis-leader     ClusterIP   10.233.56.228   <none>        6379/TCP   33m
```

L'applicazione è accessibile alla pagina `http://localhost:8080` :



Capitolo 3

HPA

L'Horizontal Pod Autoscaler (HPA) in Kubernetes è una risorsa che regola automaticamente il numero di repliche dei Pod basandosi sull'utilizzo delle risorse o altre metriche definite dall'utente.

Per implementare un HPA sono necessari:

- Cluster Kubernetes funzionante.
- Metric Server.
- Workload da scalare: in questo caso è il frontend dell'applicazione guestbook.

3.1 Installazione del metric server

Il Metric Server è un componente fondamentale in Kubernetes per abilitare funzionalità di autoscaling come l'Horizontal Pod Autoscaler (HPA).

Il Metric Server è un aggregatore di metriche di risorse di Kubernetes che raccoglie dati di utilizzo della CPU e della memoria da tutti i nodi del cluster e dai Pod in esecuzione su di essi. Questi dati sono fondamentali per il funzionamento di diversi componenti di Kubernetes che necessitano di monitorare le prestazioni e gestire il carico di lavoro.

Il Metric Server raccoglie periodicamente le metriche di utilizzo delle risorse direttamente dal kubelet di ciascun nodo del cluster. Dopo aver raccolto i dati, il Metric Server li aggrega e li rende disponibili tramite l'API delle metriche di Kubernetes, che può essere interrogata da altri componenti di Kubernetes come l'HPA. Le metriche fornite dal Metric Server sono utilizzate per le decisioni di autoscaling.

Ad esempio, l'HPA utilizza queste metriche per determinare quando aumentare o ridurre il numero di repliche di un'applicazione in base al carico attuale.

Il Metric Server può essere installato utilizzando un file manifest YAML che contiene tutte le configurazioni necessarie per il suo deployment. Il comando per scaricare il manifest è:

```
wget -O components.yaml https://github.com/kubernetes-
    ↪ sigs/metrics-server/releases/latest/download/
    ↪ components.yaml
```

Dato che si tratta di un ambiente di test si disabilita la verifica del certificato TLS del kubelet andando ad aggiungere tra gli `args` del container la seguente riga `- --kubelet-insecure-tls`.

Si può quindi avviare il metric server tramite il comando:

```
kubectl apply -f components.yaml
```

3.2 Implementazione dell'HPA

Al fine di implementare l'HPA è necessario creare il file yaml con le specifiche di scaling, le metriche da monitorare, e i target di scaling.

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: guestbook-frontend
  namespace: default
spec:
  maxReplicas: 10
  metrics:
  - resource:
      name: cpu
      target:
        averageUtilization: 50
        type: Utilization
    type: Resource
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend
```

File 3.1: frontend-hpa.yaml

Per avviare l'HPA si esegue:

```
kubectl apply -f frontend-hpa.yaml
```

Per controllare lo stato dell'HPA è possibile eseguire:

```
kubectl get hpa
```

```
NAME                    REFERENCE              TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
guestbook-frontend-hpa   Deployment/frontend   cpu: 1%/50%   1         10        3          41s
```

3.2.1 Verifica dello scaling down

Dopo poco tempo l'HPA ha portato le repliche al numero minimo possibile (1) dato che il carico di lavoro è stato nullo. Si è quindi verificato che lo scaling down funziona correttamente.

```
NAME                    REFERENCE              TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
guestbook-frontend-hpa   Deployment/frontend   cpu: 1%/50%   1         10        1          13m
```

É possibile controllare le azioni eseguite dall'HPA eseguendo il comando:

```
kubectl describe hpa guestbook-frontend-hpa
```

```
Name:                                                  guestbook-frontend-hpa
Namespace:                                             default
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Mon, 02 Sep 2024 16:34:00 +0200
Reference:                                             Deployment/frontend
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  1% (1m) / 50%
Min replicas:                                          1
Max replicas:                                          10
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from cpu resource utilization
(percentage of request)
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:
  Type    Reason             Age    From                       Message
  ----    ------             ----   ----                       -------
  Normal  SuccessfulRescale  9m53s  horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
```

3.2.2 Verifica dello scaling up

Per far attivare lo scaling up è necessario creare un carico che superi le soglie definite. Per fare ciò è possibile sfruttare l'immagine busybox per inviare richieste infinite a un servizio esposto:

```
kubectl run -i --tty load-generator --rm --image=busybox
    ↪ :1.28 --restart=Never -- /bin/sh -c "while␣sleep␣
    ↪ 0.0000000001;␣do␣wget␣-q␣-O-␣http://frontend/;␣
    ↪ done"
```

Il carico è salito al 118%:

```
NAME                    REFERENCE              TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
guestbook-frontend-hpa  Deployment/frontend    cpu: 118%/50%   1         10        1          36m
```

Dopo poco tempo il numero di repliche è aumentato ottenendo un carico sotto la soglia impostata:

```
NAME                    REFERENCE              TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
guestbook-frontend-hpa  Deployment/frontend    cpu: 44%/50%    1         10        3          37m
```