Delfi Sehidic, 2020

Kubernetes i Openstack

Inledning

Syftet med den här rapporten är att visa hur jag driftsatt en mjukvarutjänst i molnplattformen CSCloud (Openstack) med hjälp av Kubernetes. Tjänsten är en mockup av Reddit och består av sex mindre tjänster.

Bakgrund

Det som kommer krävas för applikationen är en MariaDB för inloggningsinformation, en MongoDB för bl.a inlägg och kommentarer, tre Pythontjänster som hanterar inloggning innehåll och frontend samt en Nginx reverse-proxy framför Pythontjänsterna. För att göra vissa konfigureringar smidigare kommer man även att behöva bifoga skript och filer.

Applikation kommer drivas av Kubernetes som är ett verktyg för container-orkestrering utvecklat av Google. Kubernetes gör det lätt att automatisera, skala och hantera tjänster som drivs av containrar. Docker kommer vara containerprogrammet som håller våra applikationer.
Vi kommer att behöva skapa en virtuell maskin som blir master-noden och tre virtuella maskiner som blir dess worker-noder. Sedan kommer vi att specificera hur applikationen ska struktureras i master-noden som sedan ansvarar för att delegera poddar och upprätthålla dem i de olika worker-noder som finns tillgängliga. Poddarna kommer att hålla de containrar som behövs. De är dock kortlivade och för att databasen ska fungera kommer vi att behöva en till maskin som databas. Den kommer att kommunicera med andra maskiner på subnätet via NFS. Vi kommer även att behöva en maskin som docker-registry där vi kan spara våra docker-images så att Kubernetes kan köra våra egna images med våra konfigurationer.

Detta innebär att vi kommer behöva följande maskiner i vårt moln:

Master node x 1
Worker node x 3
NFS x 1
Registry x 1

Guide

Skapa Kuberneteskluster

Logga in på OpenStack. Skapa ett Network med gruppen 172.16.0.0/24 som kopplas till internet via en router. Skapa en virtuell maskin för mastern. Jag valde senaste LTS Ubuntu som image. Ge den ett floating IP så att den även kan användas som jump machine. I mitt fall blev det 194.47.177.77. Vi passar på att initera den med en skriptfil i cloud init. Där installerar och startar vi Docker och Kubernetes. Vi kopplar den även till nätverket ovan i OpenStack.

Ladda ner SSH-nycklarna till maskinen och logga sedan in på den från din egen terminal via SSH. Då kan vi börja med att sätta upp Kubernetes.

Kör:

sudo kubeadm init --pod-network-cidr=192.168.0.0/16

kubeadm är ett verktyg som hjälper oss att sätta upp Kubernetes smidigt. IP-serien är default för Calico som vi snart återkommer till. Det är den serien podnätverket kommer att använda.

Kubeadm skriver ut följande i konsolen:

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Kör dem kommandona. Det är för att kunna använda klustret som vanliga användare. Sedan kan vi sätta upp de tre worker-noderna. De sätts upp likadant som master-noden förutom att vi i slutet av cloud-init-skriptet lägger till följande som vi fick av mastern för att de ska lägga till sig som slavar till mastern:

*kubeadm join 172.16.0.8:6443 --token p3b07r.2lwe2mg3prtywtgs *

--discovery-token-ca-cert-hash sha256:21c730258c37bab012e3763344b9c2bf1e0a17481bfc29d12b293eac3db5e19c

Det kan även vara en bra idé att köra fler skript i init-cloud för saker som behövs längre fram. T.ex installation av nfs-common

Vi måste även sätta upp ett nätverk för poddarna. Detta är för att Kubernetes försöker behandla poddarna ungefär som om de vore egna virtuella maskiner vilka då skulle ha egna IP-adresser. Tanken är att om en applikation ska kunna köra i en VM ska den kunna köras i en Kubernetes-pod. Problemet är att poddar placeras ut på olika noder och kan dela på en maskin. Därför skapar man virtuella IP-adresser som poddarna tilldelas. Det finns olika tjänster som löser detta på olika sätt. Vi kommer att använda Calico som podnätverk. Ladda ner YAML-filen för Calico:

curl https://docs.projectcalico.org/manifests/calico.yaml -O

YAML-filer används för att göra deployments i Kubernetes. En deployment beskriver ett särskilt tillstånd vi önskar ha. Kubernetes tar sedan ansvaret för att se till att det finns poddar som matchar det tillståndet. Vi sätter igång den med:

kubectl apply -f calico.yaml

Docker registry

Vi vill ha specifika images till våra deployments. Till exempel våra Pythontjänster behöver konfigureras särskilt istället för att bara köra Python rakt av. Istället för att ladda upp images till ett publikt repo ska vi skapa ett eget privat. Vi skapar en virtuell maskin och kallar den registry. Min fick lokal ip-adress 172.16.0.5. Se till att installera Docker i cloud-init när du skapar den.

Logga in på registry-maskinen via ssh och skapa en mapp:

mkdir -p /docker/repository

Skapa en container för registry image från dockerhub:

docker container run -d -p 5000:5000 --name registry -v /docker/registry:/var/lib/registry registry

I mastern och alla workernoder, redigera /etc/docker/daemon.json och lägg till:

{ “insecure-registries”: [“<IP-ADRESSEN>:5000”] }

Byt ut <IP-ADRESSEN> till den IP-adress registrymaskinen har.

Man får beakta att detta är en snabb lösning för guiden som är ganska osäker. I en riktig produktionsmiljö hade vi satt upp autentisering och dylikt för vårt registry.

Starta sedan om Docker på mastern:

sudo systemctl restart docker

Sedan när vi skapar egna Docker images för våra applikationer kan vi skicka dem till Registry och i YAML-filen för varje applikation specificera att den ska hämtas från Registry. Återkommer senare med hur vi skapar egna Docker images med Dockerfile.

NFS-server

Poddar i Kubernetes är kortlivade och tillståndslösa. Det innebär att om en pod kraschar så kommer en ny startas upp, men med förlorat tillstånd. Det blir inte bra om vi vill spara uppgifter, vilket är fallet med t.ex databaser. Därför skapar vi en till virtuell maskin i molnet som kommer ansvara för lagring. Den kommer att implementera protokollet Network File System som låter klienter på andra system komma åt filer på enheten. Jag döpte min till *nfs* och den fick en IP-adress 172.16.0.14 på det lokala nätverket. Antingen lägger du till kommandon i init-scriptet eller så installerar du manuellt, vilket jag kommer att göra.

Logga in på nfs via SSH genom mastern. Ladda ner nfs-common som är servern som implementerar protokollet:

sudo apt install nfs-kernel-server nfs-common -y

Sedan skapar vi de mappar som ska delas. Vi skapar en för både MariaDB och MongoDB.

sudo mkdir -p /nfs/mariadb/kubedata

sudo mkdir -p /nfs/mongodb/kubedata

Sätt rättigheterna så man kan komma åt dem:

sudo chown nobody:nogroup /nfs/mariadb/kubedata

sudo chmod 777 /nfs/mariadb/kubedata

sudo chown nobody:nogroup /nfs/mongodb/kubedata

sudo chmod 777 /nfs/mongodb/kubedata

Skapa sedan en fil /etc/exports där vi kan ställa in så att klienterna kommer åt filerna och skriv in:

*/nfs/mongodb/kubedata *(rw,sync,no_subtree_check)*

*/nfs/mariadb/kubedata *(rw,sync,no_subtree_check)*

Istället för * kan man sätta subnätverket som är lite mindre tillåtande och säkrare. Sedan kan vi exportera mapparna och starta om servern:

sudo exportfs -a

sudo systemctl restart nfs-kernel-server

Har man en brandvägg kan man behöva öppna upp den. Jag fick dock inactive när jag körde sudo ufw status.

Vill man testa att allt har fungerat kan man logga in i en av noderna, installera nfs-common, skapa mappen /mnt/hello och mounta den:

sudo mount <ip-adressen till nfs>:/nfs/mongodb/kubedata /mnt/hello

Byt ut <ip-adressen till nfs> till nfs-maskinens IP-adress. Testa att skapa en fil i mappen på nfs och se om den dyker upp i nodens motsvarande mapp.

Persistent Volume

För att Kubernetes ska veta att vi vill använda beständig lagring behöver vi konfigurera Persistent Volume (PV) och Persistent Volume Claim (PVC). Persistent Volume är lagring i klustret som tillhandahålls vanligtvis av en administratör. Det är det gränssnitt vi har mot själva lagringen, i detta fall vår NFS-server vi konfigurerade innan. De har en livscykel som är oberoende av poddarna som använder den. Persistent Volume Claim är ett gränssnitt mot användare som önskar använda lagring.
Dessa skapas precis som Deployments bäst med YAML-filer. De finns bifogade som storage-claims.yaml. Om du öppnar dem kan vi notera att för PV bestämmer vi namn, hur mycket utrymme som finns tillgängligt, access modes och att vi vill använda nfs och vilket ip-nummer den har och mount path. I PVC specificerar vi namn, access modes och hur mycket utrymme vi vill ha. Kubernetes kommer sedan att matcha våra PVC med en PV om det finns en. Vi kan filtrera urvalet ännu mer genom att sätta fler selectorer.
Access modes specificerar hur vi kan skriva till lagringen. Just i vårt fall klarar NFS-protokollet att hantera flera klienter som skriver eller läser lagringen, men hade vi använt till exempel Flocker kan bara en nod åt gången läsa/skriva till disken.

Skapa yaml-filen i mastern, förslagsvis med “sudo nano storage_claims.yaml” och klistra in. Ändra IP-adressen under nfs och server till samma som IP-adressen för din nfs-server. Aktivera den sedan med:

kubectl apply -f storage_claims.yaml

MariaDB

Nu är det dags att börja starta våra applikationer. De finns bifogade som en tar.gz. Skicka över den till mastern med scp och unzippa den.
Förslagsvis med detta kommande från din lokala maskin i samma mapp som filen finns:

scp -i <din SSH-nyckel> exam3-master.tar.gz ubuntu@<Floating IP till mastern>:/home/ubuntu/apps.tar.gz

Unzippa den sedan. MariaDB-tjänsten finns i mappen authdb. Vi behöver skapa en ConfigMap som innehåller de miljövariabler som MariaDB kräver, skapa en Dockerfile för att skapa en image vi kan pusha till repot och en YAML-fil för att skapa en Service.

En ConfigMap kan hålla nyckel/värde-par som kan konsumeras av poddar som miljövariabler. Detta kan vara väldigt smidigt om man vill sätta olika konfigurationsdata för samma applikation vid olika tillfällen. Till exempel om en app använder localhost under developmentfasen och en IP-adress under produktionsfasen. I vårt fall kommer databaserna behöva vissa variabler för att initieras som användarnamn och lösenord och Pythonskripten konsumerar vissa variabler för att vi ska slippa hårdkoda dem.

En Dockerfile är en textfil utan filändelse som berättar hur en Dockerimage ska byggas. Den använder ganska simpel syntax. Vi måste använda FROM och sedan den basimage vi vill bygga från. Det brukar vara ett programmeringsspråk eller en tjänst. COPY kopierar en mapp från mappen vi jobbar i och gör den tillgänglig i imagen. EXPOSE öppnar upp nätverksportar. VI kommer att öppna 3306 som är standard för MariaDB.

Skapa en Dockerfile med följande i mappen authdb:

FROM mariadb

COPY init.sql /docker-entrypoint-initdb.d

EXPOSE 3306

Skapa en Docker image från den:

sudo docker build .

Kör “docker images” och ta tagen från den som vi just skapade (none).

Tagga den och pusha den till registry:

sudo docker tag <taggen> <IP-adress till registry>:5000/mariasvc

sudo docker push <IP-adress till registry>:5000/mariasvc

Imagen går nu att nå från repot. Om man vill kan man pröva att göra en deployment för att se att det fungerar.

kubectl create deployment mariasvc —image=<IP-ADRESS REGISTRY>:5000/mariasvc

Den kommer dock att få error eftersom databasen behöver miljövariabler för att fungera.

Service och ConfigMap

För att våra miljövariabler ska fungera och tjänsterna kommunicera med varandra behöver vi skapa en beständig IP-adress. Pods är som sagt kortlivade och kan få nya adresser, vilket inte är bra eftersom vår databas ger en tjänst som andra poddar är beroende av. Vi löser detta med en Service i Kubernetes.

Service skapar ett extra nätverkslager i Kubernetes. Poddarna kommunicerar via podnätverket. Problemet är att en podd kan behöva en tjänst från ett annat set poddar men deras adresser är inte beständiga. Hur kan vi nå poddarna? Vi skapar ett till nätverk och ger ett set poddar en till beständig IP-adress, nämligen ett ClusterIp. Det blir då kube-proxys ansvar att ta emot trafiken och skicka den vidare till rätt pod.

Se den bifogade filen mariadb-svc.yaml. Vi specificerar först en Service med namn mariasvc-rest. Som selector använder vi mariadb. Selectorn måste matcha vår Deployment för att trafiken ska sändas rätt. Den använder TCP och lyssnar på MariaDBs default port 3306. Vi skulle kunna köra denna Service och sedan få ClusterIP:et att använda i ConfigMapen, men jag valde att använda DNS-namnet istället. Det blir i mitt fall mariadb.default.svc.cluster.local. Om det inte fungerar, kör “kubectl get svc” och ta ClusterIp och skriv in det istället. Isåfall kan det vara vettigt att låta Service få vara en egen fil så att inte ett nytt ClusterIp skapas när vi vill göra ändringar i vår Deployment.

I ConfigMapen kan vi sätta de miljövariabler som MariaDB behöver för att köra.
Vi applyar sedan vår ConfigMap, Service och Deployment.

Byt ut IP-adressen till ditt registry under containers, image.
Kör “kubectl apply -f mariadb-svc.yaml”

För att kontrollera att allt fungerat kan man logga in på podden. Kör “kubectl get pods -o wide” och hitta namnet på podden. Logga sedan in på den, i mitt fall med:

kubectl exec --stdoin --tty mariadb-66f6765d58-gdqgj -- /bin/bash

Sedan kan man starta MariaDB och kontrollera att tables finns där.

mysql --user=<user_name> --password=<your_password db_name>

SHOW TABLES;

LoginSVC

En av Pythontjänsterna sköter inloggning bland annat. Vi sätter upp den också.
Den kommer att behöva en Dockerfile som pushas till repot.

I mappen loginsvc på mastern skapar man en Dockerfil:

```
FROM python

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY login_svc.py .

EXPOSE 7070
```

ENTRYPOINT [“gunicorn","-b","0.0.0.0:7070","-w","4","login_svc:app"]

Denna kommer att hämta Python som base-iamge, kopiera dependencies från requirements och installera dem, ta in Pythonskriptet login.svc, öppna port 7070 som applikationen lyssnar på (det kan man se längst ner i Pythonskriptet) och köra applikationen via gunicorn.

Skapa sedan en image med
sudo docker build .

Tagga builden och pusha den till registry precis som vi gjorde med MariaDB.

Vi skapar sedan en YAML-fil som specificerar dess Service och Deployment.
Vänligen se den bifogade filen login-svc.yaml.

Kör “kubectl get svc” och anteckna ClusterIp till mariasvc-rest,

Du behöver ändra följande i yaml-filen:
Under data ska MARIADB_SERVER_ADDR vara mariasvc-rests ClusterIp.
Under containers ska image ha IP-adressen till ditt Docker registry.

kör “kubectl apply -f login-svc.yaml”

MongoDB

Nu ska vi sätta upp databasen för MongoDB.
Applya den bifogade filen mongosvc.yaml. Du behöver inte göra några ändringar i den.

Kör “kubectl get svc” och för att få alla ClusterIP.

Öppna den bifogade filen mongo.yaml. Ändra följande:

LOGIN_SERVER_ADDR ska vara ClusterIp till loginsvc-rest.
MONGO_SERVER_ADDR ska vara ClusterIP till mongosvc-rest.

Kör “kubectl apply -f mongo.yaml”

PosterSVC

Poster-tjänsten används för att posta saker till vårt Minireddit. Vi gör samma sak för poster-tjänsten som login-tjänsten. Gå in i mappen postersvc. Skapa en Dockerfile med innehållet:

FROM python:latest
COPY post_svc.py ./
COPY requirements.txt ./
RUN pip install -r requirements.txt
ENTRYPOINT [“gunicorn","-b","0.0.0.0:7075","-w","4","post_svc:app"]

Den laddar ner senaste Python som *base image*, kopierar in Pythonskriptet, kopierar in textfilen med requirements, installerar alla dependencies och kör gunicorn med post_svc:app som utgångspunkt för applikationen.

Tagga den och pusha till Docker registry.

Använd sedan YAML-filen poster-svc.yaml för att skapa vår ConfigMap, Deployment och Service. Ändra följande i filen:
LOGIN_SERVER_ADDR ska vara ClusterIP till loginsvc-rest.
MONGO_SERVER_ADDR ska vara ClusterIP till mongosvc-rest.
Image under Containers ska ha ip-adressen till ditt Docker-registry.

Kör “kubectl apply -f poster-svc.yaml”

Frontend

Frontend genererar HTML. Det finns bifogat frontsvc.yml och frontend.yml.
Kör “kubectl apply -f frontsvc.yml” och sedan “kubectl get svc” och notera ClusterIP till servicen.

Gå in i mappen frontend och skapa denna Dockerfile:

```
FROM python:3.8
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY web_svc.py .
ADD templates ./templates
EXPOSE 7080
ENTRYPOINT [“gunicorn","-b","0.0.0.0:7080","-w","4","web_svc:app"]
```

Kör:
sudo docker build .
sudo docker images
Ta tagnumret. Tagga den och pusha till Docker registry.

sudo docker tag <Taggen till din Image> <IP-adress till ditt Docker registry>:5000/frontend

sudo docker push <IP-adress till ditt Docker registry>:5000/frontend

Öppna frontend.yml och ändra följande:

LOGIN_SERVER_ADDR ska vara ClusterIP till loginsvc-rest.
POST_SERVER_ADDR ska vara ClusterIP till postersvc-rest.
FRONTEND_SERVER_ADDR ska vara ClusterIP till frontsvc.
IP-adressen under Container, Image ska vara till ditt Docker Registry.

Kör en curl för att se att den fungerar.

curl <ClusterIp till din frontend>:7080

Bör returnera HTML.

Experiment

Dags att fylla den med data. Gå in i mappen Experiment. Ändra i runex.py så att miljövariablerna stämmer. Kör “sudo nano runex.py”. Ändra följande:
login_host = “<ClusterIP till loginsvc-rest>”
login_port = '7070'
poster_host = ‘<ClusterIP till postersvc-rest>’
poster_port = '7075'

Testa sedan att köra den med “python3 runex-py”.
Curla igen och se om subreddits har skapats.

Nginx

Kör “kubectl get svc” och notera ClusterIP till frontsvc.
Öppna proxy.yml och ändra ip-adressen server under data, config till frontsvcs.

Apply yaml-filen med NodePort för Nginx.
Gå till Openstack och skapa en lastbalanserare.
Döp den till Minireddit. Sätt samma subnät som poddarna har.
Gå vidare till Listener Details och sätt protokollet till HTTP och porten till 80.
Gå till Pool Details och döp den till Kubenodes och sätt ROUND_ROBIN som balanserarmetod.
Gå till Pool Members och lägg till de tre worker-noderna.

Gå tillbaka till terminalen och kör kubectl get svc. Kolla vilka portar som proxy är öppen på. Det bör vara port 80 och en som börjar på 3. Välj den senare och sätt den porten på varje nod istället för 80.
Gå till Monitor Details och byt Monitor Type till TCP.
Tryck sedan på Create Load Balancer.
Gå in under Load Balancers igen och ge den ett floating ip.
Skapa en security group för HTTP.

Gå in på tjänsten (tryck på Minireddit), tryck på länken (står efter port id), tryck på edit port, tryck på security groups, lägg till security group HTTP som du skapade innan.

Öppna en ny flik i webbläsaren och skriv in ditt floating ip. Du bör nu se din tjänst uppe!