Guide Deploy ứng dụng lên K8s của google cloud.

1. Cài đặt các thành phần thiết yếu:
   1. Đầu tiên, ta cần cài đặt Google Cloud SDK Shell từ trang chủ của google cloud theo guilde sau : <https://cloud.google.com/sdk/docs/quickstart-windows>
   2. Enable Kubernetes Engine API
      1. Truy cập link https://console.cloud.google.com/projectselector/kubernetes?\_ga=2.233066910.161647076.1589101545-575099250.1587279164&\_gac=1.220647786.1589103935.Cj0KCQjwzN71BRCOARIsAF8pjfjYGXDZ6kN8DUQL4vlv3RYX\_KgcKF3L9m-v\_LN0uHylGa0pjrWqr04aAmx1EALw\_wcB
      2. Tạo hoặc chọn 1 project
      3. Đợi cho API và các service được bật lên. Sẽ mất một chút thời gian
2. Deploy container lên k8s
   1. Đóng gói ứng dụng vào Docker image
      1. Thiết lập biến môi trường PROJECT\_ID tới project của bạn qua câu lệnh: export PROJECT\_ID=project-id hoặc ta không cần, sử dụng trực tiếp tên project vào các chỗ cần điền PROJECT\_ID như phía sau sử dụng
      2. Tạo container image: docker build -t gcr.io/${PROJECT\_ID}/ service-registry:v1 .
   2. Upload image lên registry
      1. docker push gcr.io/${PROJECT\_ID}/}/ service-registry:v1
   3. Tạo một container cluster: Ta đã có image trên registry của g-cloud. Giờ ta cần tạo ra 1 cluster để chạy image đó. Một cluster bao gồm một nhóm Compute Engine VM instance chạy K8s. Ở hướng dẫn này ta sẽ sử dụng gcloud tool đã tải ở bước 1.
      1. Thiết lập projectId và Compute Engine zone cho gcloud tool:
         1. gcloud config set project $PROJECT\_ID
         2. gcloud config set compute/zone compute-zone
      2. Tạo cluster tên là api-cluster : gcloud container clusters create api-cluster --num-nodes=2
   4. Deploy ứng dụng đó lên cluster: Thường deploy app lên cluster,ta sẽ tạo ra 1 deployment cho nó. Và để cấu hình cho app đó, ta sử dụng 1 file .yaml để config

Ex:

---

apiVersion: "v1"

kind: "ConfigMap"

metadata:

  name: "registry-service-config"

  namespace: "default"

  labels:

    application: "pm"

data:

  SERVER\_PORT: "8761"

  EUREKA\_CLIENT\_REGISTER\_WITH\_EUREKA: "false"

  EUREKA\_CLIENT\_FETCHREGISTRY: "false"

  EUREKA\_CLIENT\_SERVER\_WAITTIMEINMSWHENSYNCEMPTY: "0"

  EUREKA\_SERVER\_EVICTION\_INTERVAL\_TIMER\_IN\_MS: "1000"

---

apiVersion: "apps/v1"

kind: "Deployment"

metadata:

  name: "registry-service"

  namespace: "default"

  labels:

    application: "pm"

spec:

  replicas: 1

  selector:

    matchLabels:

      application: "pm"

  template:

    metadata:

      labels:

        application: "pm"

    spec:

      containers:

      - name: "service-registry"

        image: "gcr.io/spartan-thunder-272123/service-registry:v2"

        ports:

        - containerPort: 8761

        env:

        - name: \_JAVA\_OPTIONS

          value: -Dsun.net.inetaddr.ttl=30 -Deureka.instance.preferIpAddress=true

        - name: "SERVER\_PORT"

          valueFrom:

            configMapKeyRef:

              key: "SERVER\_PORT"

              name: "registry-service-config"

        - name: "EUREKA\_CLIENT\_REGISTER\_WITH\_EUREKA"

          valueFrom:

            configMapKeyRef:

              key: "EUREKA\_CLIENT\_REGISTER\_WITH\_EUREKA"

              name: "registry-service-config"

        - name: "EUREKA\_CLIENT\_FETCHREGISTRY"

          valueFrom:

            configMapKeyRef:

              key: "EUREKA\_CLIENT\_FETCHREGISTRY"

              name: "registry-service-config"

        - name: "EUREKA\_CLIENT\_SERVER\_WAITTIMEINMSWHENSYNCEMPTY"

          valueFrom:

            configMapKeyRef:

              key: "EUREKA\_CLIENT\_SERVER\_WAITTIMEINMSWHENSYNCEMPTY"

              name: "registry-service-config"

        - name: "EUREKA\_SERVER\_EVICTION\_INTERVAL\_TIMER\_IN\_MS"

          valueFrom:

            configMapKeyRef:

              key: "EUREKA\_SERVER\_EVICTION\_INTERVAL\_TIMER\_IN\_MS"

              name: "registry-service-config"

        - name: "EUREKA\_INSTANCE\_PREFERIPADDRESS"

          value: "true"

---

apiVersion: "autoscaling/v2beta1"

kind: "HorizontalPodAutoscaler"

metadata:

  name: "registry-service-hpa-ehff"

  namespace: "default"

  labels:

    application: "pm"

spec:

  scaleTargetRef:

    kind: "Deployment"

    name: "registry-service"

    apiVersion: "apps/v1"

  minReplicas: 1

  maxReplicas: 1

  metrics:

  - type: "Resource"

    resource:

      name: "cpu"

      targetAverageUtilization: 80

Sau đó ta chỉ cần chạy lệnh : kubectl apply -f service-registry.yaml và việc còn lại là chờ cho app được deploy lên.

Ngoài ra, để đẩy ứng dụng ra internet, ta cần tạo ra 1 service. Service thường hoạt động kèm theo một loadbalance để cân bằng tải. Ví dụ sau em sẽ deploy một angular webapp lên k8s.

Đầu tiên ta build 1 image cho con app angular này:

File Dockerfile:

# Stage 0, "build-stage", based on Node.js, to build and compile the frontend

FROM node as build-stage

WORKDIR /app

COPY package\*.json /app/

RUN npm install

COPY ./ /app/

ARG configuration=production

RUN npm run build -- --output-path=./dist/out --configuration $configuration

# Stage 1, based on Nginx, to have only the compiled app, ready for production with Nginx

FROM nginx:1.15

#Copy ci-dashboard-dist

COPY --from=build-stage /app/dist/out/ /usr/share/nginx/html

#Copy default nginx configuration

COPY ./nginx-custom.conf /etc/nginx/conf.d/default.conf

File Deployment:

apiVersion: apps/v1beta1

kind: Deployment

metadata:

  name: front-end

  labels:

    app: front-end

spec:

  replicas: 1

  selector:

    matchLabels:

      app: front-end

  template:

    metadata:

      labels:

        app: front-end

    spec:

      containers:

        - name: pront-end

          image: gcr.io/spartan-thunder-272123/front-end:v6

          imagePullPolicy: Always

          ports:

          - containerPort: 80

Các thông số cơ bản ta cần quan tâm như:

* Replicas: số lượng replicas tối đa
* Spec: các thông số specification
  + Containers: các thông số container

Sau khi đã có bản deployment, ta cần tạo ra service cho app này

File service:

apiVersion: v1

kind: Service

metadata:

  name: front-end-service

spec:

  type: ClusterIP

  ports:

  - name: port-80

    port: 80

    protocol: TCP

  selector:

    app: front-end

* Tham số ta cần quan tâm chính selector. Nó map chính sác với deployment của ta.

Cuối cùng là loadbalance cho app:

apiVersion: v1

kind: Service

metadata:

  name: front-end-load-balance

spec:

  type: LoadBalancer

  ports:

  - name: port-80

    port: 80

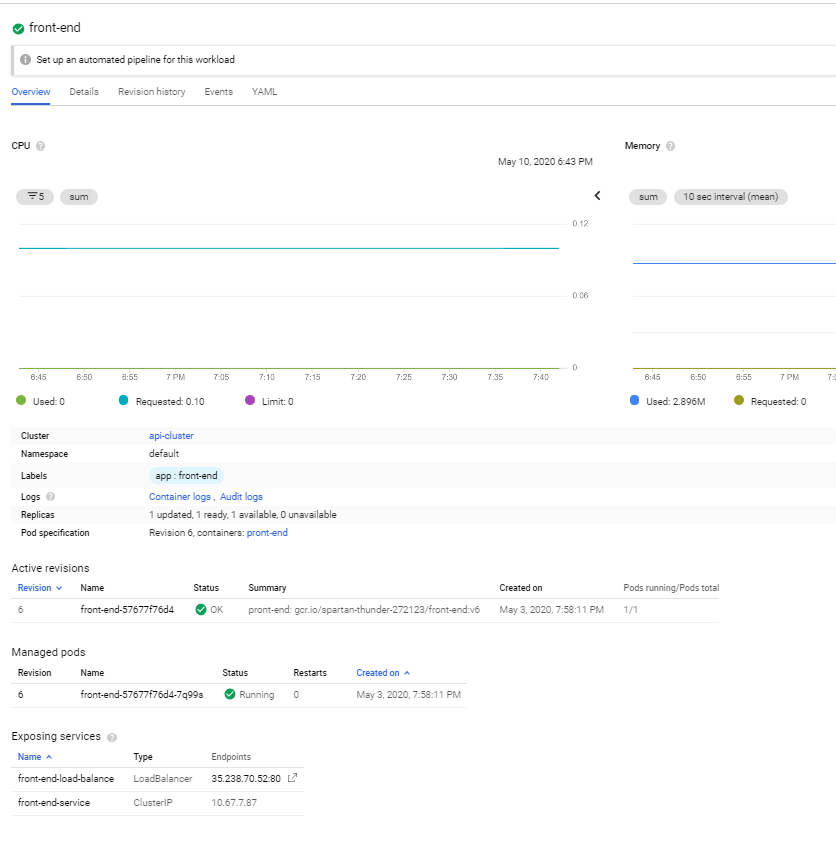
    protocol: TCP

  selector:

    app: front-end

Tương tự service, ta cũng cần selector để trỏ tới đúng thành phần.

Và kết quả ta thu được sẽ là :



Ta có thể truy cập IP của LoaBalancer kia trực tiếp trên internet.