Département informatique

Master 1 Cloud et réseaux virtuels

Projet AutoScaling et IaC

Auteur :

- Riad FELIH 21306261

1. Déploiement de Redis

Redis main

Pour Redis j'ai utilisé l'architecture main et réplicas avec auto-scaling sur les répliques.

Pour le main redis le déploiement est simple, j'ai utilisé une image redis offert par docker "redis:7.2" avec une réplique.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-main
  labels:
    name: redis-main
spec:
  replicas: 1
  selector:
    matchLabels:
      name: redis-main
  template:
    metadata:
      labels:
        name: redis-main
    spec:
      containers:
      - name: redis
        image: 'redis:7.2'
        ports:
        - containerPort: 6379
```

Le redis main est associer avec le service "redis-main-service" accessible à travers le port 6379.

```yaml
apiVersion: v1
kind: Service
metadata:
 name: redis-main-service
spec:
 type: ClusterIP
 ports:
  - protocol: TCP
    port: 6379
    targetPort: 6379
 selector:
  name: redis-main
```

Redis replicas

Les répliques redis le sont déployées avec la même image du main, le déploiement commence par 2 répliques.

Dès qu'un pod sera allumé il se connecte avec le main en utilisant la command :

"**redis-server –slaveof redis-main-service.default.svc.cluster.local 6379 –port 6380**"

Ici les répliques executent le serveur redis sur le port 6380 car je n'ai pas pu remplacer le serveur redis par default qui exécute sur 6379, donc j'ai dû créer un autre serveur sur 6380 et exposer les pods sur ce port.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-replica
  labels:
    name: redis-replica
spec:
  replicas: 2
  selector:
    matchLabels:
      name: redis-replica
  template:
    metadata:
      labels:
        name: redis-replica
    spec:
      containers:
        - name: redis
          image: 'redis:7.2'
          ports:
            - containerPort: 6380
          command:
            - redis-server
          args:
            - "--slaveof"
            - "redis-main-service.default.svc.cluster.local"
            - "6379"
            - "--port"
            - "6380"
```

Le déploiement commence par 2 répliques, Mais avec un intervalle de réplication entre 2 et 10 répliques, la duplication se fait en basant sur le taux d'utilisation de CPU des pods, le taux de CPU ciblé est 50%, le nombre des pods peut augmenter ou baisser selon la charge. Ceci est fait avec un déploiement d'un HPA (Horizontal Pod Autoscaler).

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: redis-replica-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: redis-replica
  minReplicas: 2
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Les répliques redis sont accessibles par le service "redis-replica-service" à travers le port 6379

```yaml
apiVersion: v1
kind: Service
metadata:
 name: redis-replica-service
spec:
 type: ClusterIP
 ports:
  - protocol: TCP
    port: 6379
    targetPort: 6380
 selector:
  name: redis-replica
```

2. Déploiement de Node Server

L'application node utilise mon image sur dockerhub "riadflh/node-app-img:latest" avec une réplique. Les conteneurs sont exposés sur le port 3000, et des variables d'environnement pour se connecter au redis. J'ai utilisé le nom DNS des services de redis main et redis replica.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: node-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: node-app
  template:
    metadata:
      labels:
        app: node-app
    spec:
      containers:
      - name: node-app
        image: riadflh/node-app-img:latest
        ports:
        - containerPort: 3000
        env:
        - name: REDIS_URL
          value: "redis://redis-main-service.default.svc.cluster.local:6379"
        - name: REDIS_REPLICAS_URL
          value: "redis://redis-replica-service.default.svc.cluster.local:6379"
        - name: PORT
          value: "3000"
```

Le déploiement commence par 1 réplique, Mais avec un intervalle de réplication entre 1 et 10 répliques, la duplication se fait en basant sur le taux d'utilisation de CPU des pods, le taux de CPU ciblé est 50%, le nombre des pods peut augmenter ou baisser selon la charge. Ceci est fait avec un déploiement d'un HPA (Horizontal Pod Autoscaler).

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: node-app-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: node-app
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

Les pods de node sont accessibles par le service "node-service" à travers le port 80 qui de type "LoadBalancer".

```yaml
apiVersion: v1
kind: Service
metadata:
  name: node-service
spec:
  selector:
    app: node-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

Et le service est accessible à travers le port 8080 de la machine hôte, donc le serveur node sera accessible sur “http://localhost:8080”. Ceci est fait grâce au port forwarding avec kubectl :

“**microk8s kubectl port-forward service/node-service 8080:80**”

On peut voir le autoscaling de hpa de node et redis avec la commande **“microk8s kubectl get hpa”**

```
root@riad-VirtualBox:/home/riad# microk8s kubectl get hpa
NAME                       REFERENCE                     TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
node-app-autoscaler        Deployment/node-app           3%/50%    1         10        1          5d20h
redis-replica-autoscaler   Deployment/redis-replica      3%/50%    2         10        2          5d22h
root@riad-VirtualBox:/home/riad#
```

3. Déploiement de react app

L’application react utilise mon image sur dockerhub “riadflh/react-app-img:latest” avec une réplique. Les conteneurs sont exposés sur le port 80.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: react-app
  template:
    metadata:
      labels:
        app: react-app
    spec:
      containers:
      - name: react-app
        image: riadflh/react-app-img:latest
        ports:
        - containerPort: 80
```

L’application react est accessible par le service “react-service" à travers le port 80 qui de type “LoadBalancer”.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: react-service
spec:
  selector:
    app: react-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Et le service est accessible à travers le port 80 de la machine hôte, donc l'application react sera accessible sur "http://localhost". Ceci est fait grâce au port forwarding avec kubectl :

"**microk8s kubectl port-forward service/react-service 80:80**"

4. Monitoring Grafana & prometheus

Pour le monitoring j'ai utilisé l'api "/metrics" de node server, et redis exporter pour récupérer les données de redis.

J'ai déployé 2 redis exporters, un pour le redis main et un pour les répliques secondaires .

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-main-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-main-exporter
  template:
    metadata:
      labels:
        app: redis-main-exporter
    spec:
      containers:
        - name: redis-main-exporter
          image: oliver006/redis_exporter:v1.24.0
          args:
            - "--redis.addr=redis-main-service.default.svc.cluster.local:6379"
          ports:
            - containerPort: 9121

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-replica-exporter
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis-replica-exporter
  template:
    metadata:
      labels:
        app: redis-replica-exporter
    spec:
      containers:
        - name: redis-replica-exporter
          image: oliver006/redis_exporter:v1.24.0
          args:
            - "--redis.addr=redis-replica-service.default.svc.cluster.local:6379"
          ports:
            - containerPort: 9121
```

Et chaque exporter est associé avec un service accessible à travers le port 9121.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-main-exporter
spec:
  selector:
    app: redis-main-exporter
  ports:
    - protocol: TCP
      port: 9121
      targetPort: 9121

---

apiVersion: v1
kind: Service
metadata:
  name: redis-replica-exporter
spec:
  selector:
    app: redis-replica-exporter
  ports:
    - protocol: TCP
      port: 9121
      targetPort: 9121
```

Prometheus est déployé et associé avec un service accessible a travers le port 9090 et associe avec une configuration (configure map) pour définir les Jobs et Targets de serveur node, redis exporter pour le main, et redis exporter pour les secondaires.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - name: prometheus
        image: prom/prometheus:v2.20.1
        ports:
        - containerPort: 9090
        volumeMounts:
        - name: prometheus-config
          mountPath: /etc/prometheus/prometheus.yml
          subPath: prometheus.yml
      volumes:
      - name: prometheus-config
        configMap:
          name: prometheus-config
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 5s
    scrape_configs:
      - job_name: "node-app"
        static_configs:
          - targets: ["node-service.default.svc.cluster.local"]
      - job_name: "redis-main"
        static_configs:
          - targets: ["redis-main-exporter.default.svc.cluster.local:9121"]
      - job_name: "redis-replica"
        static_configs:
          - targets: ["redis-replica-exporter.default.svc.cluster.local:9121"]
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: prometheus
spec:
  selector:
    app: prometheus
  ports:
    - protocol: TCP
      port: 9090
      targetPort: 9090
```

Finalement j’ai fait le déploiement de Grafana avec son configuration (configmap) qui d’écrit la datasource Prometheus.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana:7.1.5
        ports:
        - containerPort: 3000
        volumeMounts:
        - name: grafana-datasources
          mountPath: /etc/grafana/provisioning/datasources
      volumes:
      - name: grafana-datasources
        configMap:
          name: grafana-datasources
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-datasources
data:
  datasources.yml: |
    apiVersion: 1
    datasources:
      - name: prometheus
        type: prometheus
        access: proxy
        orgId: 1
        url: http://prometheus.default.svc.cluster.local:9090
        basicAuth: false
        isDefault: true
        editable: true
```

service accessible via le port 3000, Et le service est accessible à travers le port 3000 de la machine hôte, donc Grafana sera accessible sur “http://localhost:3000”. Ceci est fait grâce au port forwarding avec kubectl :

“**microk8s kubectl port-forward service/grafana 3000:3000**”

```yaml
apiVersion: v1
kind: Service
metadata:
  name: grafana
spec:
  selector:
    app: grafana
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: LoadBalancer
```

Voici 2 panels que j'ai crée dans une Dashboard grafana pour mesure le CPU usage pour les 3 (node server, redis main, redis replicas) et l'autre panel pour mesurer la ram active utilise par redis main et repicas.