Kubernetes

Facef - 2020

Kubernetes Parte 2

- API Kubernetes (yaml files);
- Configurações e segredos;
- Health check;
- Alocação de recursos;
- Auto *scaling* horizontal.

Material adicional https://github.com/diegofernandes/k8s-facef

Kubernetes - API

- Comunicação com o Control Plane, gerenciamento do cluster e objetos;
- HTTP API, JSON, GRPC, YAML
- kubeclt, client-libs, lens, etc.

Via YAML

- Campos Requeridos:
  - apiVersion - Versão da Kubernetes api que estamos usando para criar o objeto;
  - kind - tipo do objeto que estamos criando;
  - metadata - Informações que ajudam a identificar o objeto, nome, UID, namespace, etc;
  - spec - Qual o estado do objeto deve ser criado;
    - Cada tipo e objeto tem suas especificações(kubernetes-api-specs)
    - Ex: Pod ou Deployments

Exemplo 1

- baixar o template arquivo hello-api-deployment.yaml no repositorio https://github.com/diegofernandes/k8s-facef/
- **kubectl apply -f hello-api-deployment.yaml**
- **kubectl diff -f hello-api-deployment.yaml**
- baixar o template arquivo hello-api-service.yaml no repositorio https://github.com/diegofernandes/k8s-facef/
- **kubectl apply -f hello-api-service.yaml**
- **kubectl diff -f hello-api-service.yaml**

Configurações (ConfigMaps)

- Armazena configurações não confidenciais;
- Chave valor;
- Variáveis de Ambiente;
- Parâmetros de comandos;
- Arquivos de configurações via volume;
- Não temos Spec e sim data(Chaves e valores).

Exemplo 2

- Vamos usar os templates da pasta "config" https://github.com/diegofernandes/k8s-facef/tree/master/config
- baixar o template arquivo hello-api-config.yaml
- **kubectl apply -f hello-api-config.yaml**
- **kubectl get configmaps**
- baixar o template arquivo hello-api-deployment.yaml
- **kubectl apply -f hello-api-deployment.yaml**

Configurações - Segredos

- Armazena dados sensíveis, senhas, tokens, certificados;
- Chave Valor;
- Variáveis de Ambiente;
- Parâmetros de comandos;
- Arquivos de configurações via volume;
- Segregação de configurações;
- Permissionamento dos segredos.

Exemplo 3

- Vamos usar os templates da pasta "config" https://github.com/diegofernandes/k8s-facef/tree/master/secrets
- **kubectl create secret generic hello-api-secret --from-literal=username=devuser --from-literal=password='devuser'**
- **kubectl apply -f hello-api-deployment.yaml**
- **echo 'valor' | base64**
- **echo 'valor' | base64 --decode**
- **curl -v -u <user>:<pass> http://<ip>:porta**

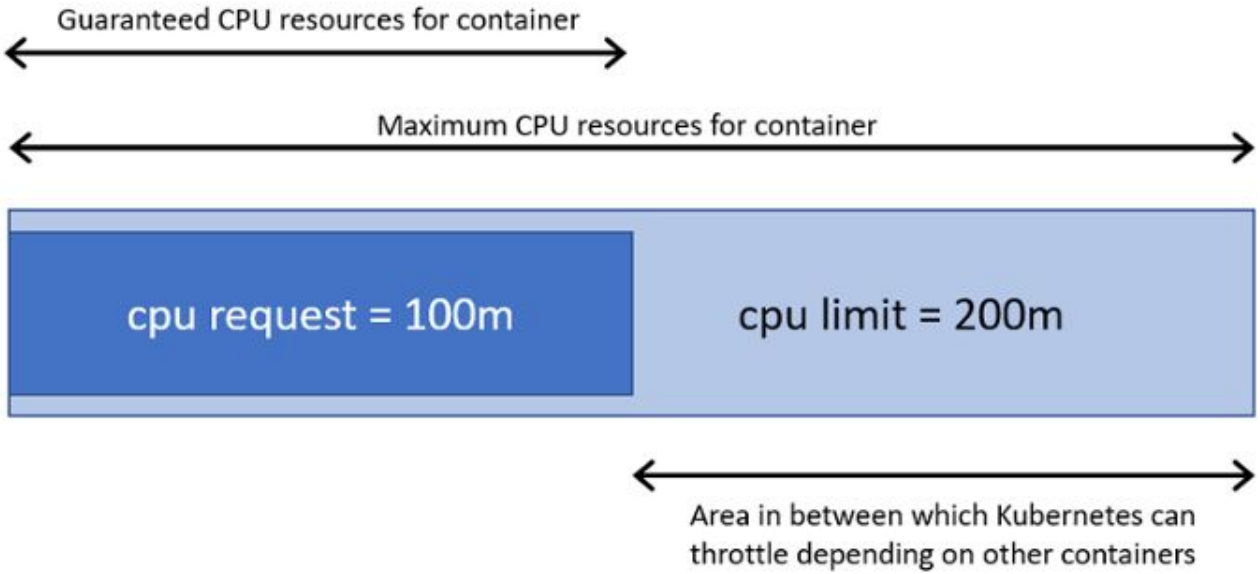
Health Check

- Checa a saudade da aplicação;
- Reinicia o Pod;
- Liveness - checa se o pod está vivo;
- Readiness - checa se o pod acessível;
- Comando, HTTP GET, TCP Socket.

Exemplo 4

- Vamos usar os templates da pasta "config": https://github.com/diegofernandes/k8s-facef/tree/master/healthcheck
- **kubectl apply -f hello-api-deployment.yaml**
- **kubectl get pods -w**

Alocação de recursos



Alocação de recursos

- Request - Quantidade garantida;
- Limit - Quantidade máxima que o pod pode usar(gordura);
- Dependência do **metric-server** para acompanhar a utilização e possíveis autoscaling;
- Estouros de memória causam morte do processo OOMKilled;
- Estouros de CPU causam CPU Throttling.

Exemplo 5

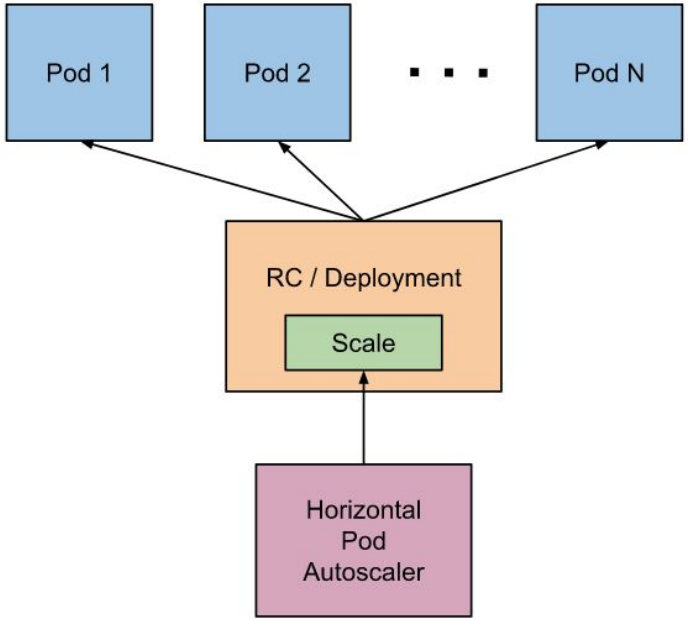
- Vamos usar os templates da pasta "resource": https://github.com/diegofernandes/k8s-facef/tree/master/resource
- **microk8s enable metrics-server** ou **minikube addons enable metrics-server**
- **kubectl apply -f pod-memory.yaml**
- **kubectl top pod memory-demo**
- **kubectl delete -f pod-memory.yaml**
- **kubectl apply -f pod-cpu.yaml**

Alocação de recursos

- Unidade de CPU:
  - 1 AWS vCPU
  - 1 GCP Core
  - 1 Azure vCore
  - 1 Hyperthread on a bare-metal Intel processor with Hyperthreading
  - 100m = 0.1vCore
- Unidade Memória:
  - Bytes
  - Ei, Pi, Ti, Gi, Mi, Ki
  - Ex 128Mi, 1Gi

Auto *scaling* horizontal.

- Por métricas dos pods CPU/Memória;
- Por métricas customizadas;
- Por métricas externas;
- Por várias métricas;
- Dependência do Metrics-Server.



Exemplo 6

- Vamos usar os templates da pasta "hpa" https://github.com/diegofernandes/k8s-facef/tree/master/hpa
- **kubectl apply -f hello-api-deployment.yaml**
- **kubectl autoscale deployment hello-api-deployment --min 1 --max 10 --cpu-percent 50**
- **kubectl get services**
- **kubectl get pods -w**
- **kubectl top pods**

Gerando volume

- **kubectl run -it --rm load-generator --image=busybox /bin/sh**
- **while true; do wget -q -O- http://hello-api-service:8080; done**

Exercício Avaliação

- Com base no exercício anterior (deployment do Star Wars api)
- Escreva o deployment em yaml
- Escreva o service em yaml
- Versione os yaml no repositório git do fonte
- Envie o link do repositório para diego.osse@gmail.com