21. 04. 2024

"svc" ((Pod) RC) RS.

Advanced sch.

(Volumes) hostPath, EmptyDir volume. pvi pvc sc.

*******IMP*********

When we are running applications in Kubernetes , you may need to pass dynamic values to your application at the run time.

These are called as application configurations.

In k8s, you can store the configurational data in an object called as CONFIGMAP

COnfigmap , stores the data in the form a key-value pair. This data can be passed to your container application

Dockerfile → ARG → image build ↓ dynam container

ENV →

image — env → dev Env — Env. → stage ENV

(Key: value.) → { plain config data — ConfigMap; base64 Encode value — Secrets }

Plaintext [base64 encoded → CH!2@4+a ↓ decode plaintext]

[encoding | decoding]

[encrypting | decrypting.] developer

'{cipher}FKSAJDFGYOS8F7GLHAKERGFHLSAJ' → encryptio

decrypted → Plaintext

password

K8S — Secret. kms spn clou

Java src main test Pom.xml Dor } build. mv.

df (image) (.JKS) → dev (dB_URL) (URL) dR, test, stg, prod

Env (cm) data. → Pod Runtime (?)

-Dspring.prof = dev, = pr.

cm | Secret → volume, as volumes

Key1 : siva, Key2, Key3 → env Pod ↳ env K = V K = V

files → volumes (Key1) → siva, Key2, Key3

configmap name: siva cou: K8S Key3 → Pod as Env va., pod as volume

env

cm, Sec → envFrom → data (Pod) Restart.

cm, Sec — Volume — pod → (mount) /name data

cm secret → Application devops Pod ✓ yes 100% d 100% R, K8S cluster yes No, SRE ↓

shaun crash. svc 100% Scheduling → (optional) volume → 100% cm/sec — 100% (Env)

DB - PASSword → cipher

si →

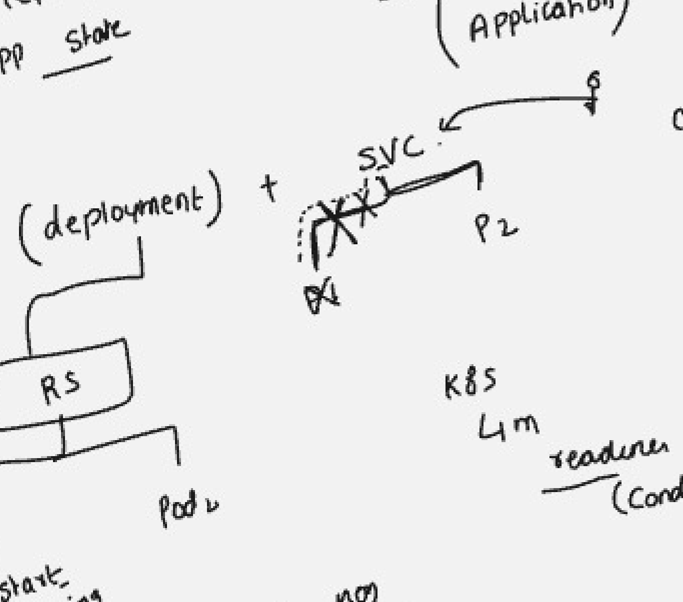
App depen { liveness Probe, Readiness Probe — 1000% → app.yml m1 m2 config-server Eureka

microser K8S- deploy liveness (probe) & readiness probe

(cont) x APP x

we need to let K8S APP state

Pod (cont Application) container



08:47 (1.2)

K8S 4m readiness (Condition) ✓ Pod ready

App (not healthy) App (not ready) } dev

spring → (actuator)

APP → Jdbc 34.10.10.4

{nginx (healthy) x, cart po healthy my health x}

APP, AP, AP, Jdbc clore; 5m 100 Active conne. mysql 50 inact

cart slave (APP cart) / monitor 200 x (Key)

dB 34.10.10.7

100 AppL yaml 300 { 100 Line 100 x Package helm

Pod → cart. 10gs → 10 gb Pvc ↳ Resource sto (cha)

Pod contai (cpu memory) } Resources namespace

memory → Bytes Requests

cpu → cpu unit

cart → Request. 512 MB. ↓ 1960

Limits

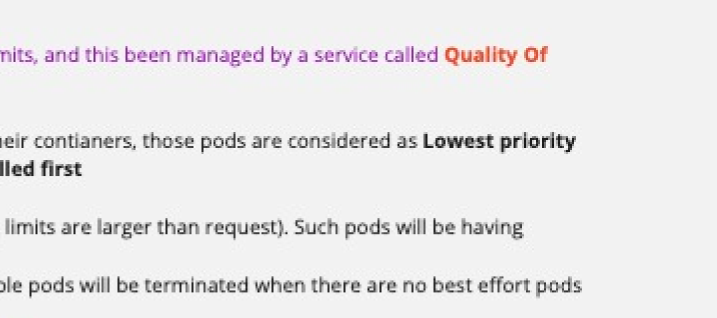
Resource Request:

- This will allow you to define the amount of resoureces (CPU,Memory) you expect a container to use.
- The k8s scheduler will use resourse requests to avoid scheduling the pods on the nodes that do not have enough available resources.

Resource Limits:

- this will provide a way for you to limit the amount of resources your container can use.
- The container run time, is responsible for enforcing the limits..

shared cluster



1Gi →

64mi → 64 mebibyte 12cpu, 64gb (1/4)

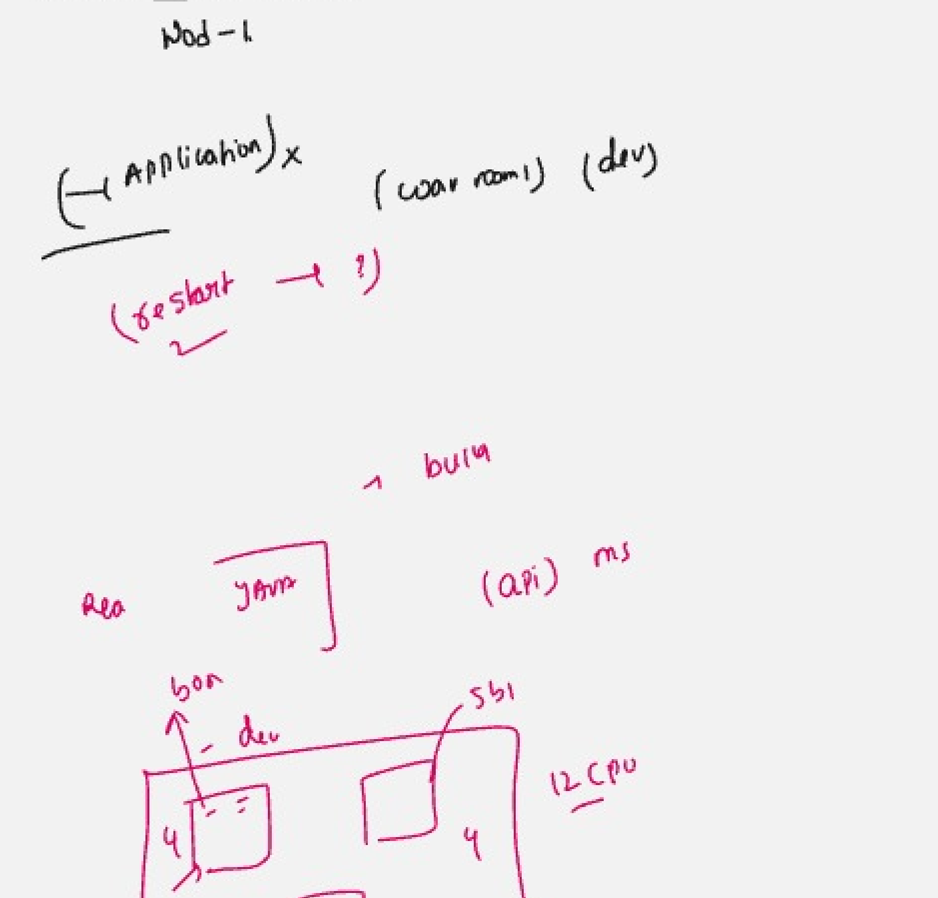
250m — cores ↓ 250 milli cpu ↳ 0.25 cpu core

(1gb - 2gb)

terminate → cluster →

Your apps will get termincated depending on the requests and limits, and this been managed by a service called **Quality Of Service (QOS)**

- Best Effort
  - pods that doesnot specify **requests and limits** into their contianers, those pods are considered as **Lowest priority and these are the one that are most likely to be killed first**
- Burstable:
  - Pods that has requests and limits defined(obvisously, limits are larger than request). Such pods will be having minimal resource guarently.
  - When the node is having very pressure , these burstable pods will be terminated when there are no best effort pods
- Guaranteed
  - the pods that have equal amount of resources and limits .
  - these type of pods are hgihest priority pods and guarented pods not be killed before the above 2

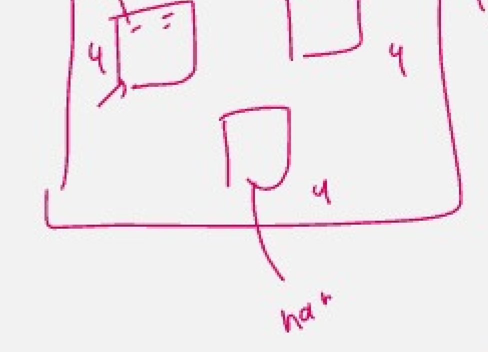


(gmha) (→ Application) x (war room) (dev)

(restart → ?)

→ build

Rea yaml (api) ms



JAVA — React → spr } (war)

React m1 m2 m3 YAVA