Chapter 4:
Kubernetes Performance and Best Practices

Table of Content

Exposing Service and Load Balancing

Access Services – What's best for me ?

Internal access – it's easy

- curl http://$(kubectl get svc myservice --template='{{.spec.clusterIP}}'):8080/

External access – it's not easy

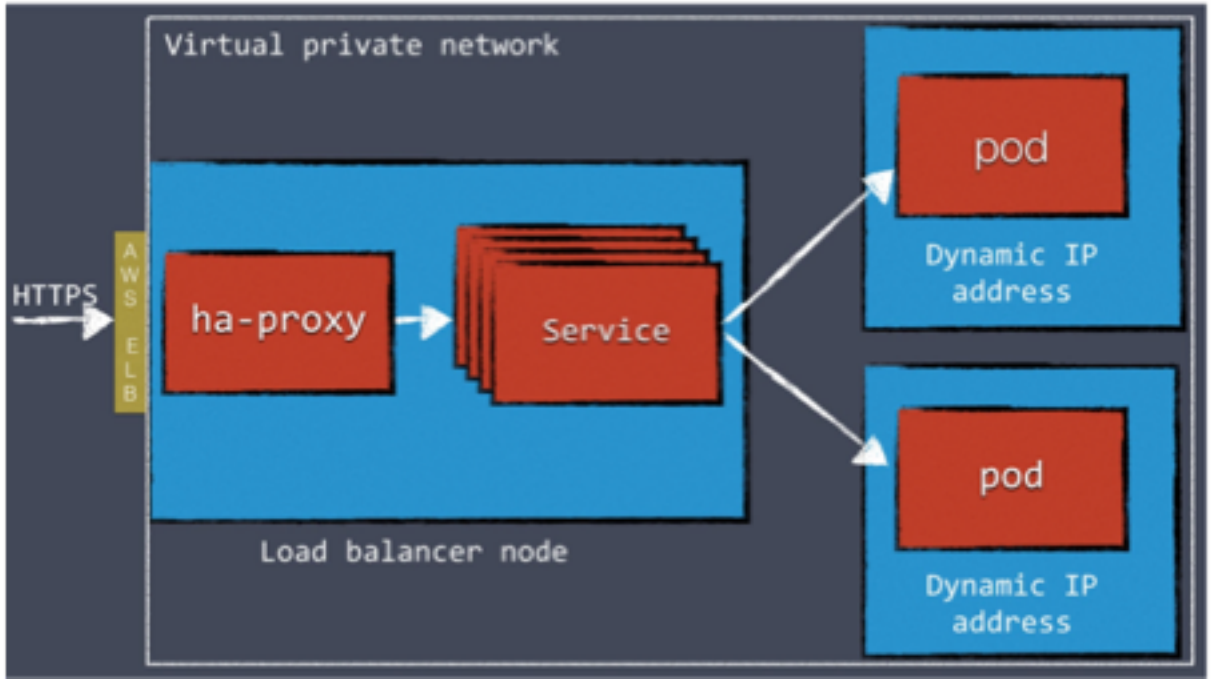
- **NodePort type** – services may clash due to same and limited number of ports
- **LoadBalancer type** –
  - Cloud: Use cloud provider load balancing, e.g. ELB in AWS. ELB instance is provisioned that proxies traffic for the Service inside the K8s cluster
  - Bare metal: DaemonSet running HAProxy -> watch services annotation through k8s API -> HAProxy config file re-written using pod IPs -> HAProxy exposes ports 80 and 443 and exposed to outside traffic as NodePorts

  Traffic – public load balancer – forward traffic to nodeports – haproxy does host path resolution through service annotation – traffic routed to pod ip

Two Step Load Balancing

Two step load balancing (in AWS)

- Configure a load balancer such as HAProxy or NGINX in front of the Kubernetes cluster.
- Use (for e.g.) AWS Elastic Load Balancer to route external web traffic to an internal HAProxy cluster.
- Can configure internal HAProxy with a “back end” for each Kubernetes service, which proxies traffic to individual pods.



OpenShift – External Load Balancer

OS SDN (software defined networking) – default overlay network

- LB as OS node, thus including LB in the SDN itself

Run OS node on the load balancer itself. So, this node gets its own bridge that the SDN configures automatically to provide access to the pods. Mark this node as un-schedulable

- LB as a pod

Even easier, simply run LB as a pod with the host port exposed. The pre-packaged HAProxy router in OS platform runs in this manner

- Establish a tunnel using Ramp node

F5 BIG-IP host cannot run OS platform. Instead of enabling BIG-IP to reach pods, an existing node in the cluster can be a ramp node and establish tunnel between F5 BIG-IP host and the ramp node.

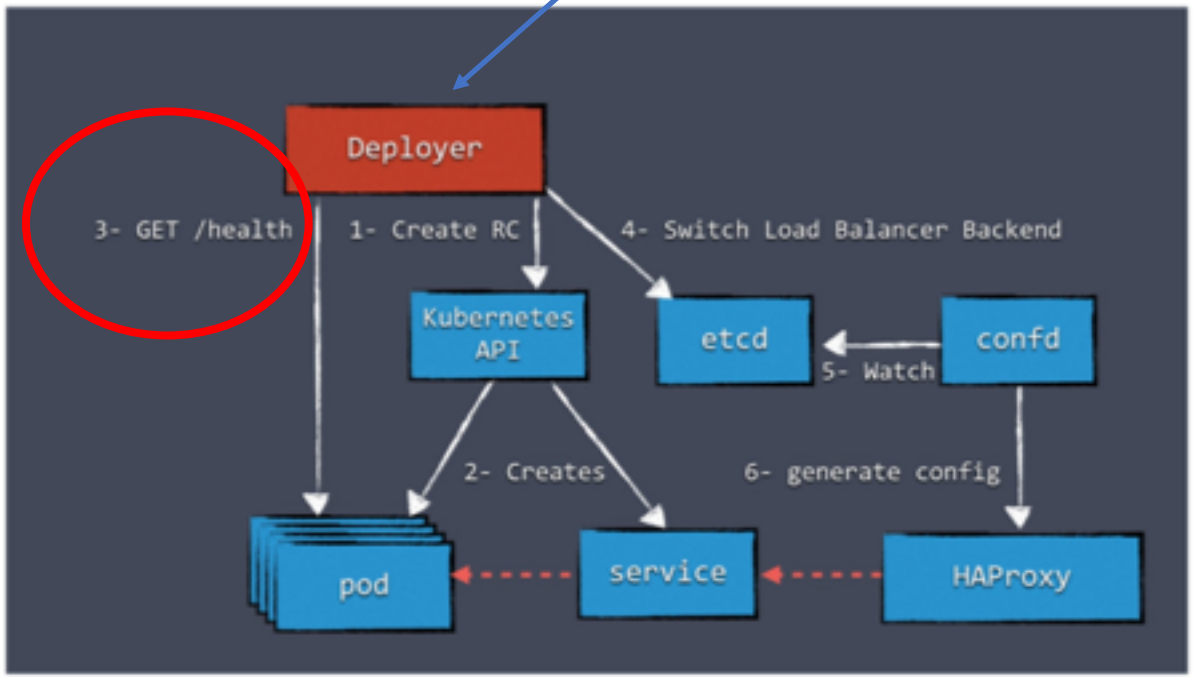
Deployment

Deployment – Blue/Green or Rolling Update

Deployer orchestrates the deployment.
Can be influenced by open source projects like
https://bitbucket.org/amdatulabs/amdatu-kubernetes-deployer/src/master/

Health check

- Important aspect - health checking on pods before switching the load balancer.
- Implement Readiness and Liveness probes on pods
- Log events
- Deploy with {**--record**}



Proper Deployment Parameters

```
spec:
  replicas: 8
  revisionHistoryLimit: 5
  minReadySeconds: 20
  type: RollingUpdate
  strategy:
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 2
```

Rolling Update parameters

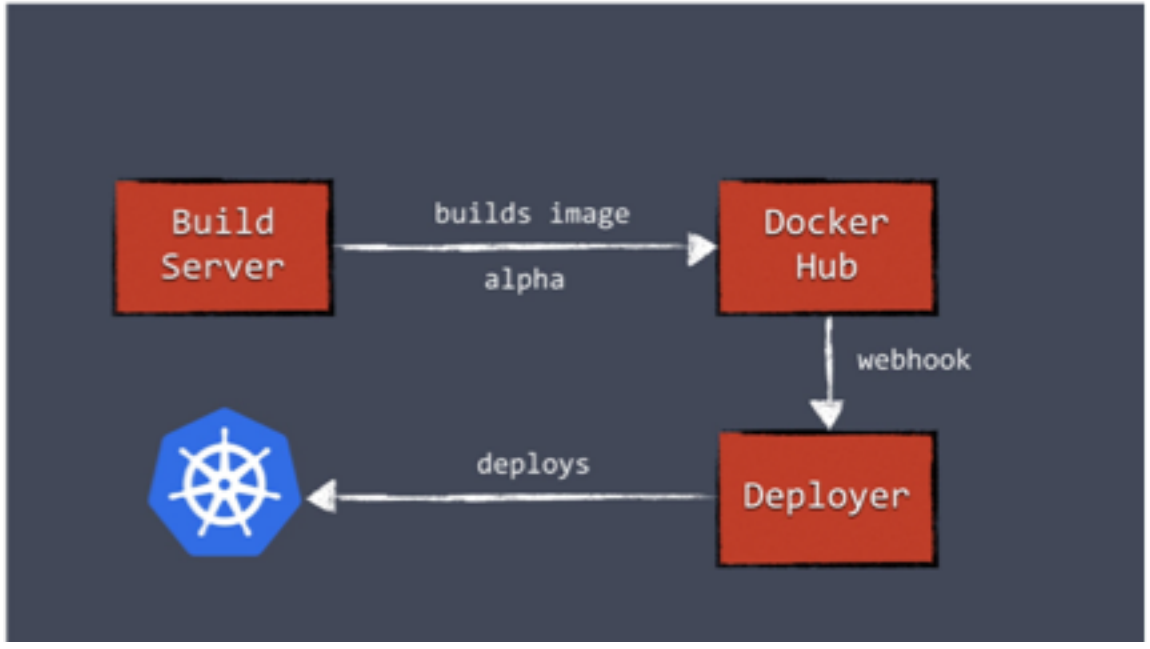
- At least **6** pods out of 8 (75%) are always running
- Prevents the expected number of replicas from being exceeded by more than **2** pods in order to avoid insufficient computing resources
- Pod is considered stable only after staying in the *Running* phase for at least **20** seconds so that late crashes, typically due to timeouts, can automatically pause the deployment

Default
25% availability – for huge production workload, app could become unresponsive.
25% Surge – when running tight on resources, this could bring you too close to limits

Making Deployments Automatic

Build Server

- Build server, after a successful build, push a new Docker image to a registry such as Docker Hub.
- Build server can invoke the Deployer to automatically deploy the new version to a test environment.



Graceful Pod ShutDown

Pod killing process

- Send SIGTERM
- Wait (this wait period is a property of the **podSpec** that can be overridden)
- Sends SIGKILL

If you have anything you need to do to ensure graceful shutdown then you need to implement a handler for SIGTERM, otherwise the process will get killed immediately and removed from etcd

Know your Resource Constraints

Resource Constraints – Extremely Important

- **Configure resource requests and CPU/memory limits on each pod**
- Can also control resource guarantees
- If you don’t constraint resources, containers may crash because they couldn't allocate enough memory

```
spec:
  resources:
    requests:
      cpu: 200m
      memory: 32Mi
    limits:
      cpu: 64m
      memory: 64Mi
```

Request – what a container is guaranteed to get

Limit – container never goes above this value

- CPU is defined in milli-cores and memory in bytes
- Memory limits – if a container passes its memory limit, it will be terminated
- CPU limits – if a CPU passes its CPU limits, k8s will throttle the container, but it won’t get killed

Resource Constraints

The world is not ideal !! People may define the resource and forget OR set a very high limit.

- Define ResourceQuota and LimitRange
- Lock down namespace using quotas – e.g. no quota on prod and very strict on dev namespace

```
spec:
  hard:
    requests.cpu: 500m
    requests.memory: 100Mib
    limits.cpu: 700m
    limits.memory: 500Mib
```

ResourceQuota
Aggregated for all containers

```
limits:
- default:
    cpu: 600m
    memory: 100Mib
min: …
max: …
```

LimitRange
Range for a single container

Properly Monitored and Logged K8s Cluster

Centralized Logging and Monitoring

- Monitoring and logging is crucial in this new dynamic environment
- Logging into a server to look a log file doesn't work anymore - large number of replicas and nodes
- Plan to ***build centralized logging and monitoring***

Options for Logging

Graylog, EFK – log management

Kafka – to collect and digest logs from containers

… lot others

Monitoring – Application Specific

Kubernetes does an excellent job of recovering when there's an error.

Kubernetes recovery works so well that containers may crash multiple times a day because of a memory leak, without anyone noticing it.

* Nice to have – **application-specific health checks and dashboards** that monitors nodes and pods

- Measure load, throughput, application errors, and other stats

Options for Monitoring

InfluxDB, Heapster, cAdvisor, Grafana

Data Stores

Data Store in K8s Volumes ??

With data, most of the times, reliability comes before performance

- Most data stores require precise config; auto discovery and configuration of nodes is not common, which is by design an important characteristic of Kubernetes
- A machine that runs a data store is often specifically tuned for that type of workload. Higher/lower IOPS

All these things don't match very well with the dynamic nature of Kubernetes deployments

Configuration Management

- *Loosely coupled*
  - Keep config and code (image) separated, so config can be persisted even when containers are restarted, scaled up, scheduled to new nodes, etc.
    - Use ConfigMaps, Secrets, and Environment variables for configuration storage
    - Prefer using CM and secrets than environment variables – no access control, these are global variables, and few process like Cron & Monit may scrub environment variables

Labels

A simple and powerful way to associate related entities

- Use plenty of descriptive labels
- Use labels to perform operations, node selection, deployments
- Use labels to monitor the cluster – Build label specific metrics – to identify performance of entities at a {TYPE} level
    - Metrics at Pod and container level might not be too useful, instead visualize for a group of pods using labels

Registries and Packages

- Use secured registries
  - Do not trust arbitrary base images
  - Signed & secured image
  - No {latest} tag
  - Have team specific base images, yet keep base image small
- Use Helm
  - Templates to define, install and upgrades k8s based application
  - Helps in streamlining CI/CD

- Docker
  - Non-root user
  - One process per container
  - Crash cleanly with an error message

Pod Scheduling and Others

- **There is a limit on number containers you can run per node**

Default is 110 per node, change this in kubelet setting

https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ (--max-pods)

- **Inotify errors**

Scheduling more pods may give inotify errors (filesystem monitoring watchers)

Change this in all host. Either use Infra as code tools or run a container as a Daemon set to change these limits in all host

e.g. https://gist.github.com/brendan-rius/5ac9ec3dd7e196222c8b8b356f8973d2

Pod Scheduling and Others

- **Set CPU and Memory Limits asap**

As discussed before

Scenario:- A containers using 200Mib memory but only 5% of a vCPU. Running xxx of these containers might end up scheduling lots in a node, resulting in OOM errors. K8s will eventually figure out, as OOM will evict pods and re-schedule, but this process may take time

- **Pod Isolation**

Critical pods should be isolated by node affinity and anti-affinity

Reliability and Agility At Scale

https://www.youtube.com/watch?v=9C6YeyyUUmI

https://www.nginx.com/blog/testing-performance-nginx-ingress-controller-kubernetes/

Follow 12 factor

https://12factor.net