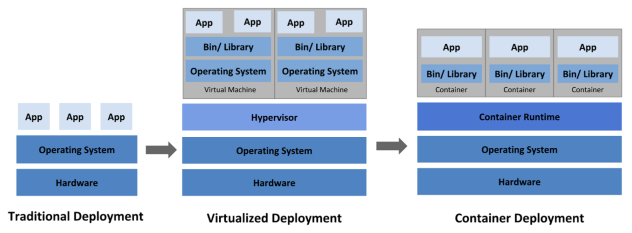
Kubernetes Cheat Sheet

by Piergiorgio Ladisa

Kubernetes Basic Definitions

Kubernetes (k8s) is a container orchestration tool, which aids in managing containerized applications across various deployment environments.



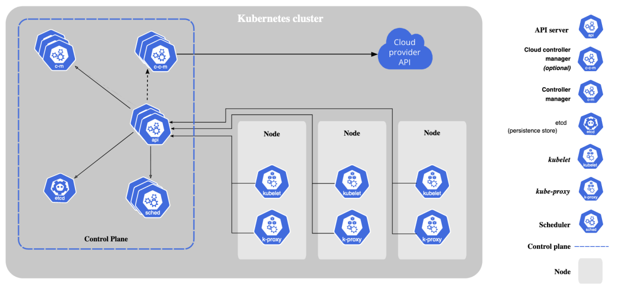
Node	A physical or virtual machine, i.e., a worker machine in k8s
Cluster	Set of worker machines (a.k.a. nodes)
Pod	Set of running container(s). It is in an abstraction over a container that facilitates interaction through k8s
Control plane	container orchestration layer that exposes the API and interfaces to manage containers.
Service	Way to expose an application running on pod(s) as a network service. Lifecycles of services and pods are separate.
ConfigMap	Component that contains the external configuration of the application
Secret	Store sensitive data as base64 strings, unencrypted by default. Pods can reference secrets via volume mounts or environment variables.
Volume	Directory containing data, accessible to the containers in a Pod.
Namespaces	Abstraction used by Kubernetes to support isolation of groups of resources within a single cluster
Deployment	API object that manages a replicated application, typically by running Pods with no local state
ReplicaSet	Workload resource that (aims to) maintain a set of replica Pods running at any given time
Job	Supervisor for pods that runs batch processes

References

https://kubernetes.io/docs/home/
https://madhuakula.com/content/attacking-and-auditing-docker-containers-and-kubernetes-clusters/kubernetes-101/technical-terms.html

Architecture

A k8s deployment results in a *cluster*, which is a collection of *nodes*. These nodes run containerized applications. Each cluster contains at least one worker node. The worker nodes host *Pods*. The *control plane* oversees the worker nodes and the Pods within the cluster.



Control Plane Components

- `kube-apiserver`: control plane's front end, serving as a gateway for cluster requests and an authentication checkpoint. It exposes the k8s API for client interaction and directs requests to the right nodes
- `kube-scheduler`: watches for newly created pods and decides which node should host the new pod. Once the Scheduler has made this decision, the `kubelet` takes over and actually schedules the pod
- `kube-controller-manager`: runs controller processes to detect changes in the cluster state (e.g., if a pod died) and attempts to restore the cluster to its previous state
- `etcd`: consistent and highly-available key-value store used as k8s backing store for all cluster data (i.e., only data pertaining master and worker nodes). Changes to the cluster and its state get stored in this key-value store
- `cloud-controller-manager` (optional): if k8s is not run on premises, this component embeds cloud-specific control logic to link the cluster into your cloud provider's API.

Node Components

- `kubelet`: process using PodSpecs to ensure containers are running and healthy. It interacts with both the container and node, configures pods, initiates them with a container, and communicates via Services.
- `kube-proxy`: network proxy running on each node that maintains rules, enabling pod communication within and beyond the cluster.
- `Container Runtime`: component responsible for managing the execution and lifecycle of containers within the Kubernetes environment. It supports, e.g., `containerd`, `docker`

Kubectl

`kubectl` is a CLI tool used for interacting with a k8s cluster's control plane by sending requests to the API Server.

For configuration, `kubectl` looks for a file named `config` in the `$HOME/.kube` directory. You can specify other kubeconfig files by setting the `KUBECONFIG` environment variable or by setting the `--kubeconfig` flag.

Kubectl context and configuration

Show merged kubeconfig settings and raw certificate data and exposed secrets:

```
$ kubectl config view
$ kubectl config view --raw
```

Show list of contexts:

```
$ kubectl config get-contexts
$ kubectl config current-context
```

Set the default context to `example`:

```
$ kubectl config use-context example
```

Kubectl - Create Commands

Create resource(s) from file(s), directory, url:

```
$ kubectl apply -f ./my-manifest.yaml
$ kubectl apply -f ./my1.yaml -f ./my2.yaml
$ kubectl apply -f ./dir
$ kubectl apply -f https://example.com/manifest.
    yaml
```

Create deployment or job manually:

```
$ kubectl create deployment [name] --image=IMG_NAME
$ kubectl create job hello --image=IMG_NAME -- [
    command]
```

Kubectl - Read Commands

Get commands with basic output:

```
$ kubectl get all | nodes | pods | services |
    replicaset | configmap | secret |
    replicationcontrollers | daemonsets
```

List a particular deployment:

```
$ kubectl get deployment my-dep
```

Read Certificates & Secrets

Retrieve the value of a key with dots, e.g. 'ca.crt':

```
$ kubectl get configmap myconfig -o jsonpath='{.
    data.ca\.crt}'
```

Output decoded secrets without external tools:

```
$ kubectl get secret my-secret -o go-template='{{
    range $k,$v := .data}}{{"### "}}{{$k}}{{"\n
    "}}{{$v|base64decode}}{{"\n\n"}}{{end}}'
```

List all Secrets currently in use by a pod:

```
$ kubectl get pods -o json | jq '.items[].spec.
    containers[].env[]?.valueFrom.secretKeyRef.name
    ' | grep -v null | sort | uniq
```

Kubectl - Update & Patch Resources

Check the history of deployments including the revision:

```
$ kubectl rollout history deployment/frontend
```

Rollback to the previous deployment:

```
$ kubectl rollout undo deployment/frontend
```

Rollback to a specific revision:

```
$ kubectl rollout undo deployment/frontend --to-
    revision=n
```

Add a Label:

```
$ kubectl label pods my-pod new-label=awesome
```

Remove a label:

```
$ kubectl label pods my-pod new-label-
```

Edit deployment:

```
$ kubectl edit deployment [name]
```

Use an alternative editor:

```
$ KUBE_EDITOR="nano" kubectl edit [resource name]
```

Kubectl - Delete Resources

Delete a deployment:

```
$ kubectl delete deployment [name]
```

Delete a pod using the type and name specified in the specified config file:

```
$ kubectl delete -f config-file.yaml
```

Delete a pod with no grace period:

```
$ kubectl delete pod POD_NAME --now
```

Delete pods and services with label `name=myLabel`:

```
$ kubectl delete pods,services -l name=myLabel
kubectl -n my-ns delete pod,svc --all
```

Delete all pods and services in a namespace:

```
$ kubectl -n [namespace name] delete pod,svc --all
```

Kubectl - Interact with Nodes and Clusters

Mark node as unschedulable and schedulable:

```
$ kubectl cordon [node name]
$ kubectl uncordon [node name]
```

Show metrics for all nodes or given node:

```
$ kubectl top node
$ kubectl top node [node name]
```

Display addresses of the master and services:

```
$ kubectl cluster-info
```

Dump current cluster state to stdout or to file:

```
$ kubectl cluster-info dump
$ kubectl cluster-info dump --output-directory=[
    path]
```

Kubectl - Authorizations

Prints all allowed actions:

```
$ kubectl auth can-i --list
```

Check to see if I can do everything in my current namespace:

```
$ kubectl auth can-i '*' '*'
```

Kubectl - Pod Inspection & Troubleshooting

List all pods in all namespaces:

```
$ kubectl get pods --all-namespaces
```

List detailed info about all pods in the current namespace:

```
$ kubectl get pods -o wide
```

Get a pod's YAML:

```
$ kubectl get pod my-pod -o yaml
```

Get all running pods:

```
$ kubectl get pods --field-selector=status.phase=
    Running
```

Log to console and stdout relatively:

```
$ kubectl logs POD_NAME
$ kubectl logs -f my-pod
```

Run pod as interactive shell:

```
$ kubectl run -i --tty POD_NAME --image=IMG_NAME --
     sh
```

Attach to running container:

```
$ kubectl attach POD_NAME -i
```

Listen on port n on the local machine and forward to port k on the pod:

```
$ kubectl port-forward POD_NAME n:k
```

Run command in a pod (1 container and multi-container cases):

```
$ kubectl exec POD_NAME -- [command]
$ kubectl exec POD_NAME -c [container] -- [command]
```

Get interactive terminal:

```
$ kubectl exec -it POD_NAME -- bin/bash
```

Create an interactive debugging session within existing pod or node:

```
$ kubectl debug POD_NAME -it --image=IMG_NAME
$ kubectl debug [node name] -it --image=IMG_NAME
```

Describe pod (e.g., state, volumes):

```
$ kubectl describe pod POD_NAME
```

Show metrics for all pods or a specific pod in the default namespace:

```
$ kubectl top pod
$ kubectl top pod POD_NAME --sort-by=cpu
```

Show metrics for a given pod and its containers:

```
$ kubectl top pod POD_NAME --containers
```