



KUBERNETES & DEPLOYMENT

Samir Zalmay

Contents

What is Kubernetes?	2
Kubernetes Basic/Core Components:	2
Kubernetes Architecture	4
Node processes	4
Managing processes	4
Minikube and Kubectl	5
Minikube	5
Kubectl	5
Install Minikube & Kubectl & Main commands	5
YAML Configuration File in Kubernetes	7
Kubernetes namespaces	9
Ingress	10
Helm	10
Docker registry	11
Sources	12

What is Kubernetes?

Official definition:

Open source container orchestration tool developed by Google. It help you manage containerized applications in different environments, like physical, virtual and cloud.

Instead of scripts and self-made tools, container orchestration tools are developed, which combine a lot of microservices together.

Advantages:

- High availability or no downtime
- Scalability or high performance
- Disaster recovery – backup and restore

Kubernetes Basic/Core Components:

Node:

A simple server or virtual machine.

Pod:

They are the smallest units of K8s. It creates this running environment on a layer on top of the container. It abstracts away from the container technologies. This way you don't have to work with Docker, but only with the Kubernetes layer. There is usually 1 application container per Pod. Each pod gets its own IP address (not the container). It is an internal IP address. When a pod dies another pod will be used with a new IP address (re-creation).

Service:

Service is a static/permanent IP address which can be attached to each pod. The lifecycles of the service and the pod are not connected. So even if the pod dies the service and its IP address will stay and it doesn't have to be changed.

External/Internal service:

You want your application to be accessible through the browser. For this an external service is created. It is a service which opens the communication from external sources. Because you don't want the IP address of a database open for a public request you don't use an external service, but an internal one. This is a type of service which you specify when creating one.

Ingress:

Regulates the forwarding to the services.

ConfigMap:

Regulates external configuration of your application. It would usually contain configuration data like urls of databases or other services. In k8s you connect it to the pod so the pod gets the data of the configmap. Putting passwords and usernames in the ConfigMap is insecure even though it could be an external configuration. Therefore we use Secret.

Secret:

Secret is just like ConfigMap, but the difference is that it is used to store secret data, like credentials. It's not stored in plain text, but in base 64 encoded format. (Certificates are also possible to put in Secrets).

Secret and ConfigMap can be used as environment variables or as properties file.

Volumes:

Databases have data. If the pod or container gets restarted the data will be lost. The data needs to be persisted long term. Therefore we can use Volumes. It attaches a physical storage to on a hard drive to your pod. The storage can be either on a local machine, meaning on the same server node where the pod is running. Or in a remote storage outside of the k8s cluster, like Cloud Storage or an own premise storage. When a Pod or Container gets restarted the data will be persisted.

Deployment and stateful set:

What happens if the application pod dies, crashes or when the pod has to restart when a container is rebuild. Basically there will be a downtime where a user can't reach the application. This is a bad thing when it happens in production. For this we don't rely on just one pod. We will replicate everything on multiple servers.

Service is a load balancer. It will catch the request and forwards it to whichever pod is running. Also it has the permanent IP address for the pods and it's replicas. This way you don't have to adjust the endpoint when a pod dies.

To specify the replicas of a pod we use the component **Deployment**. It is a blueprint for the pods. You can scale up or scale down the amount of replicas you want to create. So Deployment is another abstraction on top of the pods, which makes it more convenient to interact with the pods. So in practice you will mostly work with deployments and not with pods.

For databases deployments can not be used to replicate their pods. The reason is that databases has state, which means its data. This means that if there are multiple pods of the database they all need to have access to the same shared data storage. So that means it would need some functionality, which manages which pods are currently writing to the storage or reading in order to avoid data inconsistencies. *This mechanism* in addition to addition to the *replication feature* is offered by another k8s component, **Stateful Set**. So this is meant specifically for databases.

PS. Deploying database applications using stateful set can be difficult. That's why it is common practice to host database applications outside of the Kubernetes cluster.

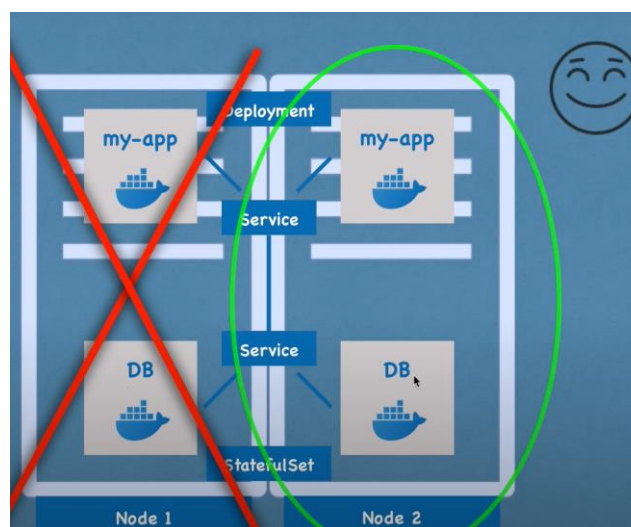


Figure 1: Deployment and Stateful Set, Using replica nodes to mitigate downtime of the application.

Kubernetes Architecture

Node processes

Each node has multiple pods running on it. Three processes must be installed on every Node. The worker nodes do the actual work.

- First process: **Container runtime**.
- Second process: **Kubelet**.
Interacts with both the container and the node. It starts the pod with a container inside.
- Third process: **Kube proxy**.
Responsible for forwarding requests from services to pods. It has intelligent forwarding logic inside that makes sure that the communication works in a performant way with low overhead. Example: If an application is making a request to a database, the request won't be forwarded to just any database replica, but it will be forwarded to the database which is in the same node.

Managing processes

How to interact with the cluster? How to schedule pod? How to monitor? How to reschedule or restart pods? How does a new server join the cluster with a new node?

All these questions above will be worked out with Master nodes. The master nodes execute these managing processes. There are four processes, which are run on every master node.

- First process: **Api server**
Client interacts with the API server. It is the cluster gateway and acts as a gatekeeper for authentication. It is good security wise, because there is only 1 entry point into the cluster.
- Second process: **Scheduler**
If you send an API server a request to make a new pod the scheduler will start the application pod to work on one of the nodes. It has an intelligent way of deciding on which specific worker node the next pod will be scheduled. It looks at the available resources of the nodes and it will schedule the pod in the pod which has the most available resources left. The scheduler just decides on which Node a new Pod should be scheduled. The process which actually starts the Pod with a Container is the Kubelet. Kubelet gets the request from the scheduler and executes the request on the node.
- Third process: **Controller manager**
If a Pod dies on a Node there must be a way to detect it and reschedule those Pods as soon as possible. The Controller manager detects state changes, such as crashing Pods. It will try to recover the cluster state as soon as possible.
- Fourth process: **etcd**
Key value store of a cluster state. It is the brain of the cluster. Every change of the cluster gets saved in the etcd store. For example when a new Pod gets scheduled or a Pod dies. It will provide data to the **Scheduler**, **Controller manager** and **API server**. The data of databases won't be stored in etcd. It is only used for the master processes.

Master nodes need less resources, because they carry less load than worker nodes.

Minikube and Kubectl

Minikube

Minikube is one Node cluster where the master processes and the worker processes both run on one Node. This makes it able to run Containers or the Pods with Containers on this node and the way it's going to run on your laptop is through a virtual box. So Minikube creates a virtual box on a laptop and the node will run in that box. So Minikube is a 1 Node K8s cluster. It can be used for testing k8s on a local setup.

Kubectl

It interacts with the cluster. It is a command line tool for k8s cluster. Most powerful of the three clients (UI, API, CLI). Kubectl communicates with the Api server to create components, delete components etc. The Worker processes will actually make this happen. Kubectl is not just for Minikube cluster, but also for Cloud clusters or Hybrid clusters etc.

Minikube needs a virtualization, because it will run in a virtual box (Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox or VMWare).

Install Minikube & Kubectl & Main commands

- <https://minikube.sigs.k8s.io/docs/start/>
- <https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/>

Kubectl get po -A (Interact with your cluster)

Minikube dashboard (Minikube bundles the Kubernetes dashboard)

Minikube start (start-up cluster)

Minikube status (see status of the cluster)

Main Kubectl commands

The infographic is titled "Basic kubectl commands" and features the Kubernetes logo. It is divided into several sections:

- Create and debug Pods in a minikube cluster:** Accompanied by a small icon of a ship's wheel.
- CRUD commands:**
 - Create deployment: `kubectl create deployment [name]`
 - Edit deployment: `kubectl edit deployment [name]`
 - Delete deployment: `kubectl delete deployment [name]`
- Status of different K8s components:**
 - Command: `kubectl get nodes | pod | services | replicaset | deployment`
- Debugging pods:**
 - Log to console: `kubectl logs [pod name]`
 - Get Interactive Terminal: `kubectl exec -it [pod name] -- bin/bash`

Kubectl get nodes (list of all nodes)

Kubectl get pod (list of all pods)

Kubectl get services (list of all services)

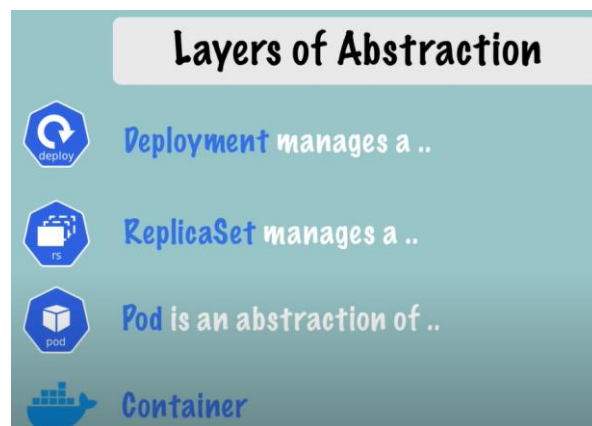
In practice pods won't be created manually. That's why we create deployments which creates our pods.

Kubectrl create deployment nginx-deplo --image=nginx (Creating deployment of nginx)

Kubectrl get deployment (get deployments list)

Kubectrl get replicaset (list of replicasets)

Replicaset manages the replica of the pods. Replicasets don't have to be created. We only work with deployments. Everything below the deployment is managed by Kubernetes.



Kubectrl edit deployment 'deployment name' (Opens the deployment file and can be edited)

Kubectrl logs 'deployment name' (deployment name can be found by 'kubectrl get deployment')

kubectrl exec -it 'pod name' -- bin/bash (useful for debugging or testing and trying something)

Kubectrl delete 'deployment name' (deleting deployment, its pods and its replicaset)

Kubectrl apply -f nginx-deployment.yaml (creating and update yaml file for deployment)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
        - containerPort: 80
```

Figure 2: Basic configuration.yaml file for deployments

Debugging pods	
Log to console	<code>kubectl logs [pod name]</code>
Get Interactive Terminal	<code>kubectl exec -it [pod name] -- bin/bash</code>
Get info about pod	<code>kubectl describe pod [pod name]</code>
Use configuration file for CRUD	
Apply a configuration file	<code>kubectl apply -f [file name]</code>
Delete with configuration file	<code>kubectl delete -f [file name]</code>

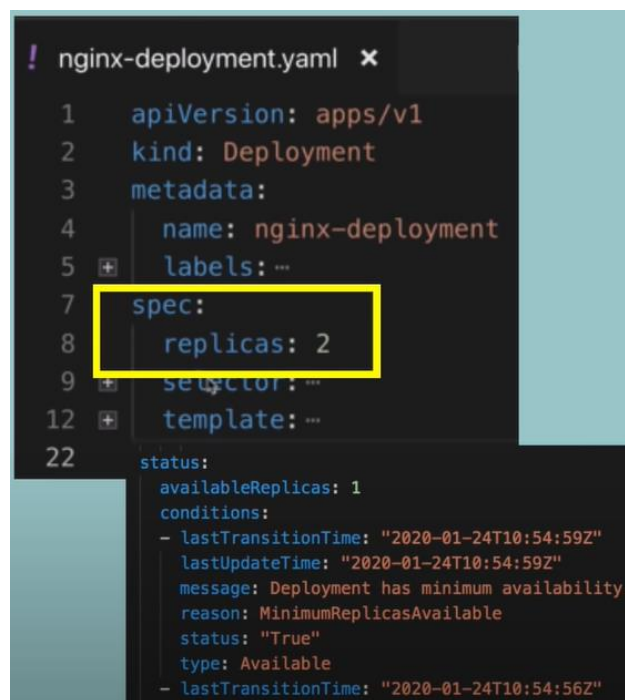
Figure 3: Main commands overview.

YAML Configuration File in Kubernetes

Every configuration file (YAML file) has 3 parts:

- Metadata
- Specifications
- Status (automatically generated and edited by Kubernetes. Will be updated continuously by kubernetes)

Information comes from the etcd. It holds the current status of any kubernetes component.



```

! nginx-deployment.yaml x
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx-deployment
5    labels: ...
7  spec:
8    replicas: 2
9    selector: ...
12  template: ...
22  status:
    availableReplicas: 1
    conditions:
    - lastTransitionTime: "2020-01-24T10:54:59Z"
      lastUpdateTime: "2020-01-24T10:54:59Z"
      message: Deployment has minimum availability.
      reason: MinimumReplicasAvailable
      status: "True"
      type: Available
    - lastTransitionTime: "2020-01-24T10:54:56Z"

```

Figure 4: Example 3 parts of a configuration file.

YAML file needs to have right indentations. Use online YAML validator to fix the indentations in the file.

Template

Template is a configuration file inside the configuration file. It has its own metadata and spec section. It applies to the pods. This will be the blueprint for the pods. (port, image and name will be described here)

Labels & Selectors

key-value pair. Pods get the label through the template blueprint. This label will be matched by the selector. It will create a connection between everything which has the same label. (i.e. connecting services to deployments)

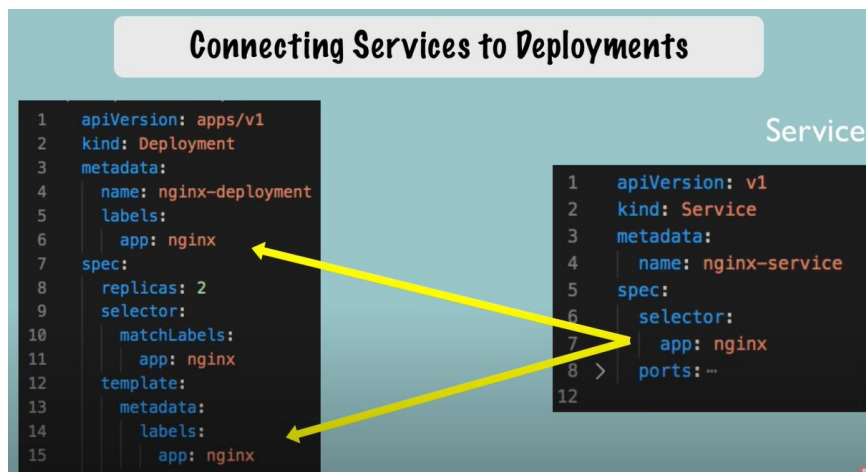


Figure 5: Example connection between two components (Service and Deployment) with labels & selectors.

Ports in Service and Pod

Service needs to know which port the Pod uses. For this the targetPort will be used. The targetPort should match the containerPort.

More commands

Kubectl describe service 'service name' (Selector, targetPort and endpoints can be found here)

Kubectl get pod -o wide (more information of the pods, like IP address)

Kubectl get deployment nginx-deployment -o yaml (gets the yaml output from the yaml file)

```
[~]$ echo -n 'username' | base64
dXNlcm5hbWU=
[~]$
```

Figure 6: Conversion of a string to base64 code. Used for Username and Password values for Secrets.

```
spec:
  containers:
  - name: mongodb
    image: mongo
    ports:
    - containerPort: 27017
    env:
    - name: MONGO_INITDB_ROOT_USERNAME
      valueFrom:
        secretKeyRef:
          name: mongodb-secret
          key: mongo-root-username
    - name: MONGO_INITDB_ROOT_PASSWORD
      value:
```

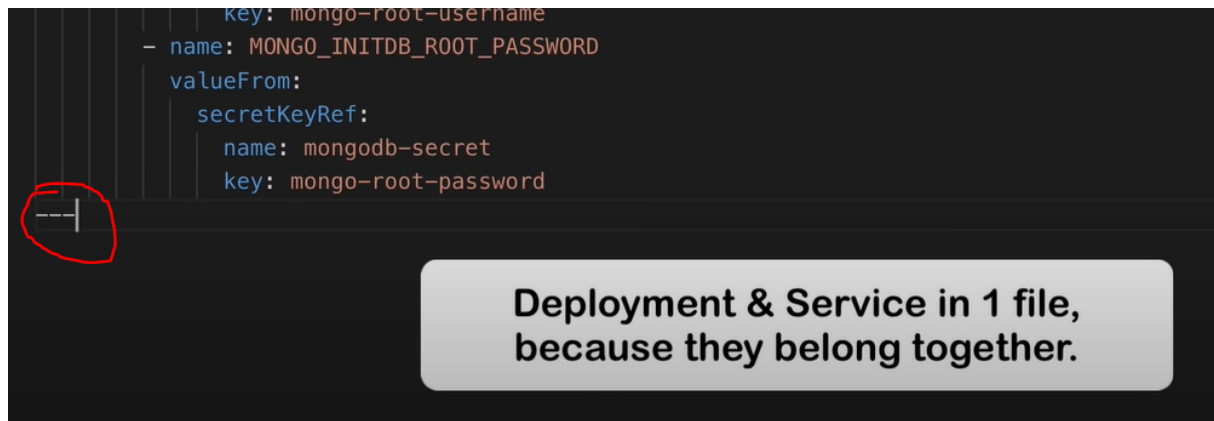


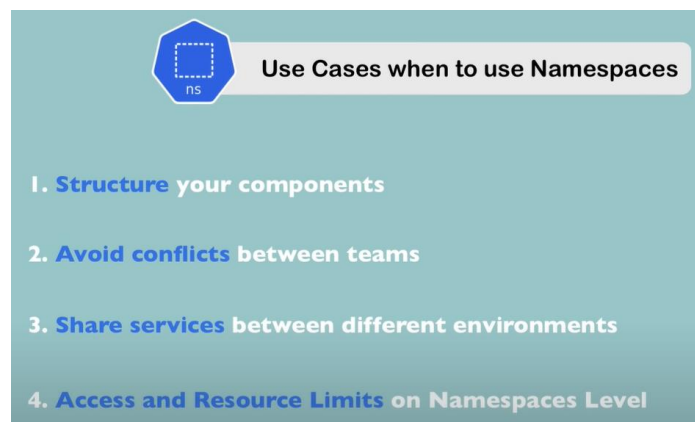
Figure 7: --- is to tell the file that we want multiple documents in one file (Deployment and Service).

Kubernetes namespaces

Organise resources in namespaces. Virtual cluster inside a cluster.

Why use namespaces:

- Default namespaces could be filled real fast. That's why we logically group resources in namespaces.
 - When working with multiple teams on a project.
 - Resource sharing: Staging and Development.
 - Access and resource limits on namespaces.
- Limit CPU, RAM, Storage per namespace such that each team has enough.



You can't access most resources from another namespace. Components which can't be created within a namespace, these are used globally inside the cluster and they can't be isolated.

When components are created without a specified namespace, a default namespace will be used. Can be done by cmd or with configuration files. Advantage: Better documented if it's in the configuration file.

Ingress

External Service vs Ingress

External services are used for testcases. Ingress is used for deployment. (IP address will have the application name in it).

Ingress controller

- Evaluates all the rules
- Manages redirections
- Entry point to the cluster.
- Many third party implementations
- K8s Nginx Ingress Controller

Ingress controller in Minikube

Install ingress controller in Minikube. Automatically starts the k8s nginx implementation of Ingress Controller. Ingress controller will be configured with just a single command.

Helm

Helm

Package manager for Kubernetes. To package YAML files and distribute them in public and private repositories.

Helm Charts

Bundle of YAML files, which are created by other people. Helm Charts can be created with Helm. They can be pushed to Helm repositories. Existing Helm charts can be downloaded and used.

Docker registry

Preparing images to be pushed to DockerHub

Docker tag 'image name': 'image tag' 'dockerhub account'/'dockerhub repository': 'tag'

Push image to DockerHub

docker push tag 'dockerhub account'/'dockerhub repository': 'tag'

Sources

<https://andrewlock.net/deploying-asp-net-core-applications-to-kubernetes-part-1-an-introduction-to-kubernetes/>

https://www.youtube.com/watch?v=X48VuDVv0do&ab_channel=TechWorldwithNana

<https://ocelot.readthedocs.io/en/latest/features/kubernetes.html> (Ocelot + Kubernetes)