# Kubretes Tutorial for Beginners

[Youtube - TechWorld with Nana](https://www.youtube.com/watch?v=X48VuDVv0do)

## Intro to K8s

### What’s kubernetes?

#### Official Definition

* *Open source container orchestration framework*.
* Originally it was developed by Google.
* Helps you manage containerized applications in different deployment environments.
  + Physical machines
  + Virtual machines
  + Cloud environments

#### Problem-Solution case study

The need for an orchestration tool.

* Trend from *Monolith* to *Microservices*
* Increase usage of *containers*: managing houdreds of container through scrips becomes impossible.
* Demand for a *proper way* of *managing* those hundreds of containers.

What features do orchestration tools offer?

* *High availability* or no downtime
* *Scalability* or high performance
* *Desaster recovery* - backup and restore

### Main K8s Components

* **Pod:**
  + A smallest unit of K8s.
  + Abstraction over a container.
  + It creates a running environment. It’s a layer in the top of the container. Kubernetes wants to abstract away the container runtime (container technologies) because it can be replaced and also because the developer has to directly work with different container technologies use in kubernetes. The developer interacts with the *kubernetes layer* only.
  + Usually one main application per pod or some side service which has to run inside of that pod.
  + Kubernetes offers out of the box a virtual network. Each *pod* gets its own IP address. When a pod dies, a new one will be created with a new IP address.
* **Service**:
  + permanent IP address
  + lifecycle of the service and the pod are not connected.
  + **external service**: opens the communication from external sources.
  + **internal service**: service tha can be reached from inside. The type of the service should be specified when creating one.
  + **Ingress**: assures domain names for external services.
  + A service has two functionalities:
    - permanent IP address
    - load balancer: the server will catch the request and will forwarded to a pod wich is not busy.
  + **Blueprint**: If more pods are needed there will not be created new pods instead there will be created a blueprint in which will be specified how many instances want you to run. The bleuprint is called **deployment**. In practice you won’t be creating pods. In practice you will be creating deployments in which you can specify how many replicas do you want. You can also scale up or down the number of replicas.
  + **Deployment**: The pod is a layer of an abstraction on the top of the containers.The deployment is another abstraction on the top of pods.
  + **StatefulSet**: We can’t replicate databases using a deployment. The reason for that is that the database has state. If we have clones of replicas of the database. The will need to access the same shared database storage and there you would need some mechanism that manages which pods are currently writing to that storage or which pods are reading from it in order to avoid inconsistencies. That mechanism in addition to replicating feature is offered by another kubernetes komponent called StatefulSet. This component is meant specifically for applications like databases. So *MySql*, *mongoDb*, *elasticSearch* or any other statefull application should be created using statefulSets and not deployments. StatefulSets will take care of replicating the pods and scaling them up or down, but making sure that database read and writes are synchronized so no database inconsistancies are offered.
* **Config Map**: external configuration of your application.
* **Secret**: It is like the config map. The difference is that it is used to store secret data. base 64 encoded format. It would contain things like credentials, certificates, things that you don’t want other people have access to.
* **Volume**: It’s a component used for store persistent data. The hardware can be on the local machine (same server) on which the pod is running or could be on a remote server (outside the kubernetes cluster). Could be a cloud storage or your own storage. Regardles if the storage is inside or outside of a cluster and the pod is restarted the data will be persisted. The cluster does not manage any data persistance.

### K8s Architecture

* **Node processes**: Worker machine in Kubernetes cluster or nodes. Each node will have multiple pods running on that node. The way that kubernetes does it is that 3 processes must be installed on each node which used to schedule and manage those pods.
  + **container runtime**: for example docker but it could be other technologies.
  + **kubelet**: schedules the pods to run on the container runtime.
  + **kube proxy**: forwards requests from services to pods.
* **Worker nodes**: do the work.
* **Master nodes**: So how to interact with this cluster? How to schedule pod? What process monitors if it dies? What process reschedules it? How does join a new node? All this managing processes are done by master nodes. Master nodes have completely different processes:
  + **Api Server**: When deploying a new application in a kubernetes cluster. It is like a cluster gateway and also acts as a gatekeeper for authentication.
  + **Scheduler**: starts the pod on one of the worker nodes. Scheduler decides on which node a pod will be scheduled. The process that actually starts the pods is the kubelet.
  + **Controller Manager**: Detects when nodes die and reschedules them as soon as possible.
  + **etcd**: It’s a key-value store of a cluster state. Cluster changes get stored in the key value store.
  + In practice kubernetes kluster is made up from multiple Master processes.
* Add new master / node server:
  + get new bare server
  + install master / worker node processes
  + add it to the cluster

### Minikube and Kubectl - local setup

* **Minicube**: Is a one node cluster where the Master and Node processes both run on one machine. This node will have a docker container preinstalled. This will run on the local computer through a VirtualBox or some other hipervisor. Creates virtualBox on the local machine and runs the nodes in that virtual box. It is used for testing purposes.
* **Kubectl**: Command line tool for kuberneter cluster. One of the Master Processes called called Api Server enables interaction with the cluster. If you want to do anything in kubernetes you have to talk first to the Api Server. You can do it through three clients: *UI*, *API*, *CLI (Kubectl)*. The most powerfull is the Kubectl. Kubectl is not just for Minicube it can be used for cloud cluster too.

### Main Kubectl Commands - K8s CLI

Installation guides: - [Minicube](https://kubernetes.io/docs/tasks/tools/install-minikube) - [Kubectl](https://kubernetes.io/docs/tasks/tools/install-kubectl)

Starting minikube:

$ minikube start  
\* minikube v1.22.0 on Microsoft Windows 10 Enterprise 10.0.18363 Build 18363  
\* Using the virtualbox driver based on existing profile  
\* Starting control plane node minikube in cluster minikube  
\* Restarting existing virtualbox VM for "minikube" ...  
\* Preparing Kubernetes v1.21.2 on Docker 20.10.6 ...  
\* Verifying Kubernetes components...  
 - Using image gcr.io/k8s-minikube/storage-provisioner:v5  
 - Using image kubernetesui/metrics-scraper:v1.0.4  
 - Using image kubernetesui/dashboard:v2.1.0  
\* Enabled addons: default-storageclass, storage-provisioner, dashboard  
\* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

Returning the status of the nodes:

$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
minikube Ready control-plane,master 12h v1.21.2  
  
$ minikube status  
minikube  
type: Control Plane  
host: Running  
kubelet: Running  
apiserver: Running  
kubeconfig: Configured  
  
$ kubectl version  
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0", GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"clean", BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc", Platform:"windows/amd64"}  
Server Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.2", GitCommit:"092fbfbf53427de67cac1e9fa54aaa09a28371d7", GitTreeState:"clean", BuildDate:"2021-06-16T12:53:14Z", GoVersion:"go1.16.5", Compiler:"gc", Platform:"linux/amd64"}

#### Basic kubectl commands

**CRUD** commands:

* *Create* deployment: kubectl create deployment [name]
* *Edit* deployment: kubectl edit deployment [name]
* *Delete* deployment: kubectl delete deployment [name]

**Status of different kubernetes components**:

kubectl get nodes | pod | services | replicates | deployment

**Debugging pods**:

* *Log* to console: kuectl logs [pod name]
* Get *Interactive terminal*: kubectl exec -it [pod name] --bin/bash

$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
minikube Ready control-plane,master 13h v1.21.2  
  
$ kubectl get pod  
No resources found in default namespace.  
  
$ kubectl get services  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 13h  
  
$ kubectl create deployment nginx-depl --image=nginx   
deployment.apps/nginx-depl created  
  
$ kubectl get deployment   
NAME READY UP-TO-DATE AVAILABLE AGE  
nginx-depl 1/1 1 1 2m32s  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
nginx-depl-5c8bf76b5b-jghrz 1/1 Running 0 3m15s

The deployment has all the information (*blueprint*) for creating the pod. The most basic information for deployment is the *name* and the *image* to use. The rest is just default.

$ kubectl get replicaset  
NAME DESIRED CURRENT READY AGE  
nginx-depl-5c8bf76b5b 1 1 1 6m45s

The *replicaset* manages the replicas in pod.

| Layers of abstraction |
| --- |
| Deployment manages a … |
| Replicaset manages a … |
| Pod manages a … |
| Container |

Below the deployment should be managed everything by the Kubernetes.

$ kubectl edit deployment nginx-depl

Autogenerated config file:

# Please edit the object below. Lines beginning with a '#' will be ignored,  
# and an empty file will abort the edit. If an error occurs while saving this file will be  
# reopened with the relevant failures.  
#  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 annotations:  
 deployment.kubernetes.io/revision: "1"  
 creationTimestamp: "2021-07-18T07:20:53Z"  
 generation: 1  
 labels:  
 app: nginx-depl  
 name: nginx-depl  
 namespace: default  
 resourceVersion: "7558"  
 uid: 0a815d97-2c40-4747-a231-f4ba42cd4f5c  
spec:  
 progressDeadlineSeconds: 600  
 replicas: 1  
 revisionHistoryLimit: 10  
 selector:  
 matchLabels:  
 app: nginx-depl  
 strategy:  
 rollingUpdate:  
 maxSurge: 25%  
 maxUnavailable: 25%  
 type: RollingUpdate  
 template:  
 metadata:  
 creationTimestamp: null  
 labels:  
 app: nginx-depl  
 spec:  
 containers:  
 - image: nginx  
 imagePullPolicy: Always  
 name: nginx  
 resources: {}  
 terminationMessagePath: /dev/termination-log  
 terminationMessagePolicy: File  
 dnsPolicy: ClusterFirst  
 restartPolicy: Always  
 schedulerName: default-scheduler  
 securityContext: {}  
 terminationGracePeriodSeconds: 30  
status:  
 availableReplicas: 1  
 conditions:  
 - lastTransitionTime: "2021-07-18T07:21:15Z"  
 lastUpdateTime: "2021-07-18T07:21:15Z"  
 message: Deployment has minimum availability.  
 reason: MinimumReplicasAvailable  
 status: "True"  
 type: Available  
 - lastTransitionTime: "2021-07-18T07:20:53Z"  
 lastUpdateTime: "2021-07-18T07:21:15Z"  
 message: ReplicaSet "nginx-depl-5c8bf76b5b" has successfully progressed.  
 reason: NewReplicaSetAvailable  
 status: "True"  
 type: Progressing  
 observedGeneration: 1  
 readyReplicas: 1  
 replicas: 1  
 updatedReplicas: 1

Edit the *nginx* versoin to 1.16 and save the file.

$ kubectl get pod  
nginx-depl-5c8bf76b5b-jghrz 1/1 Running 0 17m  
nginx-depl-7fc44fc5d4-w5zmd 0/1 ContainerCreating 0 16s  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
nginx-depl-7fc44fc5d4-w5zmd 1/1 Running 0 27s  
  
$ kubectl get replicaset  
nginx-depl-5c8bf76b5b 0 0 0 21m  
nginx-depl-7fc44fc5d4 1 1 1 3m39s

Debugging pods:

$ kubectl create deployment mongo-depl --image=mongo  
deployment.apps/mongo-depl created  
  
$ kubectl get pod  
mongo-depl-5fd6b7d4b4-xxzpf 0/1 ContainerCreating 0 40s  
nginx-depl-7fc44fc5d4-w5zmd 1/1 Running 0 17m  
  
$ kubectl logs mongo-depl-5fd6b7d4b4-xxzpf  
{"t":{"$date":"2021-07-18T07:56:13.001+00:00"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"-","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}  
{"t":{"$date":"2021-07-18T07:56:13.001+00:00"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"-","msg":"Initialized wire specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion":0,"maxWireVersion":13},"isInternalClient":true}}}  
{"t":{"$date":"2021-07-18T07:56:13.002+00:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}  
{"t":{"$date":"2021-07-18T07:56:13.002+00:00"},"s":"I", "c":"NETWORK", "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required, set tcpFastOpenServer, tcpFastOpenClient, and tcpFastOpenQueueSize."}  
{"t":{"$date":"2021-07-18T07:56:13.002+00:00"},"s":"W", "c":"ASIO", "id":22601, "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"main","msg":"Successfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","ns":"config.tenantMigrationDonors"}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"REPL", "id":5123008, "ctx":"main","msg":"Successfully registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","ns":"config.tenantMigrationRecipients"}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"CONTROL", "id":4615611, "ctx":"initandlisten","msg":"MongoDB starting","attr":{"pid":1,"port":27017,"dbPath":"/data/db","architecture":"64-bit","host":"mongo-depl-5fd6b7d4b4-xxzpf"}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"CONTROL", "id":23403, "ctx":"initandlisten","msg":"Build Info","attr":{"buildInfo":{"version":"5.0.0","gitVersion":"1184f004a99660de6f5e745573419bda8a28c0e9","openSSLVersion":"OpenSSL 1.1.1f 31 Mar 2020","modules":[],"allocator":"tcmalloc","environment":{"distmod":"ubuntu2004","distarch":"x86\_64","target\_arch":"x86\_64"}}}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"CONTROL", "id":51765, "ctx":"initandlisten","msg":"Operating System","attr":{"os":{"name":"Ubuntu","version":"20.04"}}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"CONTROL", "id":21951, "ctx":"initandlisten","msg":"Options set by command line","attr":{"options":{"net":{"bindIp":"\*"}}}}  
{"t":{"$date":"2021-07-18T07:56:13.003+00:00"},"s":"I", "c":"STORAGE", "id":22297, "ctx":"initandlisten","msg":"Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem","tags":["startupWarnings"]}  
{"t":{"$date":"2021-07-18T07:56:13.004+00:00"},"s":"I", "c":"STORAGE", "id":22315, "ctx":"initandlisten","msg":"Opening WiredTiger","attr":{"config":"create,cache\_size=2394M,session\_max=33000,eviction=(threads\_min=4,threads\_max=4),config\_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),builtin\_extension\_config=(zstd=(compression\_level=6)),file\_manager=(close\_idle\_time=600,close\_scan\_interval=10,close\_handle\_minimum=250),statistics\_log=(wait=0),verbose=[recovery\_progress,checkpoint\_progress,compact\_progress],"}}  
{"t":{"$date":"2021-07-18T07:56:13.518+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1626594973:518393][1:0x7fc5334b6c80], txn-recover: [WT\_VERB\_RECOVERY | WT\_VERB\_RECOVERY\_PROGRESS] Set global recovery timestamp: (0, 0)"}}  
{"t":{"$date":"2021-07-18T07:56:13.518+00:00"},"s":"I", "c":"STORAGE", "id":22430, "ctx":"initandlisten","msg":"WiredTiger message","attr":{"message":"[1626594973:518446][1:0x7fc5334b6c80], txn-recover: [WT\_VERB\_RECOVERY | WT\_VERB\_RECOVERY\_PROGRESS] Set global oldest timestamp: (0, 0)"}}  
{"t":{"$date":"2021-07-18T07:56:13.520+00:00"},"s":"I", "c":"STORAGE", "id":4795906, "ctx":"initandlisten","msg":"WiredTiger opened","attr":{"durationMillis":516}}  
{"t":{"$date":"2021-07-18T07:56:13.520+00:00"},"s":"I", "c":"RECOVERY", "id":23987, "ctx":"initandlisten","msg":"WiredTiger recoveryTimestamp","attr":{"recoveryTimestamp":{"$timestamp":{"t":0,"i":0}}}}  
{"t":{"$date":"2021-07-18T07:56:13.526+00:00"},"s":"I", "c":"STORAGE", "id":4366408, "ctx":"initandlisten","msg":"No table logging settings modifications are required for existing WiredTiger tables","attr":{"loggingEnabled":true}}  
{"t":{"$date":"2021-07-18T07:56:13.526+00:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Timestamp monitor starting"}  
{"t":{"$date":"2021-07-18T07:56:13.527+00:00"},"s":"W", "c":"CONTROL", "id":22120, "ctx":"initandlisten","msg":"Access control is not enabled for the database. Read and write access to data and configuration is unrestricted","tags":["startupWarnings"]}  
{"t":{"$date":"2021-07-18T07:56:13.527+00:00"},"s":"I", "c":"STORAGE", "id":20320, "ctx":"initandlisten","msg":"createCollection","attr":{"namespace":"admin.system.version","uuidDisposition":"provided","uuid":{"uuid":{"$uuid":"53e3bc29-ba81-4f95-9ee4-6b098a884179"}},"options":{"uuid":{"$uuid":"53e3bc29-ba81-4f95-9ee4-6b098a884179"}}}}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"initandlisten","msg":"Index build: done building","attr":{"buildUUID":null,"namespace":"admin.system.version","index":"\_id\_","commitTimestamp":null}}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"REPL", "id":20459, "ctx":"initandlisten","msg":"Setting featureCompatibilityVersion","attr":{"newVersion":"5.0"}}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"NETWORK", "id":4915702, "ctx":"initandlisten","msg":"Updated wire specification","attr":{"oldSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion":0,"maxWireVersion":13},"isInternalClient":true},"newSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":13,"maxWireVersion":13},"outgoing":{"minWireVersion":13,"maxWireVersion":13},"isInternalClient":true}}}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"NETWORK", "id":4915702, "ctx":"initandlisten","msg":"Updated wire specification","attr":{"oldSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":13,"maxWireVersion":13},"outgoing":{"minWireVersion":13,"maxWireVersion":13},"isInternalClient":true},"newSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":13,"maxWireVersion":13},"outgoing":{"minWireVersion":13,"maxWireVersion":13},"isInternalClient":true}}}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"STORAGE", "id":5071100, "ctx":"initandlisten","msg":"Clearing temp directory"}  
{"t":{"$date":"2021-07-18T07:56:13.531+00:00"},"s":"I", "c":"CONTROL", "id":20536, "ctx":"initandlisten","msg":"Flow Control is enabled on this deployment"}  
{"t":{"$date":"2021-07-18T07:56:13.532+00:00"},"s":"I", "c":"FTDC", "id":20625, "ctx":"initandlisten","msg":"Initializing full-time diagnostic data capture","attr":{"dataDirectory":"/data/db/diagnostic.data"}}  
{"t":{"$date":"2021-07-18T07:56:13.532+00:00"},"s":"I", "c":"STORAGE", "id":20320, "ctx":"initandlisten","msg":"createCollection","attr":{"namespace":"local.startup\_log","uuidDisposition":"generated","uuid":{"uuid":{"$uuid":"74bd9ca9-1b05-47d0-935b-3eef86189d09"}},"options":{"capped":true,"size":10485760}}}  
{"t":{"$date":"2021-07-18T07:56:13.536+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"initandlisten","msg":"Index build: done building","attr":{"buildUUID":null,"namespace":"local.startup\_log","index":"\_id\_","commitTimestamp":null}}  
{"t":{"$date":"2021-07-18T07:56:13.537+00:00"},"s":"I", "c":"STORAGE", "id":20320, "ctx":"LogicalSessionCacheRefresh","msg":"createCollection","attr":{"namespace":"config.system.sessions","uuidDisposition":"generated","uuid":{"uuid":{"$uuid":"eeec37d8-d18a-4521-aef0-9b22dd3c6107"}},"options":{}}}  
{"t":{"$date":"2021-07-18T07:56:13.538+00:00"},"s":"I", "c":"CONTROL", "id":20712, "ctx":"LogicalSessionCacheReap","msg":"Sessions collection is not set up; waiting until next sessions reap interval","attr":{"error":"NamespaceNotFound: config.system.sessions does not exist"}}  
{"t":{"$date":"2021-07-18T07:56:13.538+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"/tmp/mongodb-27017.sock"}}  
{"t":{"$date":"2021-07-18T07:56:13.538+00:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"0.0.0.0"}}  
{"t":{"$date":"2021-07-18T07:56:13.538+00:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"listener","msg":"Waiting for connections","attr":{"port":27017,"ssl":"off"}}  
{"t":{"$date":"2021-07-18T07:56:13.544+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building","attr":{"buildUUID":null,"namespace":"config.system.sessions","index":"\_id\_","commitTimestamp":null}}  
{"t":{"$date":"2021-07-18T07:56:13.544+00:00"},"s":"I", "c":"INDEX", "id":20345, "ctx":"LogicalSessionCacheRefresh","msg":"Index build: done building","attr":{"buildUUID":null,"namespace":"config.system.sessions","index":"lsidTTLIndex","commitTimestamp":null}}

$ kubectl describe pod mongo-depl-5fd6b7d4b4-xxzpf  
Name: mongo-depl-5fd6b7d4b4-xxzpf  
Namespace: default  
Priority: 0  
Node: minikube/192.168.99.100  
Start Time: Sun, 18 Jul 2021 10:55:14 +0300  
Labels: app=mongo-depl  
 pod-template-hash=5fd6b7d4b4  
Annotations: <none>  
Status: Running  
IP: 172.17.0.5  
IPs:  
 IP: 172.17.0.5  
Controlled By: ReplicaSet/mongo-depl-5fd6b7d4b4  
Containers:  
 mongo:  
 Container ID: docker://5eb4e66d8c6bcdb749c7dce30a1a1b39e7560f5ba6242d8b140abb0b4df4370e  
 Image: mongo  
 Image ID: docker-pullable://mongo@sha256:f4ff7bb4291eb5d3f530a726fc524ba8e4318d652e64f2d58560ff87d083a84c  
 Port: <none>  
 Host Port: <none>  
 State: Running  
 Started: Sun, 18 Jul 2021 10:56:12 +0300  
 Ready: True  
 Restart Count: 0  
 Environment: <none>  
 Mounts:  
 /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-42bf4 (ro)  
Conditions:  
 Type Status  
 Initialized True  
 Ready True  
 ContainersReady True  
 PodScheduled True  
Volumes:  
 kube-api-access-42bf4:  
 Type: Projected (a volume that contains injected data from multiple sources)  
 TokenExpirationSeconds: 3607  
 ConfigMapName: kube-root-ca.crt  
 ConfigMapOptional: <nil>  
 DownwardAPI: true  
QoS Class: BestEffort  
Node-Selectors: <none>  
Tolerations: node.kubernetes.io/not-ready:NoExecute op=Exists for 300s  
 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s  
Events:  
 Type Reason Age From Message  
 ---- ------ ---- ---- -------  
 Normal Scheduled 3m49s default-scheduler Successfully assigned default/mongo-depl-5fd6b7d4b4-xxzpf to minikube  
 Normal Pulling 3m48s kubelet Pulling image "mongo"  
 Normal Pulled 2m51s kubelet Successfully pulled image "mongo" in 57.615189792s  
 Normal Created 2m51s kubelet Created container mongo  
 Normal Started 2m51s kubelet Started container mongo

Getting a terminal to a given pod:

$ kubectl exec -it mongo-depl-5fd6b7d4b4-xxzpf -- bin/bash  
root@mongo-depl-5fd6b7d4b4-xxzpf:/#  
  
# ls  
bin data docker-entrypoint-initdb.d home lib lib64 media opt root sbin sys usr  
boot dev etc js-yaml.js lib32 libx32 mnt proc run srv tmp var  
  
# exit

Removing a deployment:

$ kubectl delete deployment mongo-depl  
deployment.apps "mongo-depl" deleted  
  
$ kubectl get pod   
NAME READY STATUS RESTARTS AGE  
nginx-depl-7fc44fc5d4-w5zmd 1/1 Running 0 27m  
  
$ kubectl delete deployment nginx-depl  
deployment.apps "nginx-depl" deleted  
  
$ kubectl get pod   
NAME READY STATUS RESTARTS AGE  
nginx-depl-7fc44fc5d4-w5zmd 0/1 Terminating 0 28m  
  
$ kubectl get pod   
No resources found in default namespace.

Usually configuration is not done over arguments of the kubectl create deployment command. Instead it is used the

$ kubectl apply -f nginx-deployment.yaml

### K8s YAML Configuration File

**nginx-deployment.yaml**:

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

Apply configuration:

$ kubectl apply -f nginx-deployment.yaml  
deployment.apps/nginx-deployment created  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
nginx-deployment-644599b9c9-pb7sj 1/1 Running 0 70s  
  
$ kubectl get deployment  
NAME READY UP-TO-DATE AVAILABLE AGE  
nginx-deployment 1/1 1 1 115s

Changing the number of replicas in the yaml:

$ kubectl apply -f nginx-deployment.yaml  
deployment.apps/nginx-deployment configured  
  
$ kubectl get deployment  
NAME READY UP-TO-DATE AVAILABLE AGE  
nginx-deployment 2/2 2 2 6m27s  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
nginx-deployment-644599b9c9-2dtsq 1/1 Running 0 2m13s  
nginx-deployment-644599b9c9-pb7sj 1/1 Running 0 7m10s

#### Syntax and content of the kubernetes config file

In kubernetes every config file has 3 parts:

1. Metadata
2. Specification
3. Status: It will be automaticaly generated and added by kubernetes. This is the basis of the self healing feature of kubernetes provides. Status information comes from *etcd*.

The configuration file format: - is yaml: human friendly data serialization, standard for all programming languages. - syntax: strict indentation. - store config file with your code or its own git repository.

*Blueprint for pods*: under the *template* section there is another configuration which has its own metadata and spec section. This applies to a pod and will be the *blueprint* for a pod.

#### Connecting components

The way connection is established is using *labels* and *selectors*.

Connecting Deployment to pods: In the metadata you give a component like *deployment* or pod a key-value pair.

label:  
 app: nginx

That label just sticks to that component. Pods get the label through the template blueprint.

template:  
 metadata:  
 labels:   
 app:nginx

We tell the deployment to connect or match all the labels by the selector.

selector:   
 matchLabels:  
 app: nginx

Deployment has its own label.

labels:  
 app: nginx

And this label is used by the service selector. In the specification of the service we specify a selector which basically makes a connection between the service and the deployment or it pods.

apiVersion: v1  
kind: Service  
metadata:  
 name: nginx-service  
spec:  
 selector:  
 app: nginx

Another thing it has to be configured in the services and pod are containers.

**Service**

ports:  
 - protocol: TCP  
 port: 80  
 targetPort: 8080

**Container**

ports:  
 -containerPort: 8080

Container port should match the target port.

#### Creating both component and service

**nginx-deployment.yaml**:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: nginx-deployment  
 labels:  
 app: nginx  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: nginx  
 template:  
 metadata:  
 labels:  
 app: nginx  
 spec:  
 containers:  
 - name: nginx  
 image: nginx:1.16  
 ports:  
 - containerPort: 8080

**nginx-service.yaml**:

apiVersion: v1  
kind: Service  
metadata:  
 name: nginx-service  
spec:  
 selector:  
 app: nginx  
 ports:  
 - protocol: TCP  
 port: 80  
 targetPort: 8080

deploying the pod and the service:

$ kubectl apply -f nginx-deployment.yaml  
deployment.apps/nginx-deployment configured  
  
$ kubectl apply -f nginx-service.yaml  
service/nginx-service created  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
nginx-deployment-f4b7bbcbc-4bxzx 1/1 Running 0 6m  
nginx-deployment-f4b7bbcbc-nsmk6 1/1 Running 0 6m2s  
  
$ kubectl get service  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 15h  
nginx-service ClusterIP 10.102.209.134 <none> 80/TCP 3m58s  
  
$ kubectl describe service nginx-service  
Name: nginx-service  
Namespace: default  
Labels: <none>  
Annotations: <none>  
Selector: app=nginx  
Type: ClusterIP  
IP Family Policy: SingleStack  
IP Families: IPv4  
IP: 10.102.209.134  
IPs: 10.102.209.134  
Port: <unset> 80/TCP  
TargetPort: 8080/TCP  
Endpoints: 172.17.0.6:8080,172.17.0.7:8080  
Session Affinity: None  
Events: <none>  
  
$ kubectl get pod -o wide  
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  
nginx-deployment-f4b7bbcbc-4bxzx 1/1 Running 0 10m 172.17.0.6 minikube <none> <none>  
nginx-deployment-f4b7bbcbc-nsmk6 1/1 Running 0 10m 172.17.0.7 minikube <none> <none>

Status:

$ kubectl get deployment nginx-deployment -o yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 annotations:  
 deployment.kubernetes.io/revision: "2"  
 kubectl.kubernetes.io/last-applied-configuration: |  
 {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx-deployment","namespace":"default"},"spec":{"replicas":2,"selector":{"matchLabels":{"app":"nginx"}},"template":{"metadata":{"labels":{"app":"nginx"}},"spec":{"containers":[{"image":"nginx:1.16","name":"nginx","ports":[{"containerPort":8080}]}]}}}}  
 creationTimestamp: "2021-07-18T08:19:24Z"  
 generation: 3  
 labels:  
 app: nginx  
 name: nginx-deployment  
 namespace: default  
 resourceVersion: "13151"  
 uid: 13a561f0-3356-4d0c-8ffb-42375f83e75c  
spec:  
 progressDeadlineSeconds: 600  
 replicas: 2  
 revisionHistoryLimit: 10  
 selector:  
 matchLabels:  
 app: nginx  
 strategy:  
 rollingUpdate:  
 maxSurge: 25%  
 maxUnavailable: 25%  
 type: RollingUpdate  
 template:  
 metadata:  
 creationTimestamp: null  
 labels:  
 app: nginx  
 spec:  
 containers:  
 - image: nginx:1.16  
 imagePullPolicy: IfNotPresent  
 name: nginx  
 ports:  
 - containerPort: 8080  
 protocol: TCP  
 resources: {}  
 terminationMessagePath: /dev/termination-log  
 terminationMessagePolicy: File  
 dnsPolicy: ClusterFirst  
 restartPolicy: Always  
 schedulerName: default-scheduler  
 securityContext: {}  
 terminationGracePeriodSeconds: 30  
status:  
 availableReplicas: 2  
 conditions:  
 - lastTransitionTime: "2021-07-18T08:24:22Z"  
 lastUpdateTime: "2021-07-18T08:24:22Z"  
 message: Deployment has minimum availability.  
 reason: MinimumReplicasAvailable  
 status: "True"  
 type: Available  
 - lastTransitionTime: "2021-07-18T08:19:24Z"  
 lastUpdateTime: "2021-07-18T09:30:37Z"  
 message: ReplicaSet "nginx-deployment-f4b7bbcbc" has successfully progressed.  
 reason: NewReplicaSetAvailable  
 status: "True"  
 type: Progressing  
 observedGeneration: 3  
 readyReplicas: 2  
 replicas: 2  
 updatedReplicas: 2

Config files can be used also for removing deployments.

$ kubectl delete -f nginx-deployment.yaml  
deployment.apps "nginx-deployment" deleted  
  
$ kubectl delete -f nginx-service.yaml  
service "nginx-service" deleted

### Hands-On Demo

Complete application setup with kubernetes components.

**mongo.yaml**:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mongodb-deployment  
 labels:   
 app: mongodb  
spec:  
 replicas: 1  
 selector:   
 matchLabels:  
 app: mongodb  
 template:  
 metadata:  
 labels:  
 app: mongodb  
 spec:  
 containers:  
 - name: mongodb  
 image: mongo  
 ports:   
 - containerPort: 27017  
 env:  
 - name: MONGO\_INITDB\_ROOT\_USERNAME  
 value:  
 - name: MONGO\_INITDB\_ROOT\_PASSWORD  
 value:

**mongo-secret.yaml**

apiVersion: v1  
kind: Secret  
metadata:  
 name: mongodb-secret  
type: Opaque  
data:  
 mongo-root-username: dXN1cm5hbWU=  
 mongo-root-password: cGFzc3dvmQ=

Creating the secret:

$ kubectl apply -f mongo-secret.yaml  
secret/mongodb-secret created  
  
$ kubectl get secret  
NAME TYPE DATA AGE  
default-token-ckvbq kubernetes.io/service-account-token 3 16h  
mongodb-secret Opaque 2 40s

complete the **mongo.yaml**

...  
env:  
- name: MONGO\_INITDB\_ROOT\_USERNAME  
valueFrom:   
 secretKeyRef:  
 name: mongodb-secret  
 key: mongo-root-username  
- name: MONGO\_INITDB\_ROOT\_PASSWORD  
valueFrom:   
 secretKeyRef:  
 name: mongodb-secret  
 key: mongo-root-password

Deploy the mongodb

$ kubectl apply -f mongo.yaml  
deployment.apps/mongodb-deployment created

**mongo.yaml**:

Two configurations can be added in one file. Service configuration will be added at th end of the service configuration. Actually thy belong together. They will be separated with 3 dashes: ---

...  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: mongodb-service  
spec:  
 selector:  
 app: mongodb  
 ports:  
 - protocol: TCP  
 port: 27017  
 targetPort: 27017

Creating the service by applying the same file as before:

$ kubectl apply -f mongo.yaml  
deployment.apps/mongodb-deployment unchanged  
service/mongodb-service created  
  
$ kubectl get service  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 35h  
mongodb-service ClusterIP 10.99.195.192 <none> 27017/TCP 12m  
  
$ kubectl describe service  
Name: kubernetes  
Namespace: default  
Labels: component=apiserver  
 provider=kubernetes  
Annotations: <none>  
Selector: <none>  
Type: ClusterIP  
IP Family Policy: SingleStack  
IP Families: IPv4  
IP: 10.96.0.1  
IPs: 10.96.0.1  
Port: https 443/TCP  
TargetPort: 8443/TCP  
Endpoints: 192.168.99.100:8443  
Session Affinity: None  
Events: <none>  
  
  
Name: mongodb-service  
Namespace: default  
Labels: <none>  
Annotations: <none>  
Selector: app=mongodb  
Type: ClusterIP  
IP Family Policy: SingleStack  
IP Families: IPv4  
IP: 10.99.195.192  
IPs: 10.99.195.192  
Port: <unset> 27017/TCP  
TargetPort: 27017/TCP  
Endpoints: 172.17.0.3:27017  
Session Affinity: None  
Events: <none>

**mongo-express.yaml**:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mongo-express  
 labels:   
 app: mongo-express  
spec:  
 replicas: 1  
 selector:  
 matchLabels:   
 app: mongo-express  
 template:  
 metadata:  
 labels:  
 app: mongo-express  
 spec:  
 containers:  
 - name: mongo-express  
 image: mongo-express  
 ports:   
 - containerPort: 8081  
 env:  
 - name: ME\_CONFIG\_MONGODB\_ADMINUSERNAME  
 valueFrom:   
 secretKeyRef:  
 name: mongodb-secret  
 key: mongo-root-username  
 - name: ME\_CONFIG\_MONGODB\_ADMINPASSWORD  
 valueFrom:   
 secretKeyRef:  
 name: mongodb-secret  
 key: mongo-root-password  
 - name: ME\_CONFIG\_MONGODB\_SERVER  
 value:

**mongo-configmap.yaml**:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: mongodb-configmap  
data:  
 database\_url: mongodb-service

**mongo-express.yaml**:

...  
- name: ME\_CONFIG\_MONGODB\_SERVER  
valueFrom:   
 configMapKeyRef:  
 name: mongodb-configmap  
 key: database\_url

Creating the config map:

$ kubectl apply -f mongo-configmap.yaml  
configmap/mongodb-configmap created  
  
$ kubectl apply -f mongo-express.yaml  
deployment.apps/mongo-express created  
  
$ kubectl get pod  
NAME READY STATUS RESTARTS AGE  
mongo-express-78fcf796b8-hqtzf 1/1 Running 0 79s  
mongodb-deployment-8f6675bc5-tnvhb 1/1 Running 1 20h  
  
$ kubectl logs mongo-express-78fcf796b8-hqtzf  
Welcome to mongo-express  
------------------------  
  
  
(node:10) [MONGODB DRIVER] Warning: Current Server Discovery and Monitoring engine is deprecated, and will be removed in a future version. To use the new Server Discover and Monitoring engine, pass option { useUnifiedTopology: true } to the MongoClient constructor.  
Mongo Express server listening at http://0.0.0.0:8081  
[31mServer is open to allow connections from anyone (0.0.0.0)[39m  
[31mbasicAuth credentials are "admin:pass", it is recommended you change this in your config.js![39m

**mongo-express.yaml**:

---  
apiVersion: v1  
kind: Service  
metadata:  
 name: mongo-express-service  
spec:  
 selector:  
 app: mongo-express  
 type: LoadBalancer  
 ports:  
 - protocol: TCP  
 port: 8081  
 targetPort: 8081  
 nodePort: 30000

By setting the type of the service to LoadBalancer it will make the service external. The nodePort will be the port where the external IP address will be open. The node port has a valid range: 30000-32767.

Start the extenal service:

$ kubectl apply -f mongo-express.yaml  
deployment.apps/mongo-express unchanged  
service/mongo-express-service created  
  
$ kubectl get service  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 37h  
mongo-express-service LoadBalancer 10.97.45.3 <pending> 8081:30100/TCP 85s  
mongodb-service ClusterIP 10.99.195.192 <none> 27017/TCP 113m  
  
$ minikube service mongo-express-service  
|-----------|-----------------------|-------------|-----------------------------|  
| NAMESPACE | NAME | TARGET PORT | URL |  
|-----------|-----------------------|-------------|-----------------------------|  
| default | mongo-express-service | 8081 | http://192.168.99.100:30100 |  
|-----------|-----------------------|-------------|-----------------------------|  
\* Opening service default/mongo-express-service in default browser...

## Advanced Concepts

### K8s Namespaces - Organize your Components

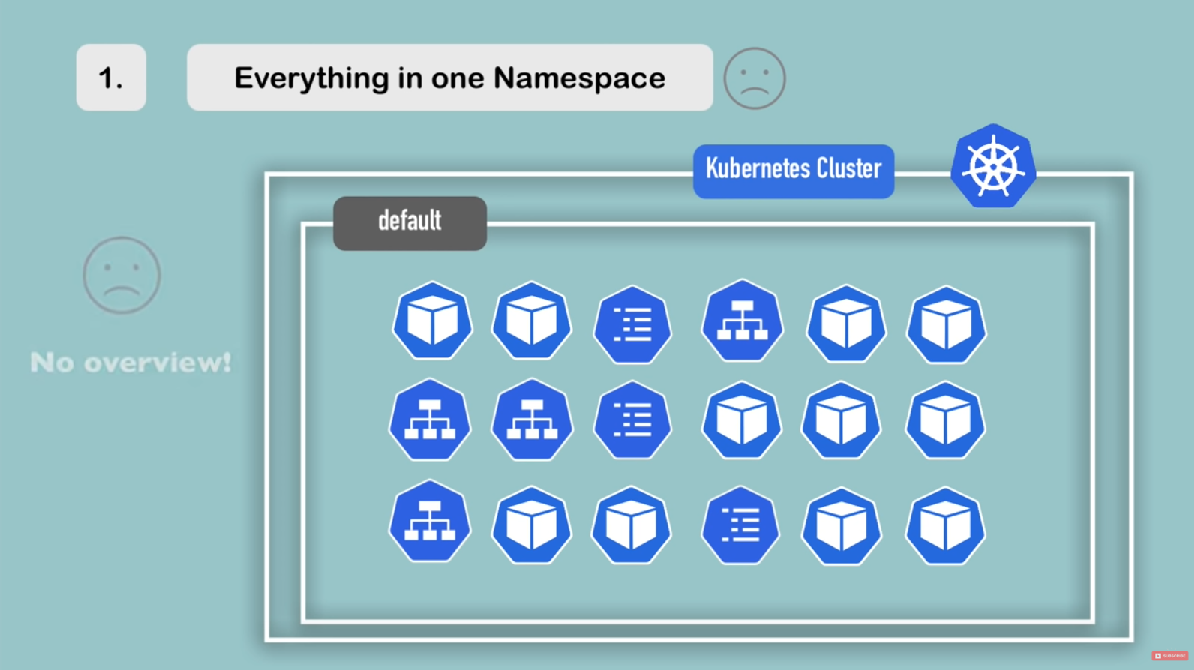
What is a namespace? - organize resources in namespaces - virtual cluster inside a cluster - 4 namespaces out of the box + kubernetes-dashboard is shipped only with minikube + kube-system \* Is not ment for your use. Don’t create or modify anything in it. \* Components deployed: system processes, managing processes + kube-public \* publicly accessable data + kube-node-lease \* information of each node availability + default \* will be used for your resources if you don’t create expicitly a namespace

$ kubectl get namespace  
NAME STATUS AGE  
default Active 37h  
kube-node-lease Active 37h  
kube-public Active 37h  
kube-system Active 37h  
kubernetes-dashboard Active 37h  
  
$ kubectl create namespace my-namespace  
namespace/my-namespace created

Namespace configuration file

**mysql-configmap.yaml**:

apiVersion: v1  
kind: ConfigMap  
metadata:  
 name: mysql-configmap  
 namespace: my-namespace  
data:  
 db\_url: mysql-service.database

What is the need for creating namespaces? 

* Group resources into namspaces
  + Database
  + Monitoring
  + Elastik Stack
  + Nginx-Ingress
* Many teams same application
  + Project A
  + Project B
* Resource Sharing: Both Staging and Environment can use the Nginx-Ingress Controller and Elastick Stack
  + Staging
  + Development
  + Nginx-Ingress Controller
  + Elastick Stack
* Resource Sharing: Blue/Green Deployment
  + Active Production Version
  + Next Production version
* Access and Resource Limits on Namespaces: Limit the resources each namespace consumes
  + Project A
  + Project B

Characteristics of namespaces - You can’t access most of the resources from another namespace - Services can be access in another namespace - There are some components in kubernetes which can’t be created within a namespace. Listing such services with the command: kubectl api-resources --namespaced=false + volume + node

How to create components in a namespace?

$ kubectl apply -f mysql-configmap.yaml --namespace=my-namespace  
configmap/mysql-configmap created

or inside the configuration file itself.

**mysql-configmap.yaml**:

...  
metadata:  
 name: mysql-configmap  
 namespace: my-namespace  
...

Test the namespace

$ kubectl get mysql-configmap -n my-namespace  
NAME DATA AGE  
mysql-configmap 1 5m37s

Changing the active namespace

Under windows actually it was quite difficult to install [kubens](https://github.com/ahmetb/kubectx#installation). So I din’t do it.

$ kubens my-namespace

### Kubernetes Ingress

External request to be able to reach your application. - Easy way: external service + See the example above the service with type: *LoadBalancer* - Ingress (a kubernetes component)

**myapp-ingress.yaml**:

apiVersion: networking.k8s.io/v1beta1  
kind: Ingress  
metadata:  
 name: myapp-ingress  
spec:  
 rules:  
 - host: myapp.com  
 http:  
 paths:  
 - backend:  
 serviceName: myapp-internal-servcie  
 servicePort: 8080

We need an *ingress implementation* too called *ingress controller*. Ingress controller are a pod or a set of pods which evaluates and processes ingress rules and this way manages all the redirections. This will be the entry point into the cluster. It has to be decided which of many third party implamentation to choose. - Kubernetes Nginx Ingress Controller (is on of them)

Install **Ingress Controller** in minikube. Automatically configures and starts the *K8s Nginx Ingress Contreller*.

$ minikube addons enable ingress  
 - Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0  
 - Using image docker.io/jettech/kube-webhook-certgen:v1.5.1  
 - Using image docker.io/jettech/kube-webhook-certgen:v1.5.1  
\* Verifying ingress addon...  
\* The 'ingress' addon is enabled  
  
$ kubectl get pod -n ingress-nginx  
NAME READY STATUS RESTARTS AGE  
ingress-nginx-admission-create-fgcsf 0/1 Completed 0 7m34s  
ingress-nginx-admission-patch-99h7v 0/1 Completed 2 7m34s  
ingress-nginx-controller-59b45fb494-z8mz6 1/1 Running 0 7m34s

Creating an Ingress Rule that the Ingress Controller can evaluate

$ kubectl get all -n kubernetes-dashboard  
NAME READY STATUS RESTARTS AGE  
pod/dashboard-metrics-scraper-7976b667d4-5xxkh 1/1 Running 3 39h  
pod/kubernetes-dashboard-6fcdf4f6d-sn5ds 1/1 Running 4 39h  
  
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  
service/dashboard-metrics-scraper ClusterIP 10.97.146.149 <none> 8000/TCP 39h  
service/kubernetes-dashboard ClusterIP 10.111.22.219 <none> 80/TCP 39h  
  
NAME READY UP-TO-DATE AVAILABLE AGE  
deployment.apps/dashboard-metrics-scraper 1/1 1 1 39h  
deployment.apps/kubernetes-dashboard 1/1 1 1 39h  
  
NAME DESIRED CURRENT READY AGE  
replicaset.apps/dashboard-metrics-scraper-7976b667d4 1 1 1 39h  
replicaset.apps/kubernetes-dashboard-6fcdf4f6d 1 1 1 39h

**dashboard-ingress.yaml**:

apiVersion: networking.k8s.io/v1beta1  
kind: Ingress  
metadata:  
 name: dashboard-ingress  
 namespace: kubernetes-dashboard  
spec:  
 rules:  
 - host: dashboard.com  
 http:  
 paths:  
 - backend:   
 serviceName: kubernetes-dashboard  
 servicePort: 80

Creating the ingress rule:

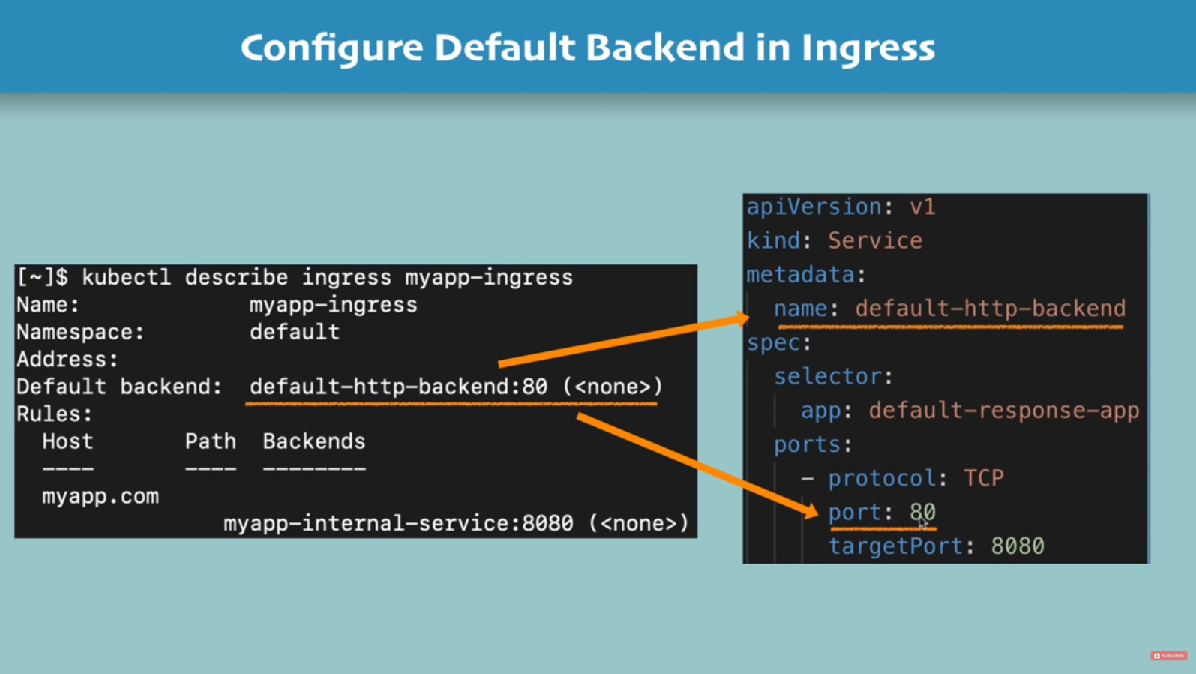
$ kubectl apply -f dashboard-ingress.yaml  
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+, unavailable in v1.22+; use networking.k8s.io/v1 Ingress  
ingress.networking.k8s.io/dashboard-ingress created  
  
$ kubectl get ingress -n kubernetes-dashboard  
NAME CLASS HOSTS ADDRESS PORTS AGE  
dashboard-ingress <none> dashboard.com 192.168.99.100 80 107s

Edit the [*hosts*](C:\Windows\System32\drivers\etc) file

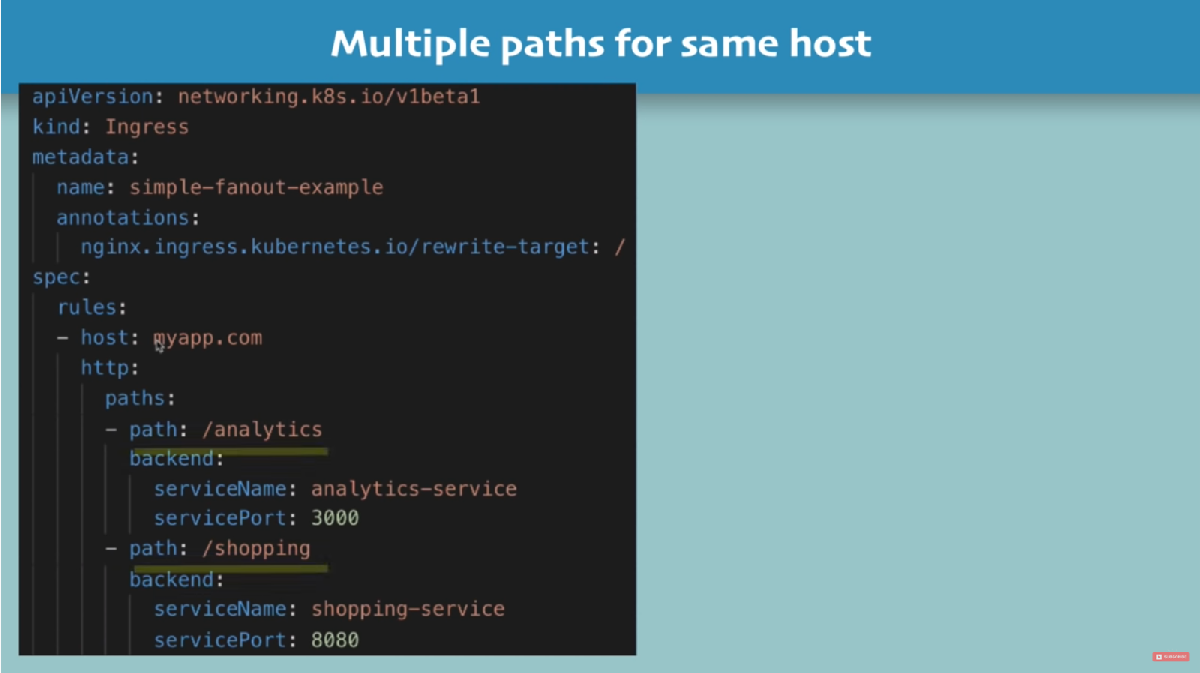
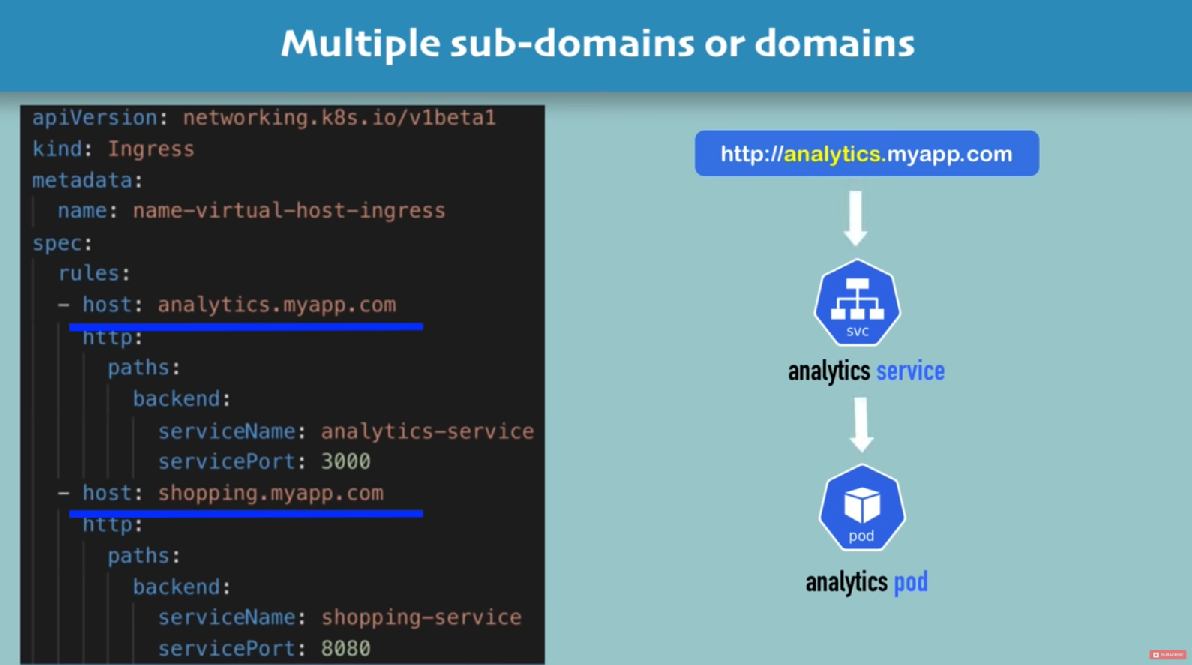
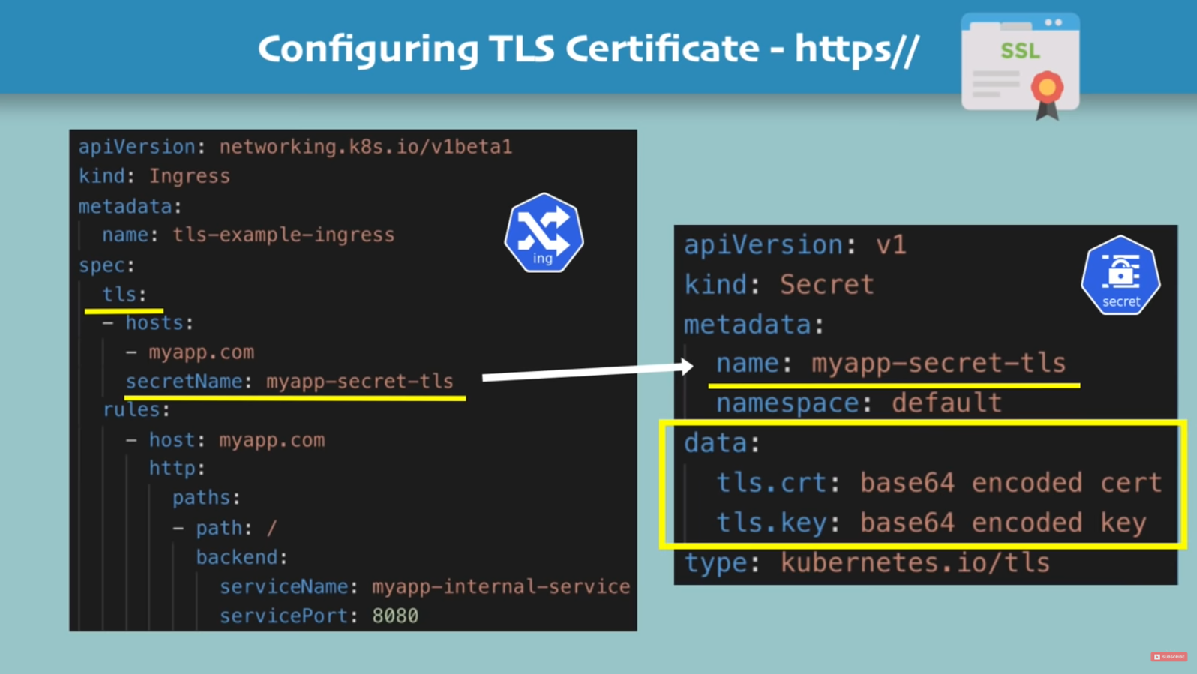
...  
192.168.99.100 dashboard.com

$ kubectl describe ingress dashboard-ingress -n kubernetes-dashboard  
Name: dashboard-ingress  
Namespace: kubernetes-dashboard  
Address: 192.168.99.100  
Default backend: default-http-backend:80 (<error: endpoints "default-http-backend" not found>)  
Rules:  
 Host Path Backends  
 ---- ---- --------  
 dashboard.com  
 kubernetes-dashboard:80 (172.17.0.4:9090)  
Annotations: <none>  
Events:  
 Type Reason Age From Message  
 ---- ------ ---- ---- -------  
 Normal Sync 14m (x2 over 14m) nginx-ingress-controller Scheduled for sync

Default http backend



More Usecases for Ingress

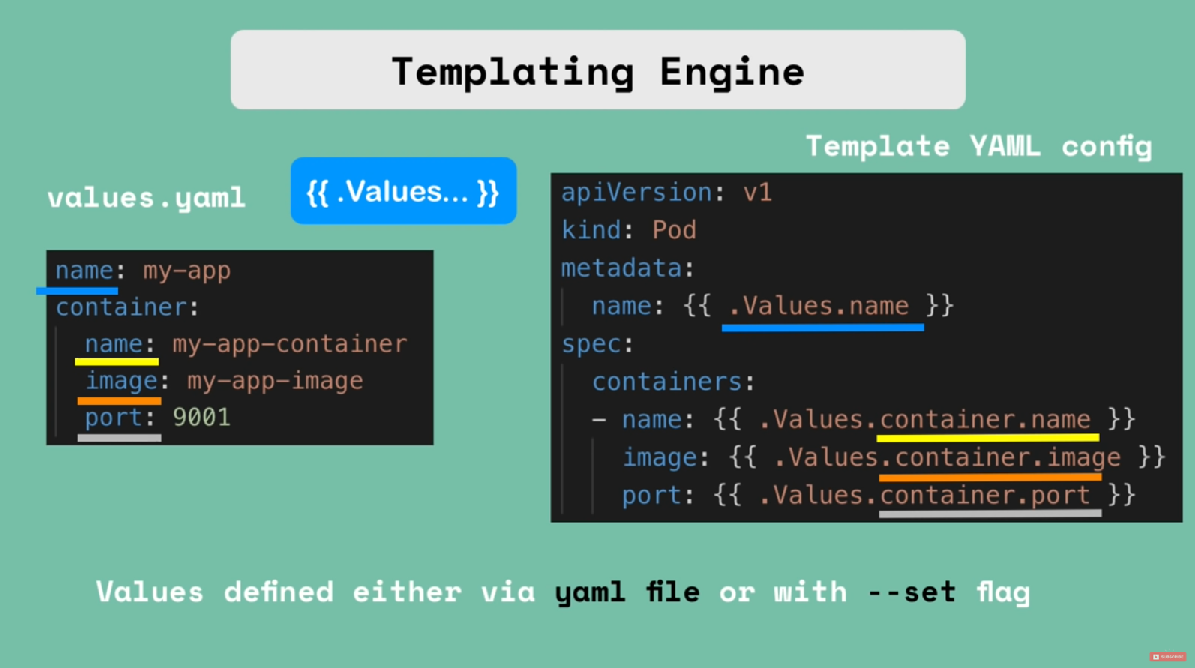
* multiple paths for same hosts 
* multiple subdomains 
* configuring TLS Certificates 

### Helm Package Manager

[install](https://github.com/helm/helm/releases)

What is helm? - package manager for kubernetes: packaging kubernetes yaml files and distribuiting them in public and private repositories. - The boundle of yamle files is called *Heml chart* - Create your own Helm chart with Helm - Push them to some helm repository - Download and use existing ones

It’s a templating engine 1) Given multiple microservices with identical deployment files only the name of the service is different. + With helm it can be defined a common blueprint for all the microservices. + Dynamic values are replaced by placeholders



1. Deploying the same set of apps in different cluseter environments

* Directory structure of Helm:

mychart/ toplevel mychart folder -> name of the chart  
 Chart.yaml meta info about chart  
 values.yaml values for the template file  
 charts/ chart dependencies folder  
 templates/ tempate folder -> the actual template files

1. Release management

$ helm upgrade <chartname>

The changes will be applied to existing deployment instead of creating new one. **Tiller** is a service which makes this part.

$ helm rollback <chartname>

Rolls back deployment

### Volumes - Persisting Data in Kubernetes

Persisting data with storage

* Persistent Volume
* Persistent Volume Claim
* Storage Class

The need for Volumes

* Data stored in databases will be gone when we restart changes will be gone. You have to explicitly configure for each application that needs saving data between restarts so that the storage doesn’t depend on the pod lifecycle.
  + If a pod restarts it will read data from that storage to get up to date data.
  + Storage must also be available on all nodes.
  + Highly available storage that will survive even if the cluster crashes.
* Having an application that writes and reads from pre configured directory, session files for application, …
  + you can configure any of these type of storage using kubernetes component called **Persistent volume**
  + Persistent volume can be tought as a cluster resource like: RAM, CPU that is used to store data.

Persistet volume gets created via yaml file. Since persisten volume is just an abstract component it must take the storage from the actual phisical storage, like: local hard drive from the cluster nodes, or your external nfs server or maybe cloud storage.

Where does this storage backend come from? Who configures it? Who makes it available to the cluster? Kubernetes doesn’t care about your actual storage, it gives you a persistent volume component as an interface to the actual storage that you as a mantainer have to take care of.

So you have to decide, what type of storage your cluster needs? You need to create and manage them by yourself. Think of storage as an external plugin to your cluster.

**storage-nfs-server.yaml**:

apiVersion: v1  
kind: PersistentVolume  
metadata:   
 name: pv-name  
spec:  
 capacity:  
 storage: 1Gi  
 volumeMode: Filesystem  
 accessMode:  
 - ReadWriteOnce  
 persistentVolumeReclaimPolicy: Recycle  
 storageClassName: slow  
 mountOption:  
 - hard  
 - nfsvers=4.0  
 nfs:  
 path: /dir/path/on/nfs/server  
 server: nfs-server-ip-address

**storage-google-cloud.yaml**:

apiVersion: v1  
kind: PersistentVolume  
metadata:   
 name: test-volume  
 labels:  
 failure-domain.beta.kubernetes.io/zone: us-centrall-a\_\_us-centrall-b  
spec:  
 capacity:  
 storage: 1Gi  
 accessMode:  
 - ReadWriteOnce  
 gcePersistentDisk:  
 pdName: my-data-disk  
 fsType: ext4

**storage-local.yaml**:

apiVersion: v1  
kind: PersistentVolume  
metadata:   
 name: example-pv  
spec:  
 capacity:  
 storage: 1Gi  
 volumeMode: Filesystem  
 accessMode:  
 - ReadWriteOnce  
 persistentVolumeReclaimPolicy: Delete  
 storageClassName: local-storage  
 local:  
 path: /mnt/disks/ssd1  
 nodeAffinity:  
 required:  
 nodeSelectorTerms:  
 - matchExpressions:  
 - key: kubernetes.io/hostname  
 operator: In  
 values:  
 - example node

Persistem volumes are not namespaced. They are accessible for the whole cluster.

Local vs. Remote volumes: - The local volume types violate the 2nd and 3rd requiments for data persitance + not being tied to one specific node rather to each node equally + surviving in cluster crash scenarios, therefore for database persistance you should always use remote storage.

Persistent volume are resourcces like CPU or RAM it have to be already there in the cluster when the pod that depend on it or uses it is created.

Two main roles in Kubernetes: - administrator: sets up and maintains the cluster and make sure that has enough resourcces - kubernetes user: deploys application in a cluster

In this case the kubernetes administrator would be the one - which configures the persistent storage. Make sure that the nfs-server is there and configured, create and configure a cloud storage which will be available for the cluster. - creates the persistent volume from this storage backends.

Delevopers expicitly need to configure their application yaml files to use those persistent volume components. Application has to claim that volume storage in another component naming *Persisten Volume Claim*. Persistent Volume Claims are also created with yaml file configurations.

**persitent-volume-claim.yaml**:

apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
 name: pvc-name  
spec:  
 storageClassName: manual  
 volumeMode: Filesystem  
 accessModes:   
 - ReadWriteOnce  
 resources:  
 requests:  
 storage: 1Gi

In the pods specificcation you have to use that claim

**pod.yaml**:

apiVersion: v1  
kind: Pod  
metadata:   
 name: mypod  
spec:  
 containers:  
 - name: myfrontend  
 image: nginx  
 volumeMounts:   
 - mountPath: "/var/www/html"  
 name: mypd  
 volumes:  
 - name: mypd  
 persistentVolumeClaim:   
 claimName: pvc-name

Claims must exist in the same cluster in which the pod is.

### K8s StatefulSet - Deploying Stateful Apps

It is a kubernetes component which is used specifically for statful apps. Examples of stateful apps are all databases.