What is K8s?

Tuesday, June 21, 2022 9:59 AM

What is K8s?

Open Source container orchestration tool (dev by Google)

- Manages containers
- Manage containerized apps in different environments

What problems does K8s solve?

Trend from Monolith to Microservices increased usage of containers

- Managing 100s if not 1000s of containers

Orchestration Tools Features

High Availability or no downtime
Scalability
Disaster recovery

Kubernetes Components

Tuesday, June 21, 2022 10:05 AM

Node & Pod

Node

server, physical or virtual

Pod

Smallest unit of K8s
Abstraction over container
Running environment on top of the container
Allows you to only interact with K8s, not directly with Docker
Usually 1 application per pod
Each pod gets it's own internal IP address & can communicate with other Pods
    New IP address on-creation
Pods are ephemeral - can die easily

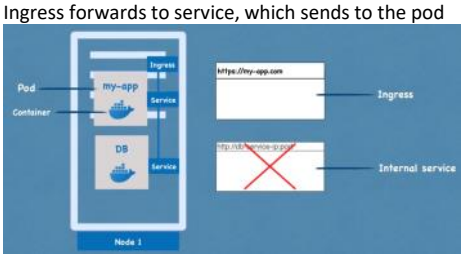
Service & Ingress

Service

Permanent IP address
Also, a Load Balancer
Lifecycle of Pod & service **not** connected
External Service - Allow outside servers/users connect to a pod

Ingress

Assign a domain to pod
Ingress forwards to service, which sends to the pod



ConfigMap & Secret

Need to be connected to container

ConfigMap

External configuration of your application
Can be used as environmental variables or properties file
Don't put credentials into ConfigMap

Secret

Used to store secret configurations; stored in base64 encoded format
Built-in security mechanism is not enabled by default

Data Storage

Data is not persistent between restarts

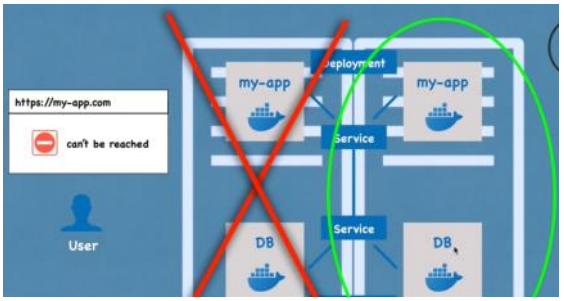
Volumes

Attaches physical hard-drive to pod
Can be on local machine or remote storage, outside the K8s cluster
K8s doesn't manage data persistance
You are responsible for backups/management of data

Deployment & Stateful Set

Deployment

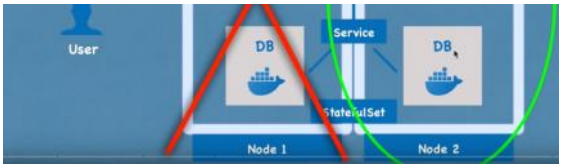
Blueprint for my-app pods
Practically, you don't create pods, you create deployments
Abstraction of Pods
FOR STATELESS APPLICATIONS
Database can't be replicated via Deployment since DB has State (it's data)
    Clones / replicas of DB need to access same data storage
    Need to control who's accessing data to avoid inconsistencies



Stateful Set

Should be used to create stateful applications - Elasticsearch, MongoDB, MySQL
FOR STATEFULL APPLICATION
Ensures DB reads/writes synchronized
Deploying StatefulSet not easy

Should be used to create stateful applications - Elasticsearch, MongoDB, MySQL
FOR STATEFULL APPLICATION
Ensures DB reads/writes synchronized
Deploying StatefulSet not easy
Common practice to host DB outside of K8s cluster & have STATELESS apps inside K8s cluster



K8s Architecture

Tuesday, June 21, 2022 10:31 AM

Node Processes

Worker Node

Worker machine in k8s cluster
Will have multiple Pods on it
Worker nodes do the actual work
3 processes must be installed on every Node:

- **Container Runtime** - e.g. Docker
- **Kubelet** - Process of K8s, interacts with both container & node
  - Responsible for applying configuration, starting the pod, & assigning resources
- **Kube Proxy** - Forwards request from services to pods
  - Communication via Services - Load balancer that catches the request and forwards to respective pod
  - Has intelligent forwarding logic
    - Makes sure comms are performant, low overhead
    - Will forward to pod on same node before sending out to other nodes

Master Processes

How you interact with your K8s cluster
Managing processes are done by Master Nodes
4 process run on every master node

- **API Server** - Cluster gateway, gets initial request / queries from cluster
  - Gatekeeper for authentication
  - Only 1 entrypoint in to the cluster (good for security)
- **Scheduler** - Schedules a new pod
  - Intelligent way of deciding which pod will get scheduled request
  - See your request & interpret number of resources you will need, then schedules on the node that has the resources to meet your Pod's needs
  - **Scheduler just decides which Node a new pod should be scheduled**
    - Kubelet receives request from the scheduler and actually executes the task
- **Controller Manger** - Detects cluster state changes
  - Detects when pods die & tries to restore cluster state
  - Controller Manage -> Scheduler -> Kubelet
- **etcd** - Key-value store of cluster state / cluster brain
  - Cluster changes get stored in key-value store
  - Retains information like:
    - What resources are available?
    - Did the cluster state change?
    - Is the cluster healthy?
  - **Application data is NOT stored in etcd**

K8s cluster typically has multiple Master nodes

- Load balance API Server requests
- Etcd forms distributed storage across master nodes

Example Cluster Set-up

Master Nodes need less resources than Worker Nodes since the worker nodes do the work
To **add new Master/Node server**:

1. Get new bare server

2. Install all the master/worker node processes
3. Add it to the cluster

Can infinitely scale horizontally

Minikube & Kubectl

Tuesday, June 21, 2022 10:52 AM

3 Ways to interact with K8s
- UI
- API
- CLI

Minikube

Test / Local Cluster Setup
1 node K8s cluster, where all master / worker processes run on one node
Docker runtime pre-installed
Creates Virtual Box on your laptop & Node runs in that Virtual Box
For Testing purposes
Minikube has Kubectl as a dependency; will install both
Comes with "minikube" command line tool for basic interactions with cluster
- Mainly for starting / stopping the cluster - management is done with Kubectl

Kubectl

Command Line Tool to Interact with your K8s cluster
Interacts directly with API Server installed on Master nodes
Most powerful method of interacting with K8s
Interacts with any K8s setup

Installation

Install Minikube (Mac, Linux and Windows): https://bit.ly/38bLcJy
Install Kubectl: https://bit.ly/32bSI2Z

Minikube Commands

Create & Start K8s & Select Hypervisor

minikube start --vm-driver=hyperkit
- Also configures kubectl to use "minikube"

Get status of nodes in cluster

kubectl get nodes
minikube status

Get Version

kubectl version

Main Kubectl Commands

Tuesday, June 21, 2022 11:34 AM

List Kubernetes components

kubectl get nodes
kubectl get pod
kubectl get services
kubectl get deployment
kubectl get replicaset

Create a pod

kubectl create
    Pod is smallest unit, but you are creating deployment (abstraction over pods)
kubectl create deployment NAME --image=image
    Image is required

Edit a Deployment

kubectl edit deployment NAME
    Brings up auto-generated config file of deployment
    If config edited - will start a new pod, then remove the old one
    In practice, would be done within Kubernetes Configuration files

Get Application Logs

kubectl logs POD NAME

Get state changes

kubectl describe pod POD NAME

Create Bash Shell within the Container

kubectl exec -it POD NAME -- bin/bash

Delete a deployment

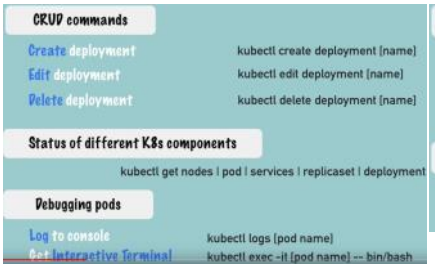
kubectl delete deployment NAME

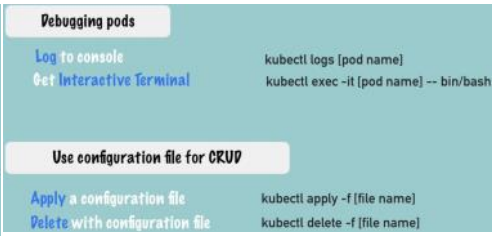
Create deployment from Configuration File

kubectl apply -f CONFIG FILE
    Can also be used to apply changes that have been made to the configuration file
    K8s knows when to create or update deployment

Replicaset - manages replicas of the pod
    Don't typically interact with

CRUD happens on Deployment level





YAML Config File in K8s

Tuesday, June 21, 2022 11:54 AM

Every configuration has 3 parts

Metadata

Contains things like name / labels

Specification (spec)

Configuration to apply to deployment
Attributes specific to the kind of deployment
apiVersion / kind - base level
Declares what you want to create

Status

Automatically generated & added by Kubernetes
    Desired state vs. Actual state
        How you want it configured vs. how it's actually configured
Status info for actual comes from etcd

YAML Format

Configuration location

Configuration files stored with your code or it's own git repo for configs

Pod configuration

spec.template
Has it's own "metadata" and "spec" section
Blueprint for a pod

Connecting Components

Components are controlled with the "kind" variable
Can have a **kind** of Deployment or Service

Labels & Selectors

Establish the connection
Metadata contains the labels
Spec contains the selectors
Any key-value pair for a component
Deployment & Pods get their own labels

Connecting Services to Deployments

Within the service, leverage the "spec.selector" config to select deployments/pods that are part of the service

Ports in Service & Pod

Service will have a list of ports to listen externally and where to redirect internally
Some pods "containerPort" should line-up with the service's "targetPort"

Validation of Service / Deployment Connection

To check that a service is hooked into a specific endpint

1. Get Metadata about service
    a. kubectl describe service SERVICE NAME
2. Under "Endpoints", it should list the IP addresses / ports of the Pods it is pointing at
3. Get additional information about Pods
    a. kubectl get pod -o wide

Validate Status

kubectl get deployment DEPLOYMENT NAME -o yaml

Demo Project MongoDB & MongoExpress

Wednesday, June 22, 2022 12:03 PM

Namespaces

Wednesday, June 22, 2022 11:24 AM

What is a Namespace?

Organize resources within Namespaces
Virtual cluster within a cluster
kubectl get namespaces

4 namespaces per Default

kubernetes-dashboard

Only with minikube
Will not be in standard cluster

kube-system

Do NOT create / modify in kube-system
System processes
Master / Kubectl process

kube-public

Publicly accessible data
ConfigMap which contains cluster information - accessible without authentication
kubectl cluster-info

kube-node-lease

Recent addition
Heartbeats of nodes
Each node has associated lease object in namespaces
Determines the availability of a node

Default

Resources you create are located here (unless specified otherwise)

Create a namespace

kubectl create namespace NAME

Can be created with a configuration file - better way
    (kind of ConfigMap)

Why use Namespaces

Default behavior- Everything is in one namespace

Difficult to have an overview of components within a namespace - **Better to group resources in Namespaces**
Logically grouping resources within your cluster
Examples:
    Database namespace for database resources
    Monitoring namespace for monitoring resources
    Elastic Stack namespace for Elastic stack (ELK)
According to Kubernetes documentation - Should not use for smaller projects or less than 10 users

Conflicts: Many teams, same application

Other teams can override deployment if they use the same name
    If using Jenkins or automation, wouldn't know they overwrite a deployment
Each team can work in their own namespace without disrupting the other

Resource Sharing: Staging & Development

Deploy common resources once and reach from both environments

Resource Sharing: Blue/Green Deployment

Two versions of production in 1 cluster:
    Current production (blue), next version of production (green)
Both current prod and next prod can leverage same resuorces

Access & Resource limits on Namespaces

Can give teams access to only their namespaces
    Each team has its own, isolated environment
Can limit CPU, RAM, Storage per Namespace

Characteristics of Namespaces

Can't access most resources from another namespace

Each Namespace must define its own ConfigMap - even if its it's own service

Can share services across namespaces
    Can access service through "service_name.namespace" (e.g. mysql-service.database)

Some components can't be created within a namespace
    Live globally within a cluster
    Can't isolate them
    Examples: Volume & Node
    Can list components not in namespace: **kubectl api-resources --namespaced=false**

Create component in a Namespace

No Namespace defined - will be added to the "default" namespace

Apply at run-time: kubectl apply -f mysql-configmap.yaml --namespace=my-namespace
Or within the configuration file itself: "metadata.namespace"
    In config file is better since it's documented & easier to apply in automation

Change active namespace (or default namespace)

Not natively possible with Kubernetes, but possible with tool called **kubens**

Kubectx & Kubens

https://github.com/ahmetb/kubectx

Tools to switch between contexts (clusters) and namespaces easily / faster

Ingress

Thursday, June 23, 2022 9:50 AM

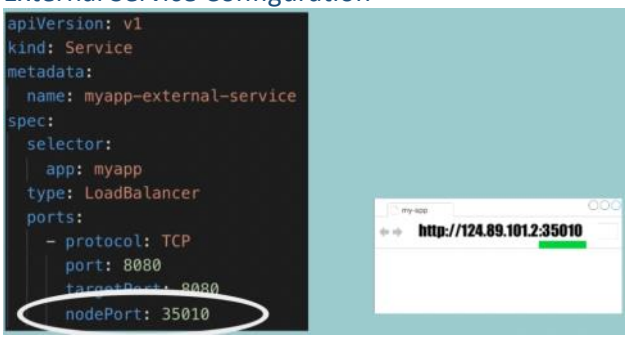
- Git Repo: https://bit.ly/3mJHVFc
- Ingress Controllers: https://bit.ly/32dfHe3
- Ingress Controller Bare Metal: https://bit.ly/3kYdmLB

External Service vs. Ingress

External service would open external IP address and port
Ingress does **not** open external IP address and port
    Instead leverages a domain

Ingress redirects traffic to internal service

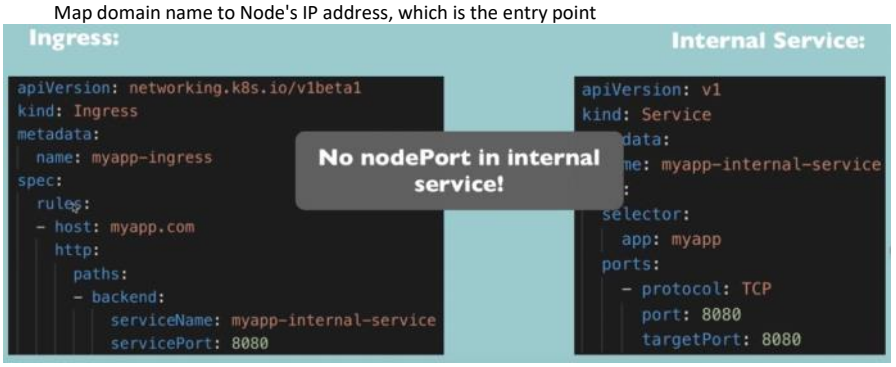
External Service Configuration



Type is "LoadBalancer" & need to configure "nodePort"

Ingress Configuration

"http" key does not correlate to "https"
"host" needs to be:
    Valid domain address
    Map domain name to Node's IP address, which is the entry point



How to configure Ingress

Need an implementation for ingress, which is **ingress controller**

Ingress controller

Another Pod or set of pods that evaluate ingress rules
Evaluate all rules and manage redirections
Entrypoint to cluster
Many third-party implementations to choose from
    e.g. K8s Nginx Ingress Controller

Cloud Environment

Cloud Load Balancer of choice would direct traffic to the ingress controller pod (one of many implementations)
    Advantage of Cloud Load Balancer:

Don't have to implement load balancer yourself

Bare Metal Environment

Need to configure some kind of entrypoint
Either inside or outside as a separate server

Example:

External Proxy Server

No server in K8s cluster is accessible from outside!

Entrypoint forwards to ingress controller pod

Ingress Controller in Minikube

Installing in Minikube

minikube addons enable ingress
Automatically starts the K8s Nginx implementation of Ingress Controller

See Nginx ingress controller running

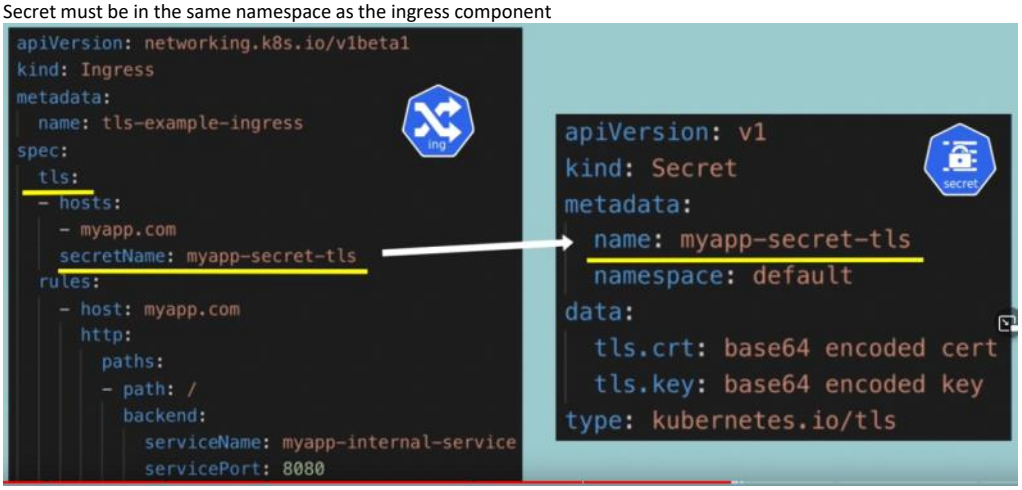
kubectl get pod -n kube-system

See all components in Namespace

kubectl get all -n kubernetes-dashboard

Configuring TLS Certificate

TLS Secret data needs to have tls.crt and tls.key key-value pairs
tls.crt & tls.key are file contents NOT paths/locations
Secret must be in the same namespace as the ingress component



Helm Package Manager

Thursday, June 23, 2022 11:04 AM

What is Helm?

Package manager for Kubernetes
Package YAML Files and Distribute them in public and private repositories

Helm Charts

Bundle of YAML files
Pushed to Helm Repository
helm install CHART NAME

Sharing Helm Charts

Searching from command line: helm search <keyword>
    Or from Helm Hub
There are also private registries for Helm charts

Templating Engine

If deployment & service configs are almost the same:
    You can define a common blueprint
    Leverage dynamic values with placeholders
        values.yaml file
            "{{ .Values.name }}"
One yaml file for many microservices

Use cases

Practical for CI/CD - Build Yaml files & replace on the fly
Same applications across different environments
    Deploy same setup in many environments

Helm Chart Structure

Directory Structure

    mychart/ # Name of the chart
        chart.yaml # Meta info about the chart (name version dependencies)
        values.yaml # Values for the template files (default values)
        charts/ # Chart dependencies (if it depends on other charts)
        templates/ # Where template files are stored
        …
helm install <chartname>
    Template files will be filled with values from values.yaml

Values injection into template files

Default Values File: value.yaml
Define additional values files: helm install --values=my-values.yaml <chartname>
    Any values not defined here, will pull from default value file
On the command line: helm install --set key=value

Release Management

Helm Version 2

Comes in two parts:

Client (helm CLI)
helm install <chartname>
Sends requests to Tiller
Server (Tiller)
Where Kubernetes cluster is installed
Keeps track of all chart executions

Changes are applied to existing deployment instead of creating a new one
Initially install chart: helm install <chartname>
Upgrade chart: helm upgrade <chartname>
Rollback changes: helm rollback <chartname>

Downsides of Tiller

Has too much power inside of K8s Cluster - Big Security Issue
Helm 3 removes Tiller - now a simple helm binary

Volumes / Persistance

Thursday, June 23, 2022 12:36 PM

Three tiers of Persistent Storage

Persistent Volume

physical or logical storage, location can be remote or local
Can also configure what type of access is required (accessMode)
Not associated with a namespace
Examples: AWS EBS, Local Hard disk

Persistent Volume Claim

claim made by developer stating how much storage is required for a pod
Associated with a namespace

Storage Classes

Provisions PV's dynamically when a PVC claims it
Abstracts underlying storage provider
Define the "**provisioner**" attribute for a specific storage provider for where to create PV's
    Each storage backend has it's own provisioner
    Internal Provisioner - "kubernetes.io/" pre-fixed
        Pre-defined provisioners native to Kubernetes
    External Provisioner - provisioners not included in Kubernetes
        Example: NFS
Define the "**parameters**" attribute for PV

Storage Class Usage

Requested with PVC that references storage class instead of persistent volume

1. Pod claims storage via PVC
2. PVC requests storage from StorageClass
3. SC creates PV that meets needs of the claim

[Kubernetes Documentation: https://kubernetes.io/docs/concepts/storage/persistent-volumes/]

ConfigMap & Secret

Local volumes, not created by PV or PVC, created by Kubernetes

Use cases

Need a file available for your pod
    Need configuration file for your pod
    Need certificate mounted for your pod

How it works

1. Create ConfigMap and/or Secret component
2. Mount that into your pod/container (same way as volume)

StatefulSet

Friday, June 24, 2022 11:05 AM

What is a StatefulSet?

K8s component used for Stateful applications
Stateful Applications - any applications that stores data to track state (e.g. databases)
Stateless Applications - don't keep record of state, each request is completely new
Best Practice to leverage Persistent storage along with Stateful applications

Deployments

Stateless Applications

Deployed using Deployment so that you can replacate your app as much as you want

Stateful Application

Deployed using StatefulSet components
Can replicate Pods, both manage pods based on an identical container specification
Configures storage in the same way as Stateless applications

Deployment Stateless vs. StatefulSet

Replicating stateful is more difficult due to more requirements
Stateless pods **can** be:
- Identical and interchangeable
- Created in random order with random hashes
- One Service that load balances to any Pod

Stateful pods **cannot** be:
- Created/deleted at same time
- Randomly addressed
- B/c replica Pods are not identical
  - Requires Pod Identity

Pod Identity with StatefulSet

Every Pod has it's own identifier - **Sticky identity** for each pod
Sticky identity allows each pod to retain state & role
Created from same specification, but not interchangeable
Persistent identifier across any re-scheduling

Why is identity necessary?

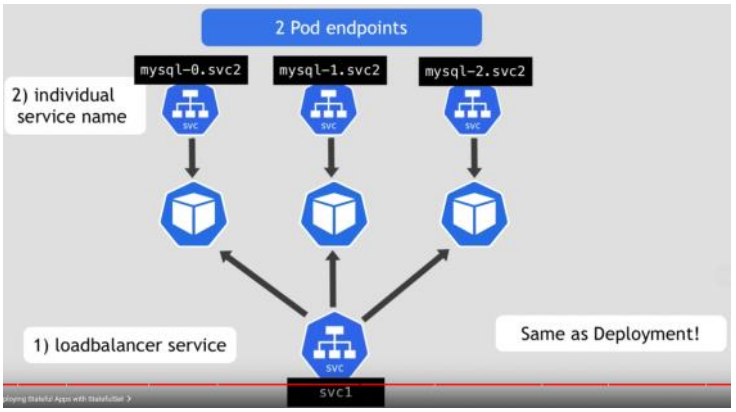
Multiple pods updating (writing) their own database would lead to **data inconsistency**
- Master pod writes, worker/slave pods read

Each of the pods do not use the same physical storage
- The each have a replica of the data that they can access themselves, leading to continuously syncing

Persistent Pod Identity will retain the Persistant storage, as well as Pod state through replacement
- **Important to use Remote Storage - needs to be accessible from all nodes!**

Next Pod is only created if previous pod is up and running!
Delete StatefulSet or scaled down by N replicas, it will occur in the reverse order!

Each pod has it's own DNS Endpoint from a Service
- **Pod DNS name will be ${pod name}.${governing service domain}**
  - **Governing service domain is the service name defined within the StatefulSet**



When a Pod restarts: IP address changes, but name & endpoint stays the same

What you need to manage (not Kubernetes-managed)

Configuring the cloning & data synchronization
Make remote storage available
Managing and back-up

Stateful applications are not perfect for containerized environments

Kubernetes Services

Friday, June 24, 2022 11:58 AM

What is a Service?

Default Behavior without Service

Each pod gets an IP address from Node's range, but these are ephemeral b/c Pods are ephemeral (are destroyed frequently)

- **To see IP address of pod within the cluster:** kubectl get pod -o wide

Service

Creates a stable IP address to access Pod(s)
Configures Loadbalancing
Loose coupling for comm within & outside the cluster
Spans all of the worker nodes

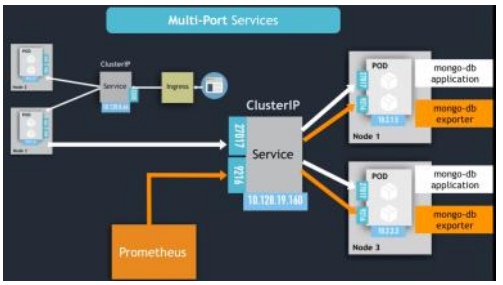
Attributes

Selectors tell the service which Pods to forward request to
- "**selector**" attribute - key-value pairs for the labels of pods identify a set of pods
- Service selector must match

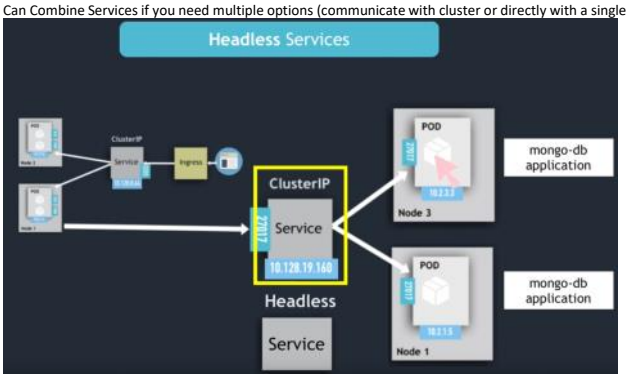
targetPort attribute defines which port to send to on the Pod

When you create a service, K8s creates an **"Endpoint" object** that keeps track of which Pods are the endpoints of the service
- Endpoint object will be the same name as the service
- **Get all endpoints:** kubectl get endpoints

Service's "**port**" attribute is arbitrary, but "**targetPort**" attribute must match the port the container is listening at



Can Combine Services if you need multiple options (communicate with cluster or directly with a single node)



Types of Services

ClusterIP Services

Default type of service
Service gets a static IP address & can receive requests on different ports

Multi-port Service

If you have multiple ports open within a service, you **must** "name" those ports

Head-less Services

Client wants to communicate with 1 specific Pod direcly
Pods want to talk directly with a specific Pod
- Not randomly selected

Use case: Stateful applications, like MySQL / MongoDB / Elasticsearch
Options for communicating with an individual Pod directly:

1. API Call to K8s API Server to get IP addresses (inefficient)
2. DNS Lookup
   a. DNS Lookup for Service - returns single IP address (clusterIP)
   b. Set **ClusterIP** to "**None**" - returns Pod IP address instead

Type Attributes

ClusterIP

Default, type not needed! **Internal** Service, only available from within the cluster

NodePort

External traffic has **access to a fixed port** on each worker node
Extension of the ClusterIP Service - Creates ClusterIP Service automatically
Defined in the "nodePort" attribute
Pre-defined range: 30000 - 32767
NodePort Servces are not secure
NOT FOR PRODUCTION - Either configure Ingress or LoadBalancer for production environments

LoadBalancer

Service accessible **externally** through **cloud providers LoadBalancer**
Extension of the NodePort Service - NodePort & ClusterIP Service are created automatically
More secure since only one port is exposed to the LoadBalancer rather than outside