K8s

1. What is K8s

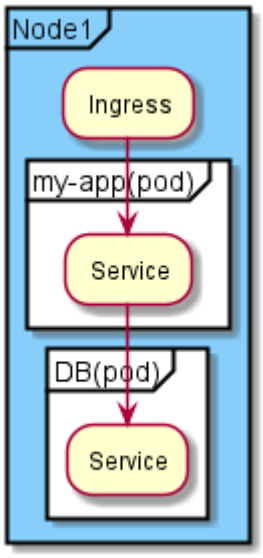
2. Main K8s Components

2.1. Nodes and Pods

- Node - physical or virtual machine.
- Smalles unit of k8s is pod. Pod is abstraction over container. Usually 1 application per Pod. Pods in k8s commuticates via virtual k8s network. Echa Pod gets its own internal IP address.

2.2. Service and Ingress

- Service - permanent IP address that can be attached to to each Pod.
- Lifecycle of Pod and Service NOT connected, if Pod dies the service and its ip address will stay, so you don’t have to change that end point any more.
- Ingress. Application needs to be acced from outside and for this you would have to create an external service. External service is the service that opens the communication from external sources. The request goes to Ingress and then it does the forwarding then to the service.



2.3. ConfigMap and Secret

- ConfigMap is external configuration of your application(url of a database or some other services, db username and password is not secure place to keep in ConfigMap). if we change endpoint of the servcice we just adjust the ConfigMap
- Secret is just like ConfigMap, but the difference is that it is used to store secret data, credentials for example. The information is in base64 encoded format

2.4. Volumes

If DB container gets restarted the data would be gone. Volumes attaches a physical storage on a hard drive to Pod. That storage can be either on a local machine, meaning on the same server node where the pod is running or it could be on a remote storage, meaning outside of the k8s cluster, it could be a cloud storage or your own premise storage which is not part of k8s cluster.

2.5. Deployment and Stateful Set

3. K8s Architecture

3.1. Two type of nodes

- Master
- Slave

```
- Each node has multiple Pods on it.
- Three processes must be installed on every node
```

- container runtime *

```
- Worker Nodes do the actual work
```

Each node has multiple Pods on it
Three processes must be installed on every node
- container runtime
- kubelet.footnote:
- kube proxy(3)

Worker Nodes do the actual work

A statement.[1]

A bold statement![2]

Another bold statement.[2]

```
(1) Kubelete - interacts with both - the container and node, Kubelet starts the pod
with a container inside, and then assigning resources from that node to the container
like cpu, ram and storage resources.
```

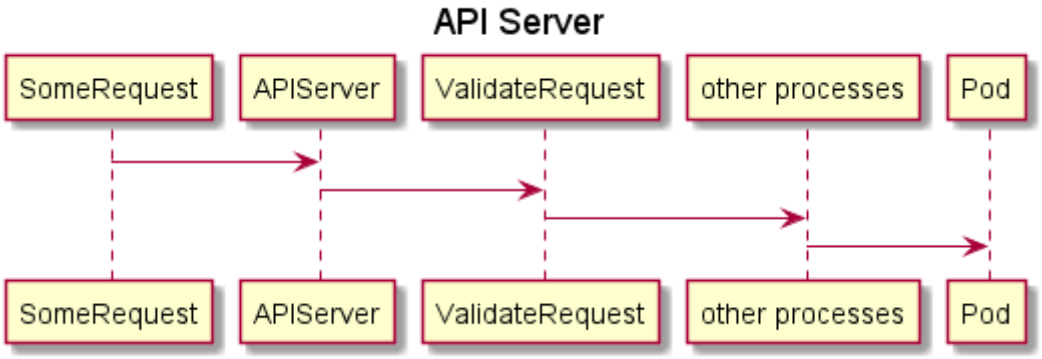
```
(2) Servlet is a sort of loadbalancer that basically caches the request directed to
the pod or the application and the forwads it to the repective pod
```

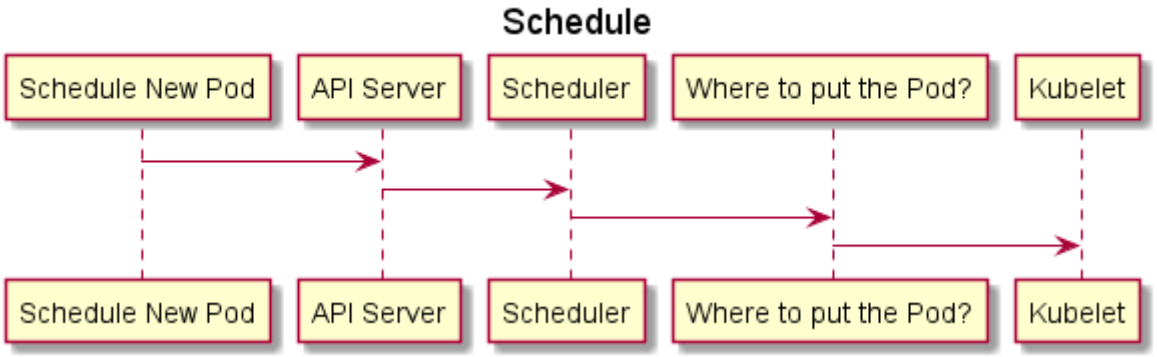
```
(3) Kube proxy forwards the requests
```

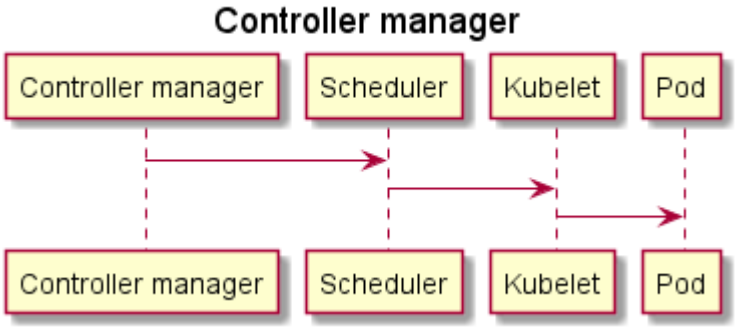
How do you interact with this cluster
- How to:
  - schedule pod?
  - monitor?
  - re-schedule/re-start pod?
  - join a new Node?

All these managing processes are done by **Master Nodes**

Master processes
- Api server(cluster gateway)
- Scheduler
- Controller manager
- etcd(key value store of a cluster state. it is a cluster brain)
  - Is the cluster healthy?
  - What resources are available?
  - Did the cluster state change?







3.2. Example cluster set-up

4. Minikube and kubectl - Local Setup

4.1. Master

4.1.1. Install

```
hostnamectl set-hostname master.art.local
echo "192.168.56.10 master.art.local master" >> /etc/hosts
echo "192.168.56.11 node1.art.local node1" >> /etc/hosts
echo "192.168.56.12 node2.art.local node2" >> /etc/hosts
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=192.168.56.10 --flannel
-iface=enp0s8" sh -
```

4.1.2. Debug

```
systemctl status k3s
journalctl -f --unit k3s
watch -n3 kubectl get nodes
```

4.1.3. Config file

```
cat /etc/rancher/k3s/k3s.yaml
chmod 644 /etc/rancher/k3s/k3s.yaml
```

4.1.4. Token

```
cat /var/lib/rancher/k3s/server/token
```

4.1.5. Uninstall

```
/usr/local/bin/k3s-uninstall.sh
```

4.2. Node

4.2.1. Install

```
hostnamectl set-hostname node1.art.local node1
echo "192.168.56.10 master.art.local master" >> /etc/hosts
echo "192.168.56.11 node1.art.local node1" >> /etc/hosts
echo "192.168.56.12 node2.art.local node2" >> /etc/hosts
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip=192.168.56.11 --flannel
-iface=enp0s8" K3S_URL="https://192.168.56.10:6443"
K3S_TOKEN="K108ada0b700e91cf2201586cedb9e0f26b7a5a7923fb551affb7aca393e03630c5::server
:8568fd7b7e57846f03294ea4f6a3de00" sh -
```

4.2.2. Install specifi K3s version

K3s versions

```
curl -sfL https://get.k3s.io | INSTALL_K3S_VERSION=v1.18.12+k3s1 sh -
```

4.2.3. Debug

```
journalctl -f --unit k3s-agent
```

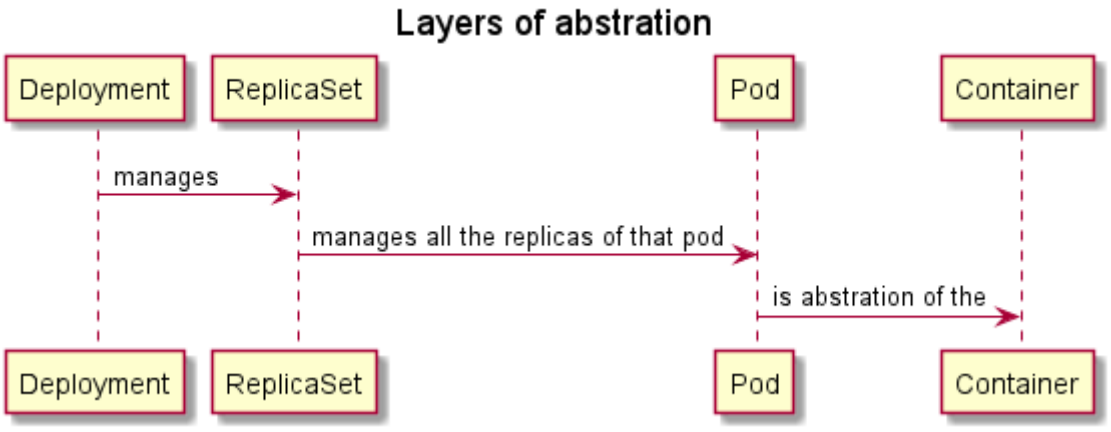
4.2.4. Uninstall

```
/usr/local/bin/k3s-agent-uninstall.sh
```

5. Main Kubectl Commands - K8s CLI

- CRUD commands
  - Create deployment
    - kubectl create deployment [name]
  - Edit deployment
    - kubectl edit deployment [name]
  - Delete deployment
    - kubectl delete deployment [name]
  - etcd(key value store of a cluster state. it is a cluster brain)
- Status of different K8s components
  - kubectl get
    - nodes
    - pod
    - services
    - replicaset
    - deployment
- Debbuging pods
  - Log to console
    - kubectl logs [pod name]
  - Get Interacitve terminal
    - kubectl exec -it [pod name] --bin/bash

```
kubectl create -h
kubectl create deployment nginx-depl --image=nginx
kubectl get deployment
kubectl get pod
kubectl get replicaset
```




```
kubectl edit deployment nginx-depl
- change image to nginx:1.16
kubectl get pod
kubectl get replicaset
```

5.1. Debugging pods

```
kubectl get pods
kubectl logs nginx-depl-7fc44fc5d4-pc9wr
kubectl create deployment mongo-depl --image=mongo
kubectl get pods
kubectl logs mongo-depl-5fd6b7d4b4-lr6dv
kubectl describe pod mongo-depl-5fd6b7d4b4-lr6dv
```

```
kubectl get pods
kubectl exec -it mongo-depl-5fd6b7d4b4-lr6dv -- bin/bash
```

5.2. Delete deployment

```
kubectl get deployment
kubectl delete deployment mongo-depl
kubectl get pod
kubectl get replicaset
```

5.3. Apply configuration file

k8s/nginx-deployment.yaml

```
apiVersion: apps/v1
# what I want to create
kind: Deployment
metadata:
  # the name of the deployment
  name: nginx-deployment
  labels:
    # this is used by the service selector
    app: nginx
spec:
  # how many replicas of the pods I want to create
  replicas: 1
  # specification for the deployment
  selector:
    matchLabels:
      app: nginx
  # specification for the pod. bluprint for the pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        # we want one container inside the pod
        - name: nginx
          image: nginx:1.16
          ports:
            - containerPort: 80
```

```
kubectl apply -f nginx-deployment.yaml
kubectl get deployment
kubectl get pod
kubectl describe service nginx-service
kubectl get pods -o wide
```

k8s/nginx-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    # makes a connection between the service and the deployment->metadata->labels->app
or deployment->spec->template->metadata->label->app
    app: nginx
  ports:
    - protocol: TCP
      # on this port the service is accessible
      port: 80
      # connects to deployment->spec->template->spec->containerPort
      targetPort: 8080
```

```
kubectl apply -f nginx-service.yaml
kubectl get service
```

5.4. Delete components using configuration files

```
kubectl delete -f nginx-deployment.yaml
kubectl delete -f nginx-service.yaml
```

6. K8s YAML Configuration File

6.1. The 3 parts of configuration file

Each K8s configuration file has 3 parts

- Metatada
- Specification
- Status - is not in the file it will be automatically generated and added by K8s. In file is "Desired status", K8s generates "Actual status". If Desired and Actual does not match then K8s knows that somthing to be fixed there, so it is going to fix it. It is the self-healing feature K8s provides. This information comes from etcd.

k8s/deployment-structure.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels: ...
# Attributes of "spec" are specific to the kind!
spec:
  replicas: 1
  selector: ...
  template: ...
```

k8s/service-structure.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
# Attributes of "spec" are specific to the kind!
spec:
  selector: ...
  ports: ...
```

status is get using

```
kubectl get deployment nginx-deployment -o yaml
kubectl get deployment nginx-deployment -o yaml > nginx-deployment-result.txt
```

6.2. Format of configuration file

YAML

7. Complete Application Setup with Kubernetes Compoenents

7.1. Demo Project: MongoDB and MongoExpress

7.1.1. mongoDB

- Deployment / Pod
- Service
- ConfigMap

- Secret

7.1.2. mongo-express

7.1.2.1. ConfigMap

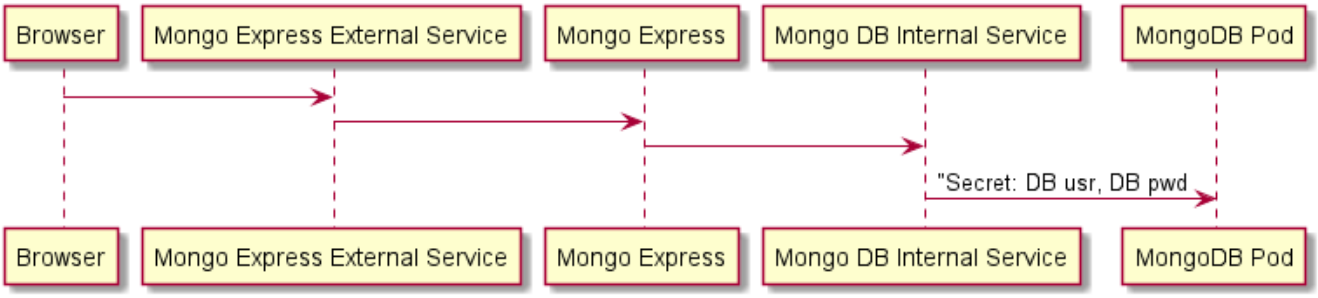
- mongoDB url

7.1.2.2. Secret

- DB user
- DB password

7.1.2.3. Accessible through browser

- in order to do that we will create external service



7.2. Lets create

7.2.1. Mongo DB Internal service

Mongo on docker hub

7.2.1.1. Create a Secret

k8s/mongo-db-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  # echo -n 'username' | base64
  mongo-root-username: dXNlcm5hbWU=
  # echo -n 'password' | base64
  mongo-root-password: cGFzc3dvcmQ=
```

```
echo -n 'username' | base64
echo -n 'password' | base64
```

```
kubectl apply -f mongo-db-secret.yaml
kubectl get secret
kubectl describe secret mongodb-secret
```

7.2.1.2. Create Deployment

k8s/mongo-db-deployment.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
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-username
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-password
```

```
kubectl apply -f mongo-db-deployment.yaml
kubectl get deployment
kubectl get pod --watch
```

7.2.1.3. Create Service

k8s/mongo-db-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    # we want this service connect to the pod
    app: mongodb
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

```
kubectl apply -f mongo-db-service.yaml
kubectl get service
kubectl get pod -o wide
kubectl get all | grep mongodb
```

https://youtu.be/X48VuDVv0do?t=5604

7.3. Mongo Express

7.3.1. Mongo Express ConfigMap

k8s/mongo-express-configmap.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  # just a service name from mongo-db-service.yaml
  dabase_url: mongodb-service
```

```
kubectl apply -f mongo-express-configmap.yaml
kubectl describe configmap mongodb-configmap
```

7.3.2. Mongo Express Deployment

Mongo-express on docker hub

k8s/mongo-express-deployment.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
              valueFrom:
                configMapKeyRef:
                  name: mongodb-configmap
                  key: dabase_url
```

```
kubectl apply -f mongo-express-deployment.yaml
```

7.3.3. Mongo Express External Service

k8s/mongo-express-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  # this makes the service external
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      # port for external IP address. Range 30000-32000
      nodePort: 30000
```

```
kubectl apply -f mongo-express-service.yaml
kubectl get service
```

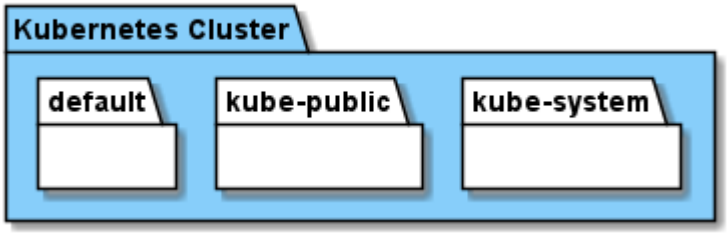
http://192.168.56.10:30000/

8. Organizing your components with K8s Namespaces

Namespace cluster inside a cluster

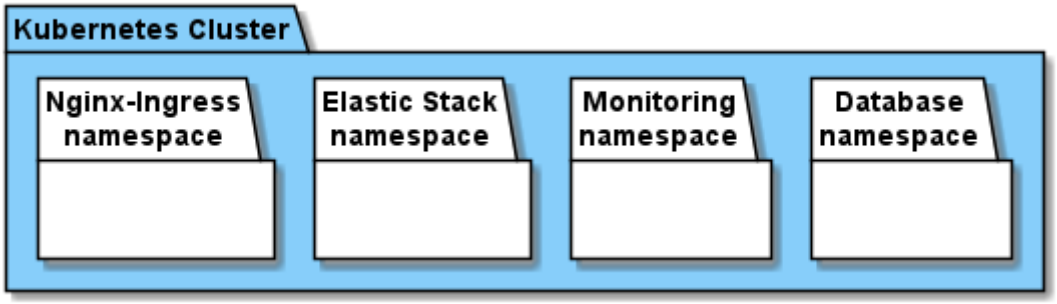
- K8s gives 4 namespaces out of the box
  - **kube-system** is not meant for your own use
  - **kube-public** - it has ConfigMap
  - **kube-node-lease** - it holds information about heartbeats of nodes
  - **default** - creates resources at the begining

```
kubectl get namespaces
kubectl create namespace my-namespace
kubectl get namespace
```

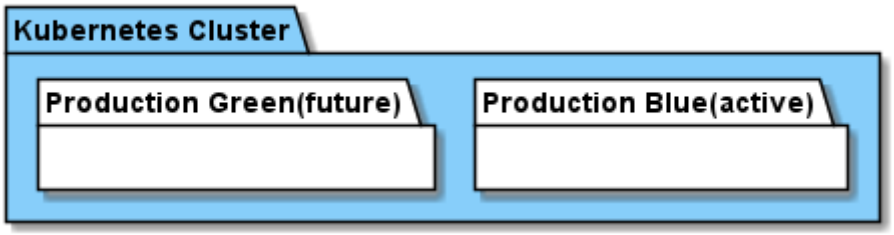


What is the need for namespaces?

- Logically group resources inside cluster.



- Conflicts: Many teams, same application
- Rsource Sharing: Staging and Development
- Rsource Sharing: Blue/Green Deployment



- Access and Resource Limits on Namespaces - for example we have two teams with different namespaces with separate resources on namespaces

  Use cases when to use Namespaces

  1. **Structure** components
  2. **Avoid conflicts** between teams
  3. **Share services** between different evironments
  4. **Access and Resource Limits** on Namespaces Level

  Characteristics of Namespaces?

  1. You can't access most resources from another Namespace
  2. Components, which can't be created within a Namespace: volume, node

Create components in Namespaces?

- First way

```
kubectl apply -f mysql-configmap.yaml --namespace=my-namespace
```

- Second in configuration file

```
metada.namespace: my-namepace
```

Change active namespace

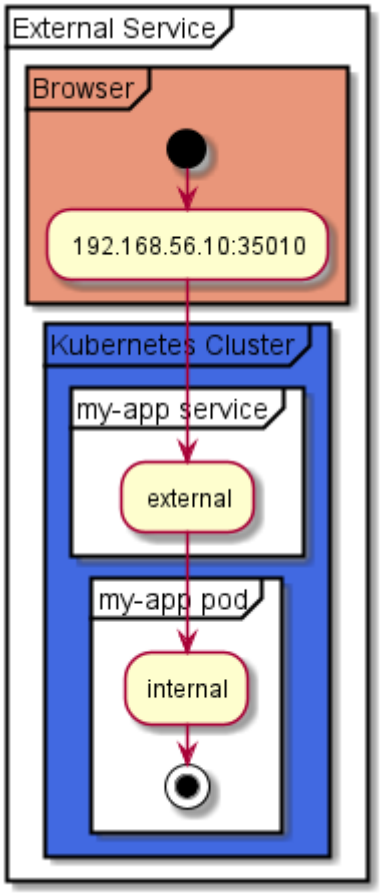
```
Tool kubens
```

9. K8s Ingress explained

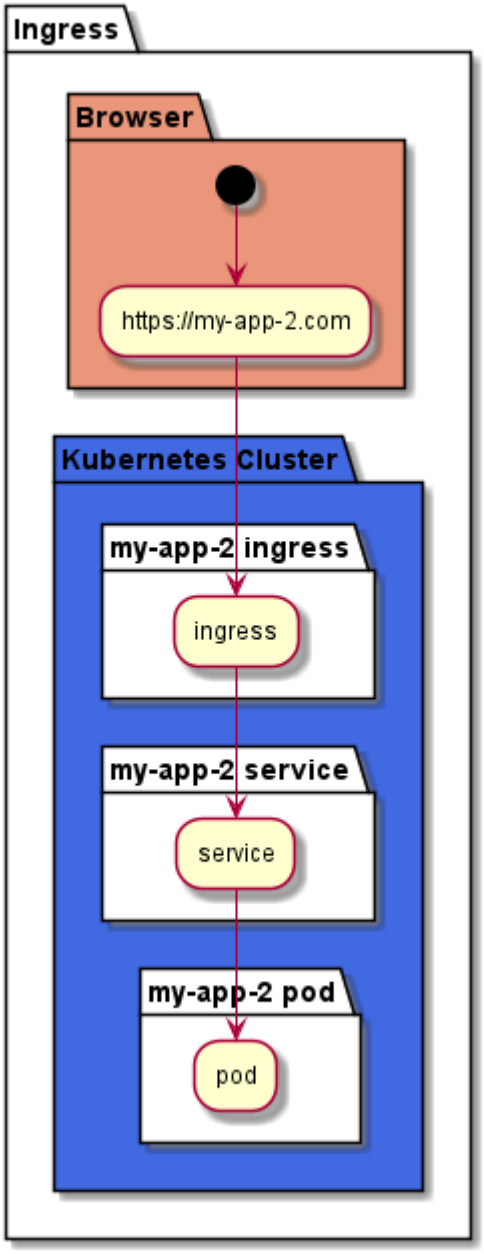
9.1. What is Ingress?

9.1.1. External service vs. Ingress

- External service good for test cases



- Ingress IP address and ports is not opened!



9.2. Ingress YAML configuration

- Internal service

k8s/myapp-external-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-external-service
spec:
  selector:
    app: myapp
  # Assign external IP to service
  type: LoadBalancer
  ports:
    - port: 8080
      targetPort: 8080
      # External port that user can access application at
      nodePort: 35010
      protocol: TCP
```

- Ingress

k8s/myapp-ingress.yaml

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

9.3. How to configure Ingress?

Ingress YAML file is not enough for Ingress routing rules to work, we need in addition is an implementation for Ingress! Implementation of Ingress is called Ingress Controller. Ingress controller is another Pod or set of Pods which evaluates, processes, manages redirections Ingress rules. Ingress controller is the entry point in the cluster for all the requests to that domain subdomain rules.

There are many third-party implementations. Ingress Controllers, Bare metal

9.3.1. Configure Ingress

TODO

```
GITHUB_URL=https://github.com/kubernetes/dashboard/releases

VERSION_KUBE_DASHBOARD=$(curl -w '%{url_effective}' -I -L -s -S ${GITHUB_URL}/latest
-o /dev/null | sed -e 's|.*/||')

k3s kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/${VERSION_KUBE_DASHBOARD}/aio/d
eploy/recommended.yaml
```

```
cat <<EOF > dashboard.admin-user.yml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
EOF
```

```
cat <<EOF > dashboard.admin-user-role.yml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: admin-user
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: cluster-admin
subjects:
- kind: ServiceAccount
name: admin-user
namespace: kubernetes-dashboard
EOF
```

```
k3s kubectl create -f dashboard.admin-user.yml -f dashboard.admin-user-role.yml
```

k8s/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
              # execute "kubectl get all -n kubernetes-dashboard" to get the port
              servicePort: 80
```

TODO

```
kubectl apply -f dashboard-ingress.yaml
# wait for ADDRESS to be assigned by K8s
watch kubectl get ingress -n kubernetes-dashboard
echo "10.0.2.15 dashboard.com" >> /etc/hosts
ping dashboard.com
curl dashboard.com
kubectl describe ingress dashboard-ingress -n kubernetes-dashboard
kubectl get ingress dashboard-ingress -n kubernetes-dashboard
```

```
kubectl -n kubernetes-dashboard edit svc kubernetes-dashboard
# change spec->type->ClusterIP to spec->type->NodePort
# add spec->ports->nodePort=32323
k3s kubectl -n kubernetes-dashboard describe secret admin-user-token | grep ^token
https://192.168.56.10:32323/
```

10. Helm - Package Manager

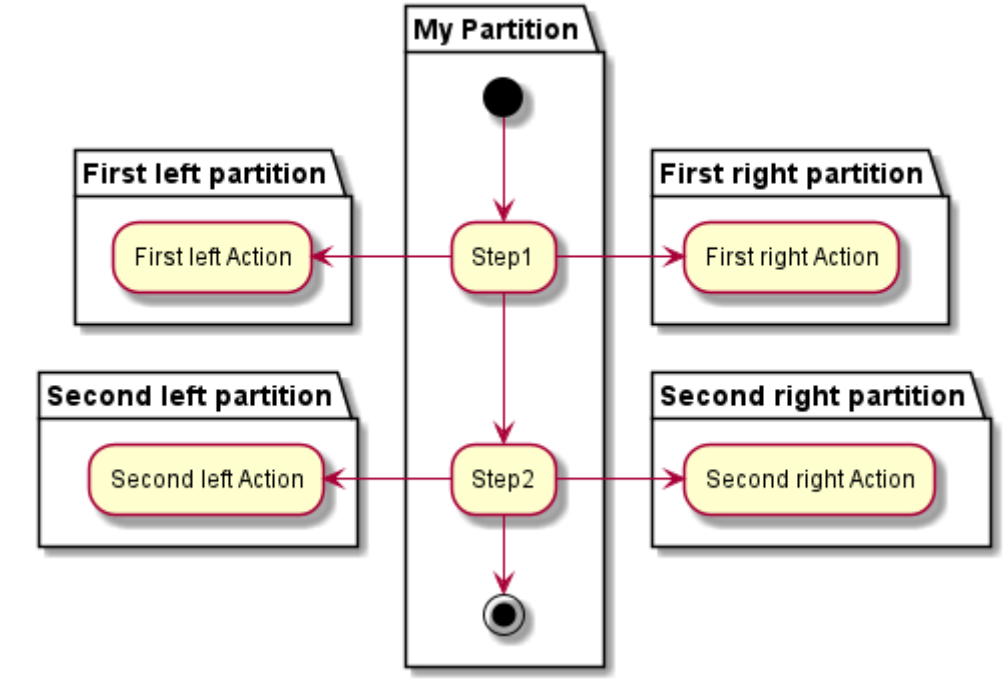
package manager for K8s to package yaml files, like yum, apt

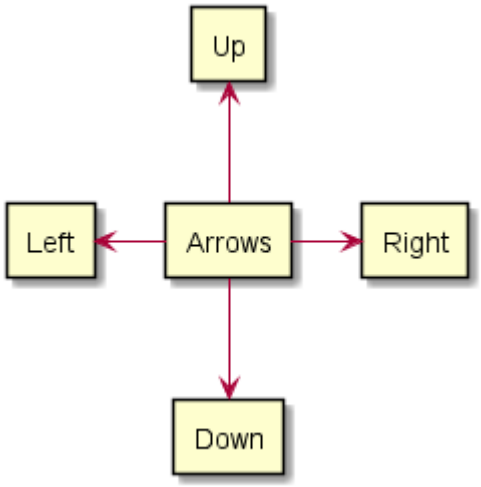
TODO

11. Persisting Data in K8s with Volumes

12. Deploying Stateful Apps with StatefulSet

13. K8s Services explained





[1] (1) Kubelete - interacts with both - the container and node, Kubelet starts the pod with a container inside, and then assigning resources from that node to the container like cpu, ram and storage resources.

[2] Opinions are my own.