useful links

- links collections
- git source
- architecture
- cheat sheet
- docs task by section
- troubleshooting
- java application template generator
- kubernetes patterns

tutorials

- main tutorial
- tutorial
- tutorial external
- interactive course
- interactive course
- interactive course helm

playground and examples

- k8s examples
- playground
- playground
- ibmcloud

tools

- cli manager
- smart shell for kubectl
- log collector
- prompt for K8S
- realtime changes in cluster

workplace installation

- kubectl installation

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/v1.17.4/bin/linux/amd64/kubectl
curl -LO https://storage.googleapis.com/kubernetes-
release/release/v1.18.0/bin/linux/amd64/kubectl
```

- kubectl autocompletion

```
source <(kubectl completion bash)
# source <(kubectl completion zsh)
```

or

```
# source /usr/share/bash-completion/bash_completion
kubectl completion bash >/etc/bash_completion.d/kubectl
```

- trace logging

```
rm -rf ~/.kube/cache
kubectl get pods -v=6
kubectl get pods -v=7
kubectl get pods -v=8
# with specific context file from ~/.kube, specific config
kubectl --kubeconfig=config-rancher  get pods -v=8
```

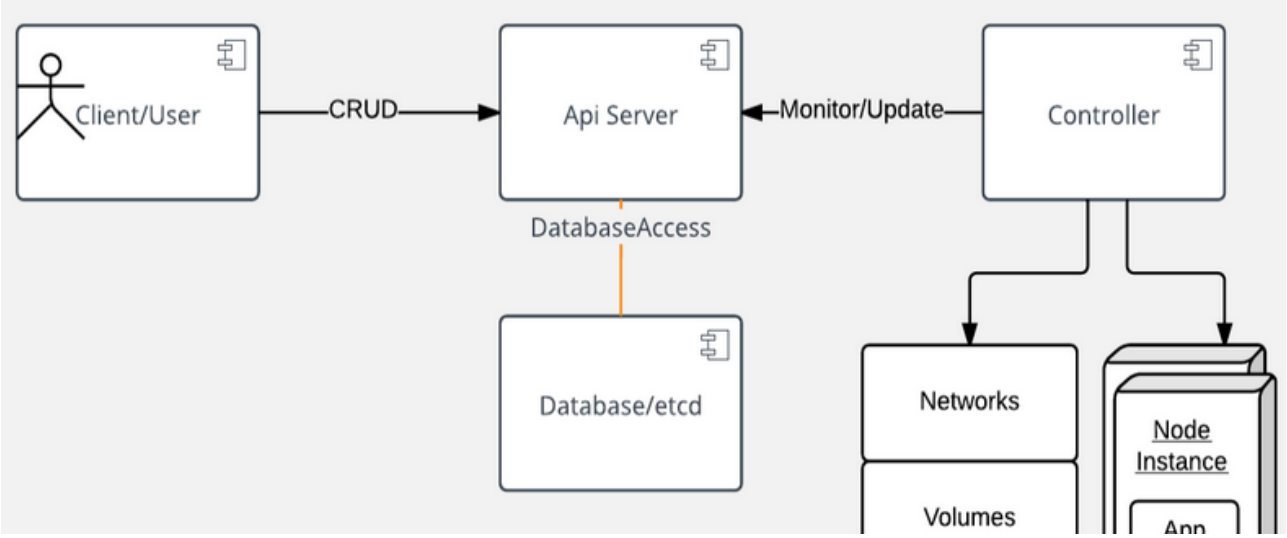
- explain yaml schema

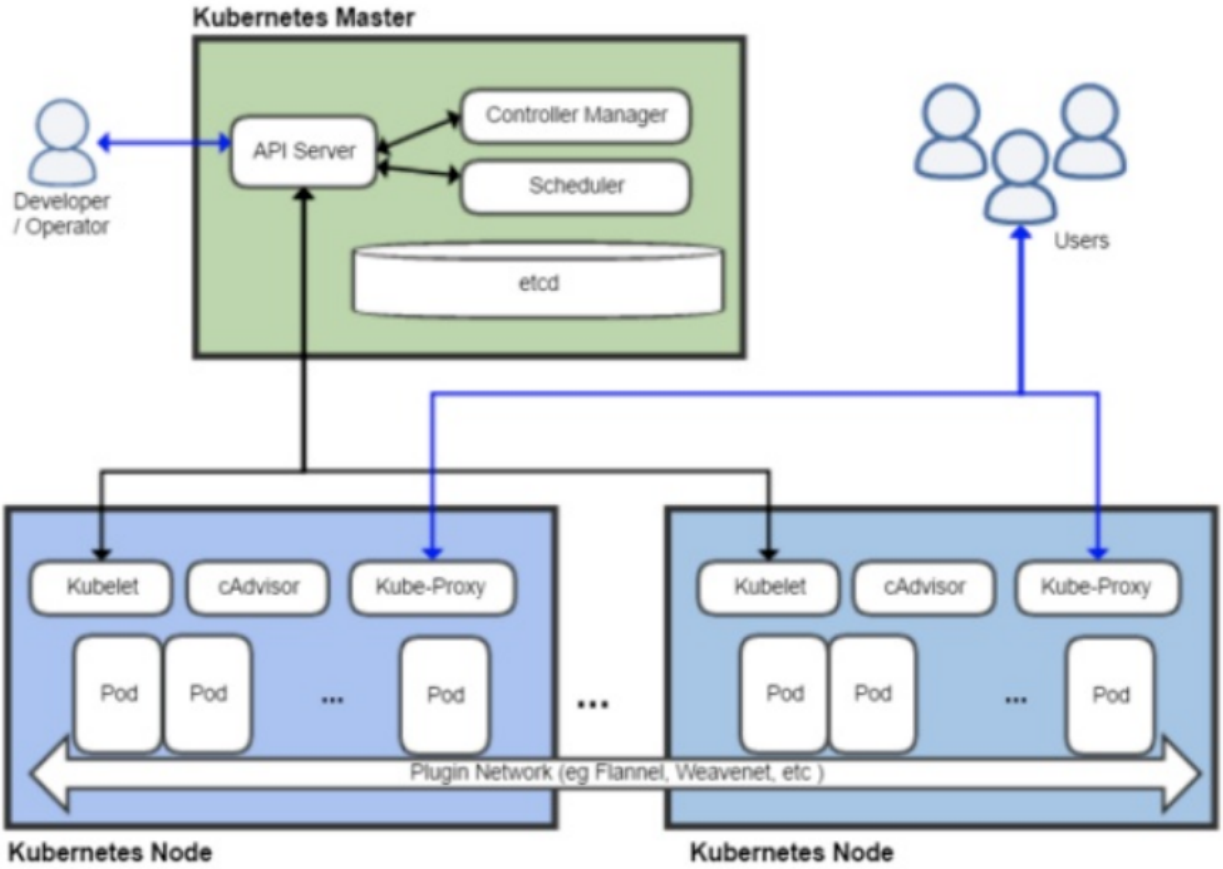
```
kubectl explain pods
kubectl explain pods --recursive
kubectl explain pods --recursive --api-version=autoscaling/v2beta1
```

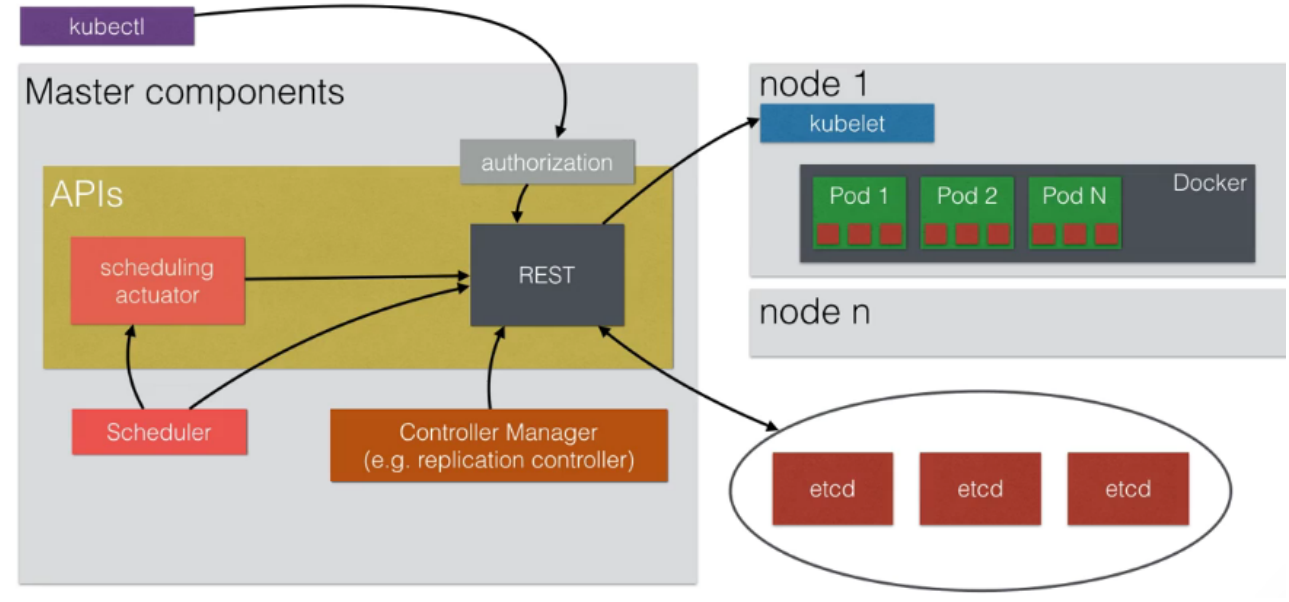
- python client

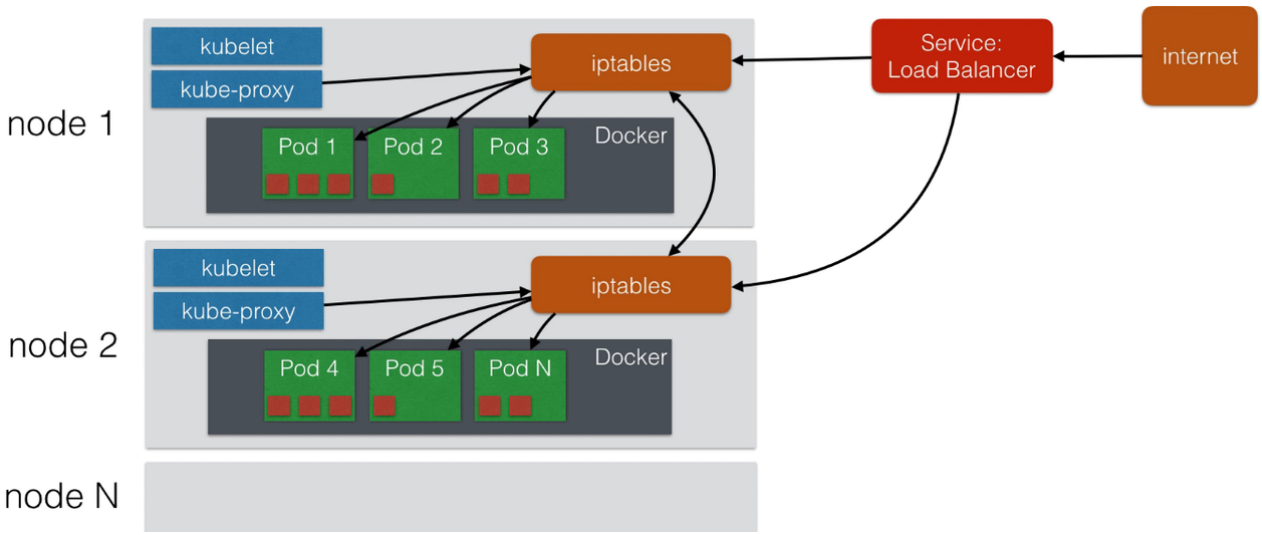
```
pip install kubernetes
```

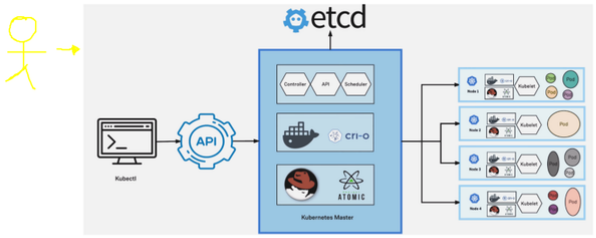
Architecture

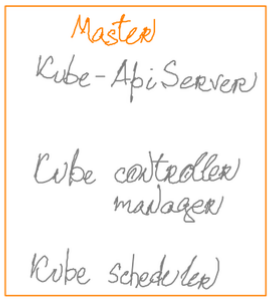

Client/User
CRUD
Api Server
Monitor/Update
Controller
DatabaseAccess
Database/etcd
Networks
Volumes
Node
Instance

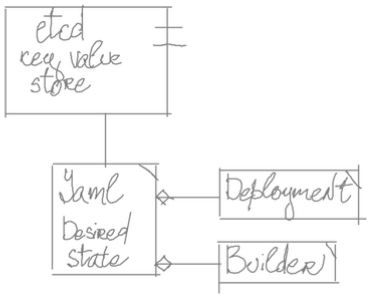

Kubernetes Master
Developer
/ Operator
API Server
Controller Manager
Scheduler
etcd
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node


kubectl
Master components
APIs
authorization
scheduling actuator
REST
Scheduler
Controller Manager (e.g. replication controller)
node 1
kubelet
Pod 1
Pod 2
Pod N
Docker
node n
etcd
etcd
etcd

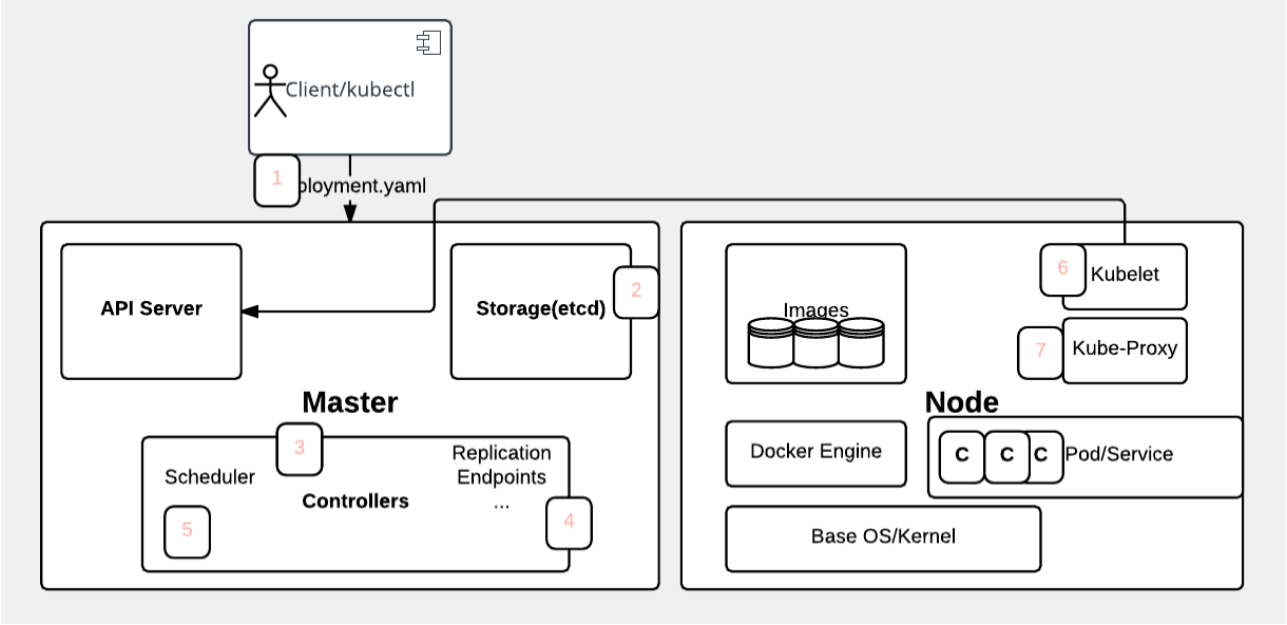

node 1
kubelet
kube-proxy
iptables
Pod 1
Pod 2
Pod 3
Docker
Service: Load Balancer
internet
node 2
kubelet
kube-proxy
iptables
Pod 4
Pod 5
Pod N
Docker
node N


etcd
API


Master
Kube-Api Server
Kube controller manager
Kube scheduler


etcd
key value store
yaml
Desired state
Deployment
Builder

workflow



1. The user deploys a new app by using the kubectl CLI. Kubectl sends the request to the API server.
2. The API server receives the request and stores it in the data store (etcd). After the request is written to the data store, the API server is done with the request.
3. Watchers detect the resource changes and send notifications to the Controller to act on those changes.
4. The Controller detects the new app and creates new pods to match the desired number of instances. Any changes to the stored model will be used to create or delete pods.
5. The Scheduler assigns new pods to a node based on specific criteria. The Scheduler decides on whether to run pods on specific nodes in the cluster. The Scheduler modifies the model with the node information.
6. A Kubelet on a node detects a pod with an assignment to itself and deploys the requested containers through the container runtime, for example, Docker. Each node watches the storage to see what pods it is assigned to run. The node takes necessary actions on the resources assigned to it such as to create or delete pods.
7. Kubeproxy manages network traffic for the pods, including service discovery and load balancing. Kubeproxy is responsible for communication between pods that want to interact.

---

k3s - Lightweight Kubernetes

---

microk8s

installation

- https://github.com/ubuntu/microk8s
- https://microk8s.io/

```
sudo snap install microk8s --classic
sudo snap install microk8s --classic --edge
```

enable addons

```
microk8s.start
microk8s.enable dns dashboard
```

check installation

```
microk8s.inspect
```

check journals for services

```
journalctl -u snap.microk8s.daemon-docker
```

- snap.microk8s.daemon-apiserver
- snap.microk8s.daemon-controller-manager
- snap.microk8s.daemon-scheduler
- snap.microk8s.daemon-kubelet
- snap.microk8s.daemon-proxy
- snap.microk8s.daemon-docker
- snap.microk8s.daemon-etcd

---

minikube

installation

```
sudo snap install minikube
```

installation

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 &&
chmod +x minikube
curl -Lo kubectl https://storage.googleapis.com/kubernetes-release/release/$(curl
-s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl && chmod +x kubectl

export MINIKUBE_WANTUPDATENOTIFICATION=false
export MINIKUBE_WANTREPORTERRORPROMPT=false
export MINIKUBE_HOME=$HOME
export CHANGE_MINIKUBE_NONE_USER=true
mkdir $HOME/.kube || true
touch $HOME/.kube/config

export KUBECONFIG=$HOME/.kube/config
sudo -E ./minikube start --vm-driver=none

# wait that Minikube has created
for i in {1..150}; do # timeout for 5 minutes
   ./kubectl get po &> /dev/null
   if [ $? -ne 1 ]; then
      break
  fi
  sleep 2
done
```

set up env

```
minikube completion bash
```

start

```
minikube start
```

uninstall kube, uninstall kubectl, uninstall minikube

```
kubectl delete node --all
kubectl delete pods --all
kubectl stop
kubectl delete

launchctl stop '*kubelet*.mount'
launchctl stop localkube.service
launchctl disable localkube.service

sudo kubeadm reset
## network cleaining up
# sudo ip link del cni0
# sudo ip link del flannel.1
# sudo systemctl restart network

rm -rf ~/.kube ~/.minikube
sudo rm -rf /usr/local/bin/localkube /usr/local/bin/minikube
sudo rm -rf /etc/kubernetes/

# sudo apt-get purge kubeadm kubectl kubelet kubernetes-cni kube*
sudo apt-get purge kube*
sudo apt-get autoremove

docker system prune -af --volumes
```

start without VirtualBox/KVM

```
export MINIKUBE_WANTUPDATENOTIFICATION=false
export MINIKUBE_WANTREPORTERRORPROMPT=false
export MINIKUBE_HOME=$HOME
export CHANGE_MINIKUBE_NONE_USER=true

export KUBECONFIG=$HOME/.kube/config
sudo -E minikube start --vm-driver=none
```

kubectl using minikube context

permanently

```
kubectl config use-context minikube
```

temporary

```
kubectl get pods --context=minikube
```

example of started kube processes

```
/usr/bin/kubelet
    --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
    --kubeconfig=/etc/kubernetes/kubelet.conf
    --config=/var/lib/kubelet/config.yaml
    --cgroup-driver=cgroupfs
    --cni-bin-dir=/opt/cni/bin
    --cni-conf-dir=/etc/cni/net.d
    --network-plugin=cni
    --resolv-conf=/run/systemd/resolve/resolv.conf
    --feature-gates=DevicePlugins=true

kube-apiserver
    --authorization-mode=Node,RBAC
    --advertise-address=10.143.226.20
    --allow-privileged=true
    --client-ca-file=/etc/kubernetes/pki/ca.crt
    --disable-admission-plugins=PersistentVolumeLabel
    --enable-admission-plugins=NodeRestriction
    --enable-bootstrap-token-auth=true
    --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt --etcd-
keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-
servers=https://127.0.0.1:2379 --insecure-port=0 --kubelet-client-
certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubelet-client-
key=/etc/kubernetes/pki/apiserver-kubelet-client.key --kubelet-preferred-address-
types=InternalIP,ExternalIP,Hostname --proxy-client-cert-
file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-
file=/etc/kubernetes/pki/front-proxy-client.key --requestheader-allowed-
names=front-proxy-client --requestheader-client-ca-
file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-
prefix=X-Remote-Extra- --requestheader-group-headers=X-Remote-Group --
requestheader-username-headers=X-Remote-User --secure-port=6443 --service-
account-key-file=/etc/kubernetes/pki/sa.pub --service-cluster-ip-
range=10.96.0.0/12 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls-
private-key-file=/etc/kubernetes/pki/apiserver.key
```

```
kube-controller-manager
    --address=127.0.0.1
    --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    --controllers=*,bootstrapsigner,tokencleaner
    --kubeconfig=/etc/kubernetes/controller-manager.conf
    --leader-elect=true
    --root-ca-file=/etc/kubernetes/pki/ca.crt
    --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    --use-service-account-credentials=true

etcd
    --advertise-client-urls=https://127.0.0.1:2379
    --cert-file=/etc/kubernetes/pki/etcd/server.crt
    --client-cert-auth=true
    --data-dir=/var/lib/etcd
    --initial-advertise-peer-urls=https://127.0.0.1:2380
    --initial-cluster=gtxmachine0=https://127.0.0.1:2380
    --key-file=/etc/kubernetes/pki/etcd/server.key
    --listen-client-urls=https://127.0.0.1:2379
    --listen-peer-urls=https://127.0.0.1:2380
    --name=gtxmachine0
    --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    --peer-client-cert-auth=true
    --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    --snapshot-count=10000
    --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt

kube-scheduler
    --address=127.0.0.1
    --kubeconfig=/etc/kubernetes/scheduler.conf
    --leader-elect=true

/usr/local/bin/kube-proxy
    --config=/var/lib/kube-proxy/config.conf

/opt/bin/flanneld
    --ip-masq
    --kube-subnet-mgr
```

kubectl using different config file, kubectl config, different config kubectl

```
kubectl --kubeconfig=/home/user/.kube/config-student1 get pods
```

kubectl config with rancher, rancher with kubectl, rancher kubectl config

- certificate-authority-data - from admin account
- token - Bearer Token

```
apiVersion: v1
clusters:
- cluster:
    server: "https://10.14.22.20:9443/k8s/clusters/c-7w47z"
    certificate-authority-data: "....tLUVORCBDRVJUSUZJQ0FURS0tLS0t"
  name: "ev-cluster"

contexts:
- context:
    user: "ev-user"
    cluster: "ev-cluster"
  name: "ev-context"

current-context: "ev-context"

kind: Config
preferences: {}
users:
- name: "ev-user"
  user:
    token: "token-6g4gv:lq4wbw4lmwtxkblmbbsbd7hc5j56v2ssjvfkxd"
```

install on ubuntu, install ubuntu, installation ubuntu, ubuntu installation

update software

```
# check accessible list
sudo apt list | grep kube
# update system info
sudo apt-get update
# install one package
sudo apt-get install -y kubeadm=1.18.2-00
```

```
# FROM ubuntu:18
# environment
sudo apt install docker.io
sudo systemctl enable docker
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
sudo apt install curl
# kube
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
sudo apt install kubeadm
sudo swapoff -a
# init for using flannel ( check inside kube-flannel.yaml section net-conf.json/Netw
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
rm -rf $HOME/.kube
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
# !!! install flannel ( or weave.... )
# kubectl get nodes
```

install via rancher

```
docker stop rancher
docker rm rancher

# -it --entrypoint="/bin/bash" \
docker run -d --restart=unless-stopped \
  --name rancher \
  -v /var/lib/rancher:/var/lib/rancher \
  -v /var/lib/rancher-log:/var/log \
  -p 9080:80 -p 9443:443 \
  -e HTTP_PROXY="http://qqml:mlfu\$1@proxy.muc:8080" \
  -e HTTPS_PROXY="http://qqml:mlfu\$1@proxy.muc:8080" \
  -e NO_PROXY="localhost,127.0.0.1,127.0.0.0/8,10.0.0.0/8,192.168.0.0/16" \
  rancher/rancher:latest
```

uninstall

cleanup node

```
# clean up for worker
## !!! most important !!!
sudo rm -rf /etc/cni/net.d

sudo rm -rf /opt/cni/bin
sudo rm -rf /var/lib/kubelet
sudo rm -rf /var/lib/cni
sudo rm -rf /etc/kubernetes
sudo rm -rf /run/calico
sudo rm -rf /run/flannel

sudo rm -rf /etc/ceph
sudo rm -rf /opt/rke
sudo rm -rf /var/lib/calico
sudo rm -rf /var/lib/etcd

sudo rm -rf /var/log/containers
sudo rm -rf /var/log/pods

# rancher full reset !!!
sudo rm -rf /var/lib/rancher/*
sudo rm -rf /var/lib/rancher-log/*
```

upgrade k8s

logs

```
### Master
## API Server, responsible for serving the API
/var/log/kube-apiserver.log
## Scheduler, responsible for making scheduling decisions
/var/log/kube-scheduler.log
## Controller that manages replication controllers
/var/log/kube-controller-manager.log
### Worker Nodes
## Kubelet, responsible for running containers on the node
/var/log/kubelet.log
## Kube Proxy, responsible for service load balancing
/var/log/kube-proxy.log
```

CLI

kubernetes version, k8s version

```
kubeadm version
```

one of the field will be like: GitVersion:"v1.11.1"

access cluster

- reverse proxy activate proxy from current node

```
kubectl proxy --port 9090
# execute request against kubectl via reverse-proxy
curl {current node ip}:9090/api
```

- token access

```
$TOKEN=$(kubectl describe secret $(kubectl get secrets | grep ^default | cut -f1
echo $TOKEN | tee token.crt
echo "Authorization: Bearer "$TOKEN | tee token.header
# execute from remote node against ( cat ~/.kube/config | grep server )
curl https://{ip:port}/api --header @token.crt --insecure
```

connect to remote machine, rsh

```
# connect to remote machine
kubectl --namespace namespace-metrics --kubeconfig=config-rancher exec -ti sm-
grafana-deployment-5bdb64-6dnb8 -- /bin/sh
```

check namespaces

```
kubectl get namespaces
```

at least three namespaces will be provided

```
default       Active    15m
kube-public   Active    15m
kube-system   Active    15m
```

create namespace

```
kubectl create namespace my-own-namespace
```

or via yaml file

```
kubectl apply -f {filename}
```

```
kind: Namespace
apiVersion: v1
metadata:
  name: test
```

create limits for namespace

example for previous namespace declaration

```
apiVersion: v1
kind: LimitRange
metadata:
  name: my-own-namespace
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

limits for certain container

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
```

```
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

> also can be limited: pods, pv/pvc, services, configmaps...

print limits

```
kubectl get quota --namespace my-own-namespace
kubectl describe quota/compute-quota --namespace my-own-namespace
kubectl describe quota/object-quota --namespace my-own-namespace

kubectl describe {pod-name} limits
kubectl describe {pod-name} limits --namespace my-own-namespace
```

delete namespace

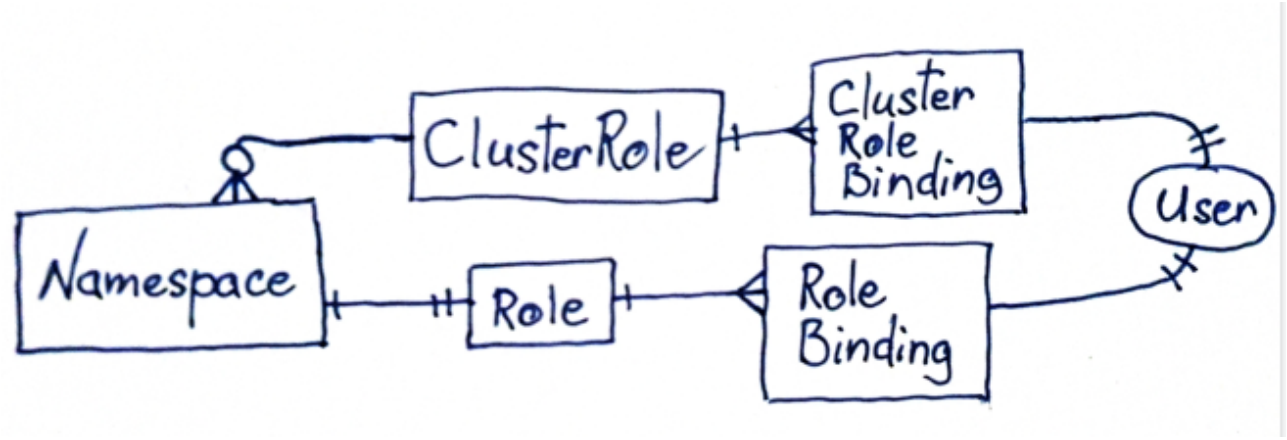
```
kubectl delete namespace {name of namespace}
```

users

- normal user
  - client certificates
  - bearer tokens
  - authentication proxy
  - http basic authentication
  - OpenId
- service user
  - service account tokens
  - credentials using secrets

    - specific to namespace
    - created by objects
    - anonymous user (not authenticated)

exeternal applications, user management, managing users



- https://github.com/sighupio/permission-manager
- https://blog.kubernauts.io/permission-manager-rbac-management-for-kubernetes-ed46c2f38cfb

```
be sure that your kube-apiserver is using RBAC autorization
```bash
ps aux | grep kube-apiserver
# expected output
# --authorization-mode=Node,RBAC
```

```
# read existing roles
kubectl get clusterRoles
# describe roles created by permission-management
kubectl describe clusterRoles/template-namespaced-resources___developer
kubectl describe clusterRoles/template-namespaced-resources___operation

# get all rolebindings
kubectl get RoleBinding --all-namespaces
kubectl get ClusterRoleBinding  --all-namespaces
kubectl get rolebindings.rbac.authorization.k8s.io --all-namespaces


# describe one of bindings
kubectl describe ClusterRoleBinding/student1___template-cluster-resources___read-onl
kubectl describe rolebindings.rbac.authorization.k8s.io/student1___template-namespac
```
```

Direct request to api, user management curl

```
TOKEN="Authorization: Basic YWRtaW46b2xnYSZ2aXRhbGlp"
curl -X GET -H "$TOKEN" http://localhost:4000/api/list-users
```

etcd

etcdctl installation

untar etcdctl from https://github.com/etcd-io/etcd/releases

etcd querying, etcd request key-values

```
docker exec -ti `docker ps | grep etcd | awk '{print $1}'` /bin/sh
etcdctl get / --prefix --keys-only
# etcdctl --endpoints=http://localhost:2379 get / --prefix --keys-only
etcdctl  get / --prefix --keys-only | grep permis
etcdctl  get /registry/namespaces/permission-manager -w=json
```

configuration, configmap

create configmap

example of configuration

```
color.ok=green
color.error=red
textmode=true
security.user.external.login_attempts=5
```

- create configuration on cluster

  ```
  kubectl create configmap my-config-file --from-env-
  file=/local/path/to/config.properties
  ```

  will be created next configuration

  ```
  ...
  data:
    color.ok=green
  ```

```
    color.error=red
    textmode=true
    security.user.external.login_attempts=5
```

- or configuration with additional key, additional abstraction over the properties (like Map of properties)

  ```
  kubectl create configmap my-config-file --from-file=name-or-key-of-config=/local/path/to/config.properties
  ```

  created file is:

  ```
  data:
    name-or-key-of-config:
      color.ok=green
      color.error=red
      textmode=true
      security.user.external.login_attempts=5
  ```

- or configuration with additional key based on filename (key will be a name of file)

  ```
  kubectl create configmap my-config-file --from-file=/local/path/to/
  ```

  created file is:

  ```
  data:
    config.properties:
      color.ok=green
      color.error=red
      textmode=true
      security.user.external.login_attempts=5
  ```

- or inline creation

  ```
  kubectl create configmap special-config --from-literal=color.ok=green --from-literal=color.error=red
  ```

get configurations, read configuration in specific format

```
kubectl get configmap
kubectl get configmap --namespace kube-system
kubectl get configmap --namespace kube-system kube-proxy --output json
```

using configuration, using of configmap

- one variable from configmap

```
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "echo $(MY_ENVIRONMENT_VARIABLE)" ]
      env:
        - name: MY_ENVIRONMENT_VARIABLE
          valueFrom:
            configMapKeyRef:
              name: my-config-file
              key: security.user.external.login_attempts
```

- all variables from configmap

```
...
     envFrom:
     - configMapRef:
         name: my-config-file
```

start readiness, check cluster

```
kubectl cluster-info dump
kubectl get node
minikube dashboard
```

addons

```
minikube addons list
minikube addons enable ingress
```

labels

Executable File | 1400 lines (1222 sloc) | 38.3 KB

```
kubectl get nodes --show-labels
```

add label to Node

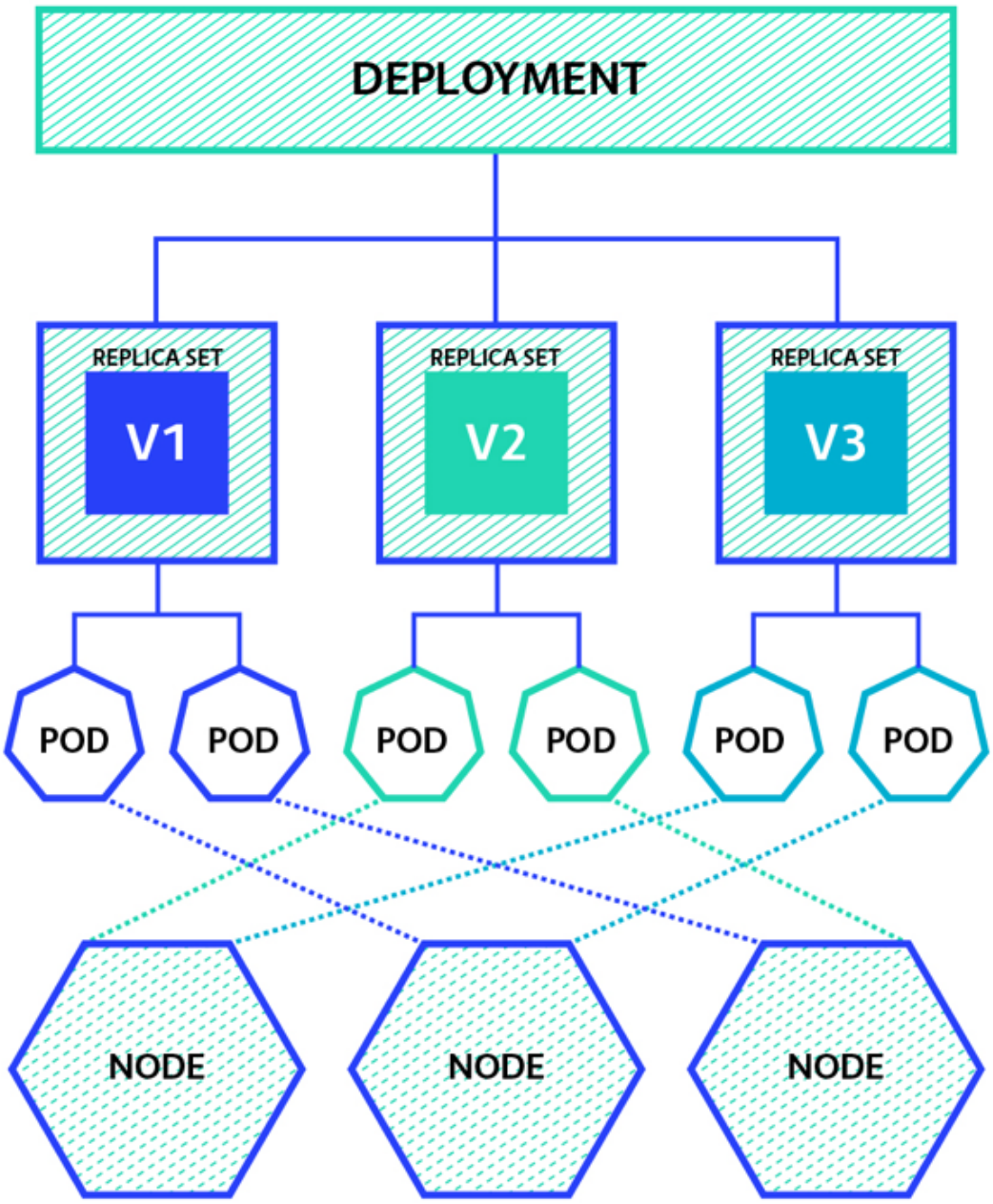
```
kubectl label nodes {node name} my_label=my_value
```

remove label from Node

```
kubectl label nodes {node name} my_label-
```

deployment

to see deployment from external world, remote access to pod, deployment access:
user ----> Route -----> Service ----> Deployment



start dummy container

```
kubectl run hello-minikube --image=k8s.gcr.io/echoserver:1.4 --port=8080
```

create deployment (with replica set)

```
kubectl run http --image=katacoda/docker-http-server:latest --replicas=1
```

scale deployment

```
kubectl scale --replicas=3 deployment {name of the deployment}
```

create from yaml file

```
kubectl create -f /path/to/controller.yml
```

update yaml file

```
kubectl apply -f /path/to/controller.yml
```

create service, expose service, inline service fastly

```
kubectl expose deployment helloworld-deployment --type=NodePort --
name=helloworld-service
kubectl expose deployment helloworld-deployment --external-ip="172.17.0.13" --
port=8000 --target-port=80
```

port forwarding, expose service

```
kubectl port-forward svc/my_service 8080 --namespace my_namespace
```

reach out service

```
minikube service helloworld-service
minikube service helloworld-service --url
```

service port range

```
kube-apiserver --service-node-port-range=30000-40000
```

describe resources, information about resources, inspect resources, inspect pod

```
kubectl describe deployment {name of deployment}
kubectl describe service {name of service}
```

describe users, user token

```
kubectl --namespace kube-system describe secret admin-user
```

get resources

```
kubectl get all --all-namespaces
kubectl get pods
kubectl get pods --namespace kube-system
kubectl get pods --show-labels
kubectl get pods --output=wide --selector="run=load-balancer-example"
kubectl get pods --namespace training --field-
selector="status.phase==Running,status.phase!=Unknown"
kubectl get service --output=wide
kubectl get service --output=wide --selector="app=helloworld"
kubectl get deployments
kubectl get replicasets
kubectl get nodes
kubectl get cronjobs
kubectl get daemonsets
kubectl get pods,deployments,services,rs,cm,pv,pvc -n demo
```

determinate cluster 'hostIP' to reach out application(s)

```
minikube ip
```

open 'kube-dns-....'/hostIP open 'kube-proxy-....'/hostIP

edit configuration of controller

```
kubectl edit pod hello-minikube-{some random hash}
kubectl edit deploy hello-minikube
kubectl edit ReplicationControllers helloworld-controller
kubectl set image deployment/helloworld-deployment {name of image}
```

rollout status

```
kubectl rollout status  deployment/helloworld-deployment
```

rollout history

```
kubectl rollout history  deployment/helloworld-deployment
kubectl rollout undo deployment/helloworld-deployment
kubectl rollout undo deployment/helloworld-deployment --to-revision={number of
revision from 'history'}
```

delete running container

```
kubectl delete pod hello-minikube-6c47c66d8-td9p2
```

delete deployment

```
kubectl delete deploy hello-minikube
```

delete ReplicationController

```
kubectl delete rc helloworld-controller
```

delete PV/PVC

```
oc delete pvc/pvc-scenario-output-prod
```

port forwarding from local to pod/deployment/service

next receipts allow to redirect 127.0.0.1:8080 to pod:6379

```
kubectl port-forward redis-master-765d459796-258hz      8080:6379
kubectl port-forward pods/redis-master-765d459796-258hz 8080:6379
kubectl port-forward deployment/redis-master            8080:6379
kubectl port-forward rs/redis-master                    8080:6379
kubectl port-forward svc/redis-master                   8080:6379
```

NodeSelector for certain host

```
spec:
   template:
      spec:
         nodeSelector:
            kubernetes.io/hostname: gtxmachine1-ev
```

persistent volume

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-volume3
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data3"
```

to access created volume

```
ls /mnt/data3
```

list of existing volumes

```
kubectl get pv
kubectl get pvc
```

container lifecycle

```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
  - name: lifecycle-demo-container
    image: nginx
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo Hello from the postStart handler > /usr/s
      preStop:
        exec:
          command: ["/bin/sh","-c","nginx -s quit; while killall -0 nginx; do sleep
```

```
containers:
  - name: lifecycle
    image: busybox
    lifecycle:
      postStart:
        exec:
          command:
            - "touch"
            - "/var/log/lifecycle/post-start"
      preStop:
        httpGet:
          path: "/abort"
          port: 8080
```

Serverless

- OpenFaas
- Kubeless
- Fission
- OpenWhisk

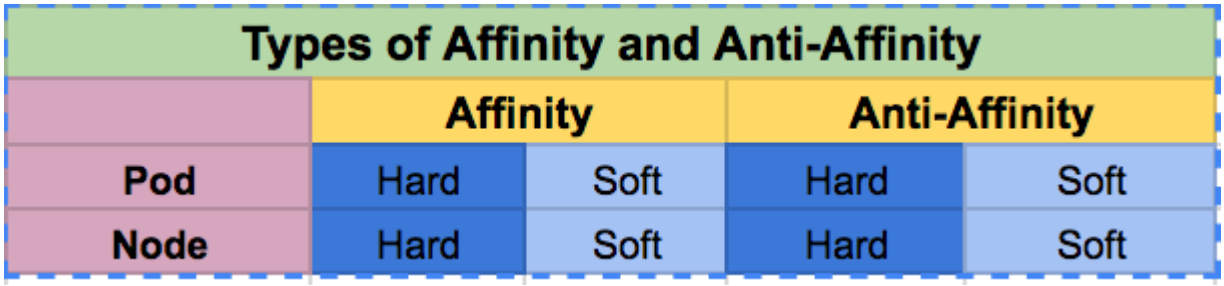
deploy Pod on Node with label

```
apiVersion: v1
kind: Pod
metadata:
...
spec:
...
  nodeSelector:
    my_label=my_value
```

create Deployment for specific node

```
apiVersion: some-version
kind: Deployment
metadata:
...
spec:
...
  nodeSelector:
    my_label=my_value
```

resolving destination node



- nodeAffinity
- - preferred - deploy in any case, with preferrence my_label=my_value

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: my_label
            operator: In
            values:
            - my_value
```

- required - deploy only when label matched my_label=my_value

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: my_label
            operator: In
            values:
            - my_value
```

- nodeAntiAffinity

```
spec:
  affinity:
    nodeAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

- podAffinity
- preferred spec.affinity.podAffinity.preferredDuringSchedulingIgnoredDuringExecution
- required spec.affinity.podAffinity.requiredDuringSchedulingIgnoredDuringExecution
- podAntiAffinity
- preferred spec.affinity.podAntiAffinity.preferredDuringSchedulingIgnoredDuringExecution
- required spec.affinity.podAntiAffinity.requiredDuringSchedulingIgnoredDuringExecution

delete node from cluster

```
kubectl get nodes
kubectl delete {node name}
```

add node to cluster

```
ssh {master node}
kubeadm token create --print-join-command  --ttl 0
```

expected result from previous command

```
kubeadm join 10.14.26.210:6443 --token 7h0dmx.2v5oe1jwed --discovery-token-ca-
cert-hash sha256:1d28ebf950316b8f3fdf680af5619ea2682707f2e966fc0
```

go to node, clean up and apply token

```
ssh {node address}
# hard way: rm -rf /etc/kubernetes
kubeadm reset
# apply token from previous step with additional flag: --ignore-preflight-
errors=all
kubeadm join 10.14.26.210:6443 --token 7h0dmx.2v5oe1jwed --discovery-token-ca-
cert-hash sha256:1d28ebf950316b8f3fdf680af5619ea2682707f2e966fc0 --ignore-
preflight-errors=all
```

expected result from previous command

```
...
This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

next block is not mandatory in most cases

```
systemctl restart kubelet
```

logs

```
kubectl logs <name of pod>
```

create dashboard

```
kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/master/aio/deploy/recommended
dashboard.yaml
```

access dashboard

```
kubectl -n kube-system describe secret admin-user
http://127.0.0.1:8001/api/v1/namespaces/kube-system/services/https:kubernetes-
dashboard:/proxy/#!/overview?namespace=default
kubectl proxy
```

common

execute command on specific pod

```
kubectl exec -it {name of a pod}  -- bash -c "echo hi > /path/to/output/test.txt"
```

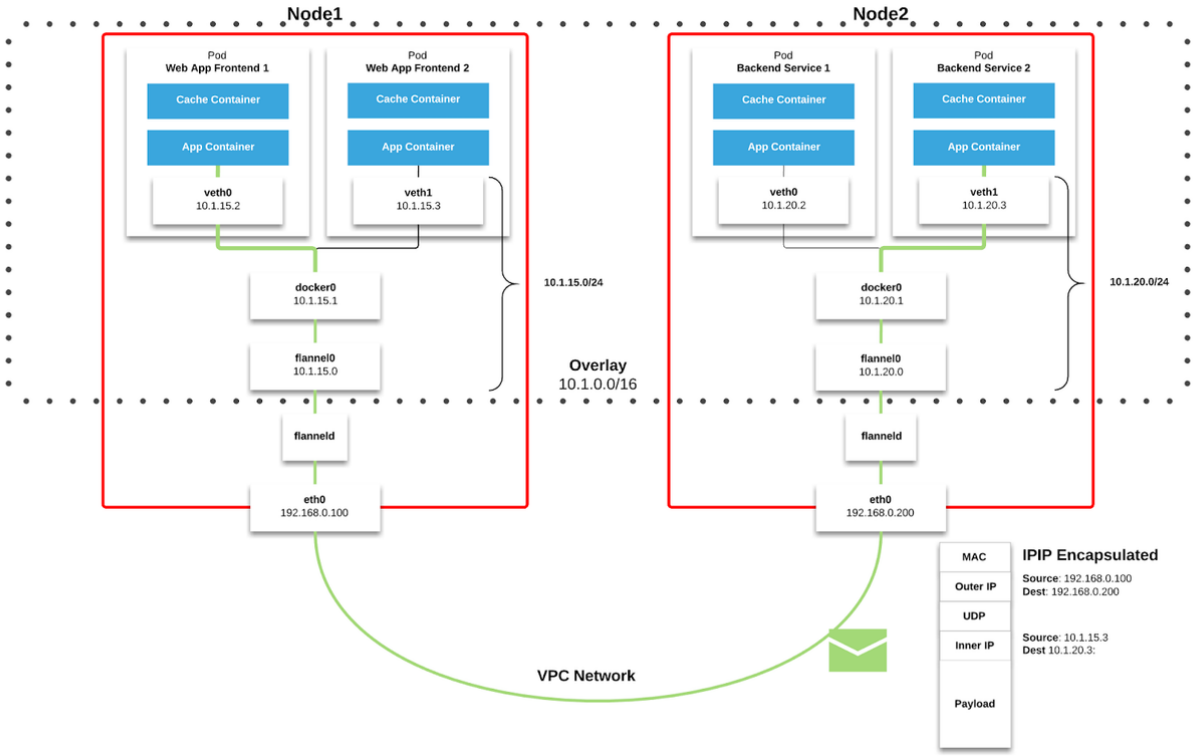
Extending

custom controller

Weave

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |
```

Flannel



restart nodes

```
# remove died pods
kubectl delete pods kube-flannel-ds-amd64-zsfz  --grace-period=0 --force
# delete all resources from file and ignore not found
kubectl delete -f --ignore-not-found https://raw.githubusercontent.com/coreos/flanne
kubectl create  -f https://raw.githubusercontent.com/coreos/flannel/master/Documenta
```

install flannel

```
### apply this with possible issue with installation:
## kube-flannel.yml": daemonsets.apps "kube-flannel-ds-s390x" is forbidden: User
"system:node:name-of-my-server" cannot get daemonsets.apps in the namespace
"kube-system"
```

```
# sudo kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878
flannel.yml

## Container runtime network not ready: NetworkReady=false
reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni
config uninitialized
# sudo kubectl -n kube-system apply -f
https://raw.githubusercontent.com/coreos/flannel/bc79dd1505b0c8681ece4de4c0d86c5cd26
flannel.yml

## print all logs
journalctl -f -u kubelet.service
# $KUBELET_NETWORK_ARGS in
# /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

## ideal way, not working properly in most cases
sudo kubectl -n kube-system apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-
flannel.yml

## check installation
ps aux | grep flannel
# root     13046  0.4  0.0 645968 24748 ?        Ssl  10:49   0:00
/opt/bin/flanneld --ip-masq --kube-subnet-mgr

ifconfig
cni0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.244.0.1  netmask 255.255.255.0  broadcast 0.0.0.0
flannel.1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
        inet 10.244.0.0  netmask 255.255.255.255  broadcast 0.0.0.0
```

change settings and restart

```
kubectl edit cm kube-flannel-cfg -n kube-system
# net-conf.json: | { "Network": "10.244.0.0/16", "Backend": { "Type": "vxlan" } }

# Wipe current CNI network interfaces remaining the old network pool:
sudo ip link del cni0; sudo ip link del flannel.1

# Re-spawn Flannel and CoreDNS pods respectively:
kubectl delete pod --selector=app=flannel -n kube-system
kubectl delete pod --selector=k8s-app=kube-dns -n kube-system

# waiting for restart of all services
```

read logs

```
kubectl logs --namespace kube-system kube-flannel-ds-amd64-j4frw -c kube-flannel
```

read logs from all pods

```
for each_node in $(kubectl get pods --namespace kube-system | grep flannel | awk
'{print $1}');do echo $each_node;kubectl logs --namespace kube-system $each_node
-c kube-flannel;done
```

read settings

```
kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp ls /etc/kube-
flannel/
kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp cat /etc/kube-
flannel/cni-conf.json
kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp cat /etc/kube-
flannel/net-conf.json

kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp ls /run/flannel/
kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp cat
/run/flannel/subnet.env

kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp ls
/etc/cni/net.d
kubectl --namespace kube-system exec kube-flannel-ds-amd64-wc4zp cat
/etc/cni/net.d/10-flannel.conflist
```

read DNS logs

```
kubectl get svc --namespace=kube-system | grep kube-dns
kubectl logs --namespace=kube-system coredns-78fcd94-7tlpw | tail
```

simple POD, dummy pod, waiting pod

```
kind: Pod
apiVersion: v1
metadata:
  name: sleep-dummy-pod
  namespace: students
spec:
  containers:
    - name: sleep-dummy-pod
```

```
      image: ubuntu
      command: ["/bin/bash", "-ec", "while :; do echo '.'; sleep 3600 ; done"]
  restartPolicy: Never
```

NFS (Network File System)

nfs server

```
# nfs server
vim /etc/exports
# /mnt/disks/k8s-local-storage/nfs        10.55.0.0/16(rw,sync,no_subtree_check)
# /mnt/disks/k8s-local-storage1/nfs       10.55.0.0/16(rw,sync,no_subtree_check)

sudo exportfs -a
sudo exportfs -v

systemctl status nfs-server
ll /sys/module/nfs/parameters/
ll /sys/module/nfsd/parameters/
sudo blkid
sudo vim /etc/fstab
# UUID=35c71cfa-6ee2-414a-5555-effc30555555 /mnt/disks/k8s-local-storage ext4 defaul
# UUID=42665716-1f89-44d4-5555-37b207555555 /mnt/disks/k8s-local-storage1 ext4 defau
nfsstat
```

master. mount volume (nfs server)

```
# create point
sudo mkdir /mnt/disks/k8s-local-storage1
# mount
sudo mount /dev/sdc /mnt/disks/k8s-local-storage1
sudo chmod 755 /mnt/disks/k8s-local-storage1
# createlink
sudo ln -s /mnt/disks/k8s-local-storage1/nfs /mnt/nfs1
ls -la /mnt/disks
ls -la /mnt

# update storage
sudo cat /etc/exports
# /mnt/disks/k8s-local-storage1/nfs       10.55.0.0/16(rw,sync,no_subtree_check)

# restart
```

```
sudo exportfs -a
sudo exportfs -v
```

nfs client

```
sudo blkid

sudo mkdir /mnt/nfs1
sudo chmod 777 /mnt/nfs1

sudo vim /etc/fstab
# add record
# 10.55.0.3:/mnt/disks/k8s-local-storage1/nfs /mnt/nfs1 nfs rw,noauto,x-systemd.autc
10.55.0.3:/mnt/disks/k8s-local-storage1/nfs /mnt/nfs1 nfs defaults 0 0

# refresh fstab
sudo mount -av

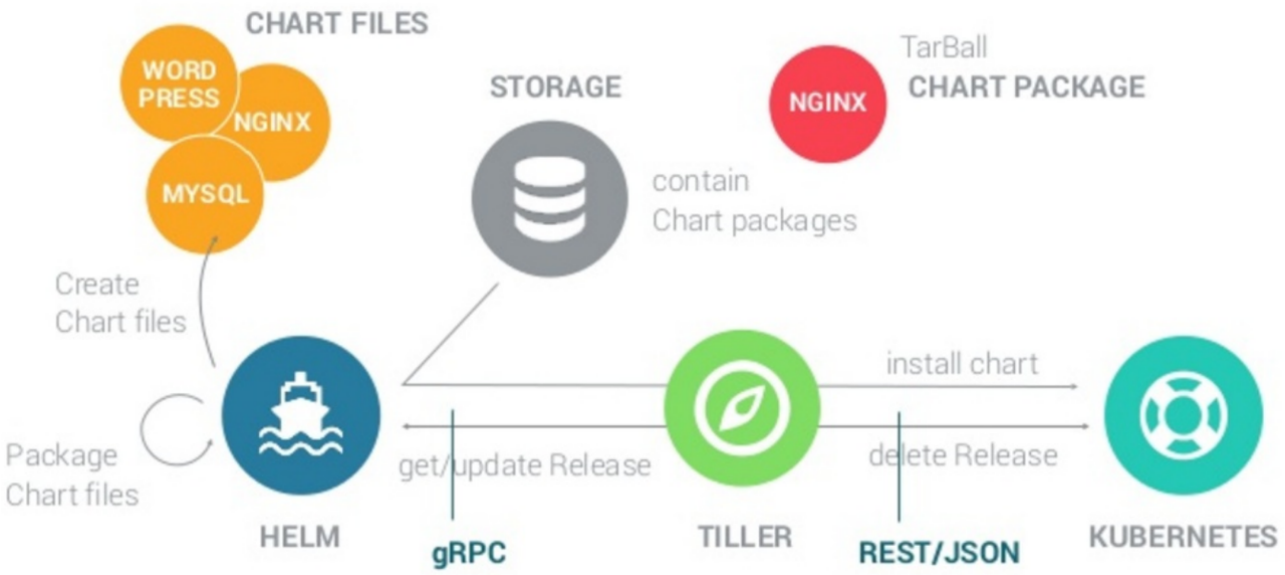
# for server
ls /mnt/disks/k8s-local-storage
ls /mnt/disks/k8s-local-storage1

# for clients
ls /mnt/disks/k8s-local-storage1
```

Helm

documentation

Architecture



installation

```
sudo snap install helm --classic
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
```

de-installation

```
helm reset
```

initialization

```
helm init
# sync latest available packages
helm repo update
```

useful variables

```
* $HELM_HOME: the location of Helm's configuration
* $TILLER_HOST: the host and port that Tiller is listening on
* $HELM_BIN: the path to the helm command on your system
```

```
* $HELM_PLUGIN_DIR: the full path to this plugin (not shown above, but we'll see
it in a moment).
```

analyze local package

```
helm inspect { folder }
helm lint { folder }
```

search remote package

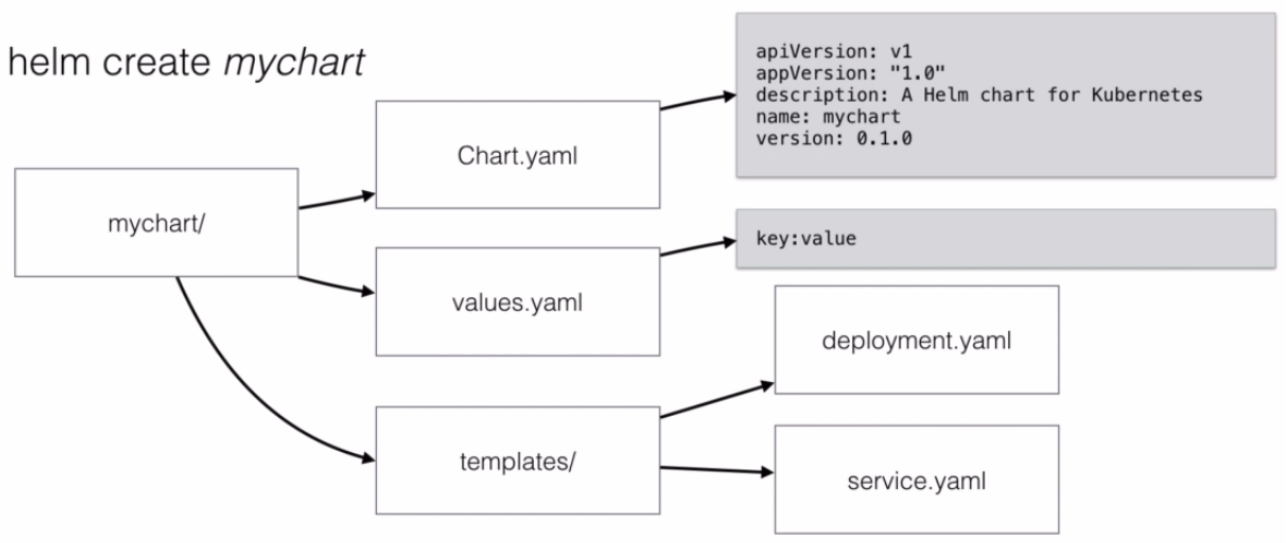
```
helm search
helm describe {full name of the package}
```

information about remote package

```
helm info {name of resource}
helm status {name of resource}
```

create package locally

```
helm create
```



create package with local templates

```
ls -la ~/.helm/starters/
```

install package

```
helm install { full name of the package }
helm install --name {my name for new package} { full name of the package }
helm install --name {my name for new package} --namespace {namespace} -f
values.yml --debug --dry-run { full name of the package }
```

install aws plugin

```
helm plugin install https://github.com/hypnoglow/helm-s3.git
```

list of installed packages

```
helm list
helm list --all
helm ls
```

package upgrade

local package

```
helm upgrade  {deployment/svc/rs/rc name} . --set
replicas=2,maria.db.password="new password"
```

package by name

```
helm upgrade {name of package} {folder with helm scripts} --set replicas=2
```

check upgrade

```
helm history
helm rollback {name of package} {revision of history}
```

remove packageHelm

```
helm delete --purge {name of package}
```

trouble shooting

issue with 'helm list'

```
E1209 22:25:57.285192    5149 portforward.go:331] an error occurred forwarding
40679 -> 44134: error forwarding port 44134 to pod
de4963c7380948763c96bdda35e44ad8299477b41b5c4958f0902eb821565b19, uid : unable to
do port forwarding: socat not found.
Error: transport is closing
```

solution

```
sudo apt install socat
```

incompatible version of client and server

```
Error: incompatible versions client[v2.12.3] server[v2.11.0]
```

solution

```
helm init --upgrade
kubectl get pods --namespace kube-system # waiting for start Tiller
helm version
```

certificate is expired

```
bootstrap.go:195] Part of the existing bootstrap client certificate is expired:
2019-08-22 11:29:48 +0000 UTC
```

solution

```
sudo cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf

# archive configuration
sudo cp /etc/kubernetes/pki /etc/kubernetes/pki_backup
sudo mkdir /etc/kubernetes/conf_backup
sudo cp /etc/kubernetes/*.conf /etc/kubernetes/conf_backup

# remove certificates
sudo rm /etc/kubernetes/pki/./apiserver-kubelet-client.crt
sudo rm /etc/kubernetes/pki/./etcd/healthcheck-client.crt
sudo rm /etc/kubernetes/pki/./etcd/server.crt
sudo rm /etc/kubernetes/pki/./etcd/peer.crt
sudo rm /etc/kubernetes/pki/./etcd/ca.crt
sudo rm /etc/kubernetes/pki/./front-proxy-client.crt
sudo rm /etc/kubernetes/pki/./apiserver-etcd-client.crt
sudo rm /etc/kubernetes/pki/./front-proxy-ca.crt
sudo rm /etc/kubernetes/pki/./apiserver.crt
sudo rm /etc/kubernetes/pki/./ca.crt
sudo rm /etc/kubernetes/pki/apiserver.crt
sudo rm /etc/kubernetes/pki/apiserver-etcd-client.crt
sudo rm /etc/kubernetes/pki/apiserver-kubelet-client.crt
sudo rm /etc/kubernetes/pki/ca.crt
sudo rm /etc/kubernetes/pki/front-proxy-ca.crt
sudo rm /etc/kubernetes/pki/front-proxy-client.crt

# remove configurations
```

```
sudo rm /etc/kubernetes/apiserver-kubelet-client.*
sudo rm /etc/kubernetes/front-proxy-client.*
sudo rm /etc/kubernetes/etcd/*
sudo rm /etc/kubernetes/apiserver-etcd-client.*
sudo rm /etc/kubernetes/admin.conf
sudo rm /etc/kubernetes/controller-manager.conf
sudo rm /etc/kubernetes/kubelet.conf
sudo rm /etc/kubernetes/scheduler.conf

# re-init certificates
sudo kubeadm init phase certs all --apiserver-advertise-address {master ip address}

# re-init configurations
sudo kubeadm init phase kubeconfig all --ignore-preflight-errors=all

# re-start
sudo systemctl stop kubectl.service
sudo systemctl restart docker.service
docker system prune -af --volumes
reboot

# /usr/bin/kubelet
sudo systemctl start kubectl.service

# init locate kubectl
sudo cp /etc/kubernetes/admin.conf ~/.kube/config

# check certificate
openssl x509 -in /etc/kubernetes/pki/apiserver.crt  -noout -text  | grep "Not After"
```

template frameworks

- go template
- sprig template