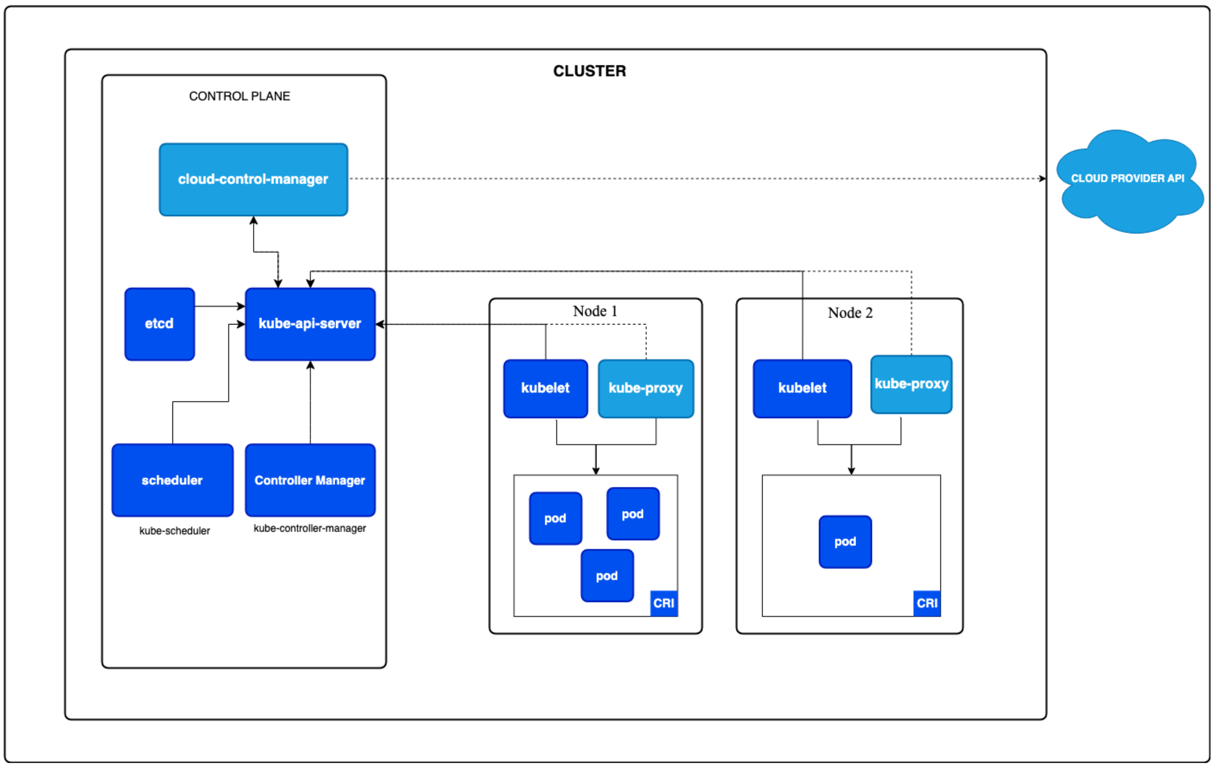
Kubernetes - K8s

Kubernetes Architecture



In each cluster there are three main components that we need to know before working with K8s

- Master Node (Control plane)
- Worker Node
- etcd

Master Node

- Controlled and managed all of operation in K8s included
    - kube-api-server : act as an intermediary for cluster communication via RESTFul API
    - kube-scheduler : act as an pod manager when new Pos comes in it will find the appropriate worker node for that coming pod which requires information from many sources to make decisions
    - kube-contoller-manager :
        - Node Controller
        - Replication Controller : manage pod replication
        - Endpoints Controller

        - Service Account & Token controllers : manage service account and API access token
    - cloud-controller-manager : this controller is available in only Cloud Provider that support K8s. It act like a kube-controller-manager but working with Cloud Provider instead.

Worker Node

- Container runtime included two services
    - kubelet : Receive commands from kube-api-server and manage container runtime, such as checking if Pods (Containers) are still running on Worker nodes. If there is a problem, it will report to kube-api-server that there is a problem with the Pods and make them manage the Initial Pods again. (manager of Pods)
    - kube-proxy : this service help to make Pods can be connected from outside the Cluster it act like a proxy server to receive request from outside then resend again to appropriate pods inside the node.

etcd

Use to store all of Persistance cluster state this components also in control plane which directly connects by kube-api-server

Networking in Kubernetes

- **Container-to-container** communication within a Pod
- **Pod-to-Pod** communication across nodes
- **Pod-to-Service** communication within the cluster
- **External-to-Service** communication with the outside world

Type of Objects kin Kubernetes

- Pods : A Pod can host a single container or a group of containers. It's important to note that, when a Pod contains multiple containers, all of the containers always run on a single worker node. Pods is that they are **ephemeral in nature**
- Deployment : **allows you to declaratively manage the desired state of your application**, such as the number of replicas, the image to use for the Pods, and the resources required.
- ReplicaSets : **Deployments don’t manage Pods directly**. That’s the job of the ReplicaSet object.
    - ReplicaSet is created automatically

- The ReplicaSet ensures that the desired number of replicas (copies) are running at all times by creating or deleting Pods as needed.

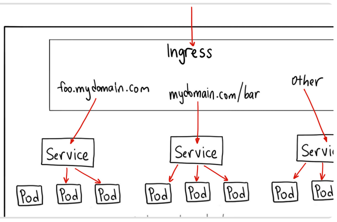
- **StatefulSet**
  - **ensures that each Pod is uniquely identified by a number, starting at zero**.
  - When a Pod in a StatefulSet must be replaced, for example, due to node failure, the new Pod is assigned the same numeric identifier as the previous Pod.
- **DaemonSets**
  - **ensures that a copy of a Pod is running across all, or a subset of nodes in a Kubernetes cluster**.
- **PersistentVolume**
  - **represents a piece of storage that you can attach to your Pod(s).**
  - if your Pod gets deleted, the PersistentVolume will survive.
  - can attach using a PersistentVolume, like local disks, network storage, and cloud storage.
- Service : **a way to access a group of Pods that provide the same functionality**. the Service also provides some simple load balancing.
- **Namespaces**
  - **a way to divide a single Kubernetes cluster into multiple virtual clusters**.

ClusterIP vs NodePort vs Loadbalancer

Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?

Recently, someone asked me what the difference between NodePorts, LoadBalancers, and Ingress were. They are all different ways to get...

https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0



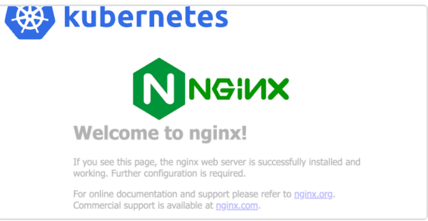
- ClusterIP : the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.
- NodePort : **is a way to expose your application to external clients but doesn't contain LoadBalancer**
- LoadBalancer : is NodePort bit including LoadBalancer
  - key different NodePort and LoadBalancer
    - NodePort : Client → node
    - LoadBalancer : client → load balancer → node

Deploying Nginx on Kuberetes Various way.

Deploying Nginx on Kubernetes: Exploring Various Methods

Mastering Nginx Deployment in Kubernetes: A Comprehensive Guide to Pods, Deployments, and Beyond

https://medium.com/@muppedaanvesh/deploying-nginx-on-kubernetes-a-quick-guide-04d533414967



ConfigMap & Secret

- ConfigMap
  - non-confidential
  - key-value format
  - can be injected into container
- Secret

  Kubernetes Config & Secrets: Essential Guide to ConfigMaps

  Explore managing Kubernetes ConfigMaps & Secrets for efficient app deployment. Learn best practices & essential tips for containerized environments...

  https://www.getambassador.io/blog/kubernetes-configurations-secrets-configmaps

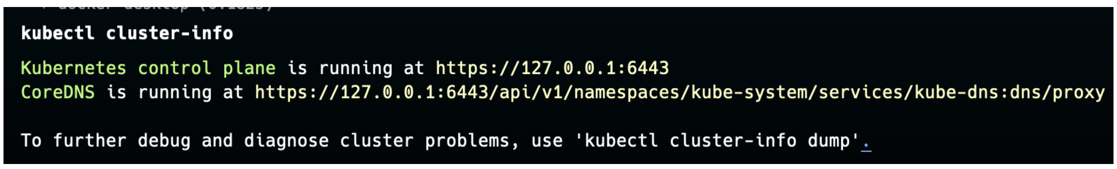
  ambassador
Blog
Managing Configurations and Secrets in Kubernetes: ConfigMaps and Secrets

  - The same as ConfigMap
  - sensitive information

Workshop

Initially, I use docker desktop on Mac and enable to use an Kubernetes already,

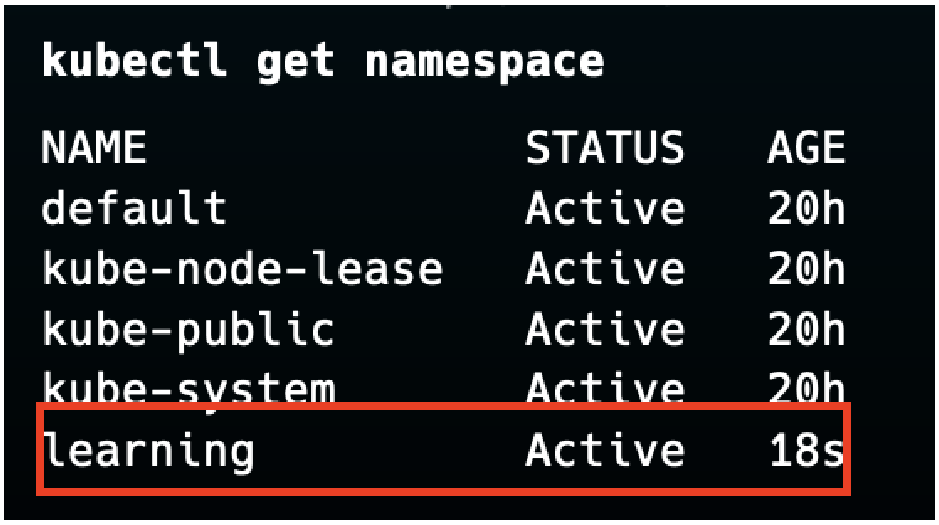
- first I will check if Kubernetes is working by
  - run `kubectl cluster-info` to check if there is any response to me
    - the response is



```
kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:6443
CoreDNS is running at https://127.0.0.1:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

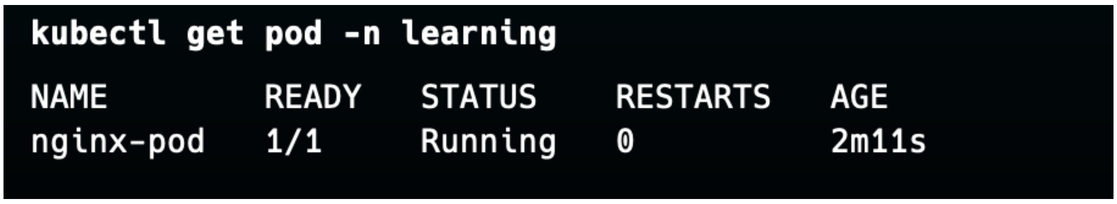
- Next I will try to deploy Nginx with many method and strategy. Before we start we need to create new namespace for this learning section via `kubectl create namespace learning` . After we created let's check the result if everything work find let's get started.



```
kubectl get namespace

NAME              STATUS   AGE
default           Active   20h
kube-node-lease   Active   20h
kube-public       Active   20h
kube-system       Active   20h
learning          Active   18s
```

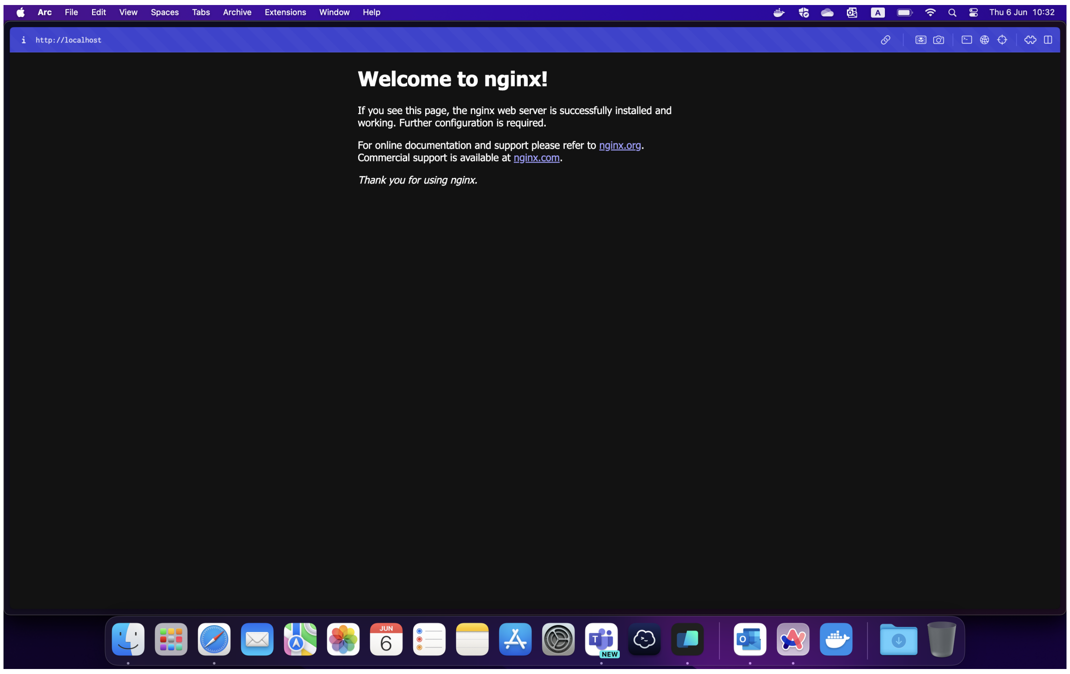
- **Imperative Way**
  - run `kubectl run nginx-pod --image=nginx --port=80 -n learning`
  - Now check the pod that created



```
kubectl get pod -n learning

NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          2m11s
```

  - Now if you need to enter this site this deployed, Sorry you can't because you need to setup for some service that would be the door that external user need to connect to. let's type `kubectl expose pod nginx-pod --type=LoadBalancer --port=80 --name=nginx-service --namespace=learning`
  - then check the result in website on localhost:80 should be look like this.


http://localhost

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

- Use a maniface file
  - Create a file call nginx.yaml and copy this script to create pod and service like above.

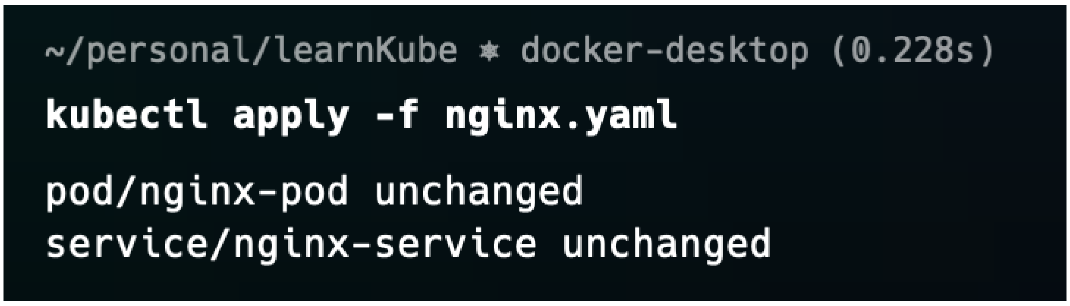
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
  namespace: learning
spec:
  containers:
  - name: nginx-container
    image: nginx:latest
    ports:
      - containerPort: 80
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
      requests:
        memory: "64Mi"
        cpu: "250m"

---

apiVersion: v1
kind: Service
```

```
metadata:
  name: nginx-service
  namespace: learning
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

- after that save the file then run `kubectl apply -f nginx.yaml`



```
~/personal/learnKube ⎈ docker-desktop (0.228s)
kubectl apply -f nginx.yaml

pod/nginx-pod unchanged
service/nginx-service unchanged
```

- now check the result.