## **Installation:**

<https://kubernetes.io/docs/tasks/tools/>

1. Install minikube from <https://minikube.sigs.k8s.io/docs/start/>
2. Launch Docker for Desktop
3. Run minikube start (It will take a while to start)

Install Kubectl using (<https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/> )

1. curl.exe -Lo <https://dl.k8s.io/release/v1.28.4/bin/windows/amd64/kubectl.exe>
2. CertUtil -hashfile kubectl.exe SHA256
3. Enable metrics server addon for minikube

minikube addons enable metrics-server

## **Basic Commands**

* **Getting K8 version**

kubectl version –client

* **Understand all the nodes associated using**

Kubectl get nodes – Gets all the nodes present

* **Stop the minikube using**

Minikube stop

* **For finding all the processes running in kubectl**

kubectl get po -A

* **Use minikube to run the same commands:**

minikube kubectl -- get pods -A

* **Find cluster-info**

Kubectl cluster-info

* **Create a sample app and deploy it**

kubectl run --image=nginx web

* **Check that the pod is up and running**

kubectl get pods

kubectl describe pod web

* **Run a simple application using**

Kubectl apply -f web-declarative.yaml

* **Find the full config created with defaults**

kubectl get pod web-declarative -o yaml

* **Understand the YAML specific field details**

kubectl explain pod.spec.restartPolicy

### **Dashboard:**

minikube dashboard

### **Impertive Vs Declartive**

Kubectl run –image=nginx web

Kubectl create -f web-declarative.yaml

apiVersion: v1

kind: Pod

metadata:

  name: web-declarative

  labels:

    site: blog

spec:

  containers:

    - name: web

      image: nginx:1.18.0

Multi-container:

Kubectl create -f multi.yaml

apiVersion: v1

kind: Pod

metadata:

  name: multi

spec:

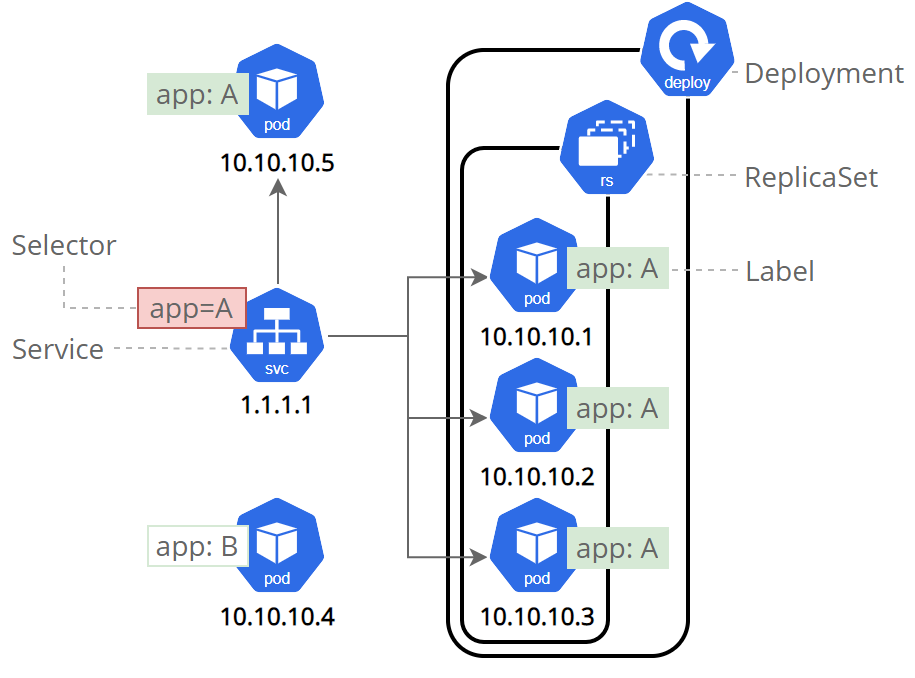
  containers:

    - name: c1

      image: ubuntu

    - name: c2

      image: ubuntu



### **Simple Deployment:**

<https://kubernetes.io/docs/tutorials/hello-minikube/>

1. Use minikube to create a cluster
   1. minikube start
2. Use Kubectl to create a pod
3. Create a sample deployment (imperative)
   1. kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.39 -- /agnhost netexec --http-port=8080
4. Check the status using
   1. Kubectl get deployments
   2. Kubectl get pods
5. Expose the node as service (imperative)
   1. kubectl expose deployment hello-node --type=LoadBalancer --port=8080
6. Run the service
   1. minikube service hello-node
7. Cleanup the service & deployment
   1. kubectl delete service hello-node
   2. kubectl delete deployment hello-node
   3. kubectl delete pod web

Troubleshooting:

If minikube doesn’t start -

docker context use default

If the docker doesn’t run from the hub, use

Docker login

If the service doesn’t run, use the image from dockerhub.

A deployment is responsible for keeping a set of pods running. A service is responsible for enabling network access to a set of pods

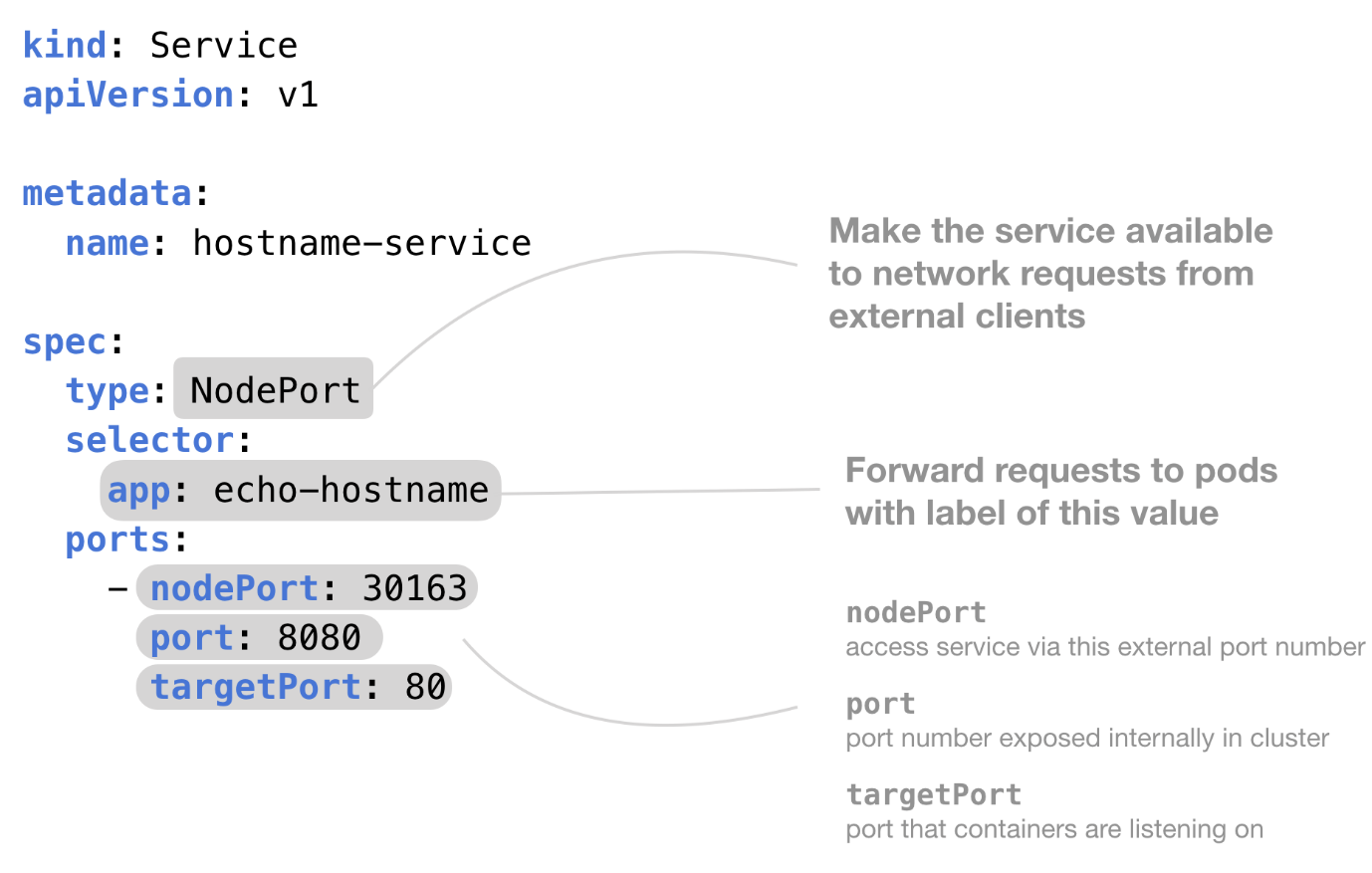
We could use a deployment without a service to keep a set of identical pods running in the Kubernetes cluster

We could also use a service without a deployment. We'd need to create each pod individually (rather than "all-at-once" like a deployment).

### **Templates**

<https://k8s.io/examples/controllers/nginx-deployment.yaml>

API reference: <https://kubernetes.io/docs/reference/kubernetes-api/>



Deployments:

Specifies replicas and pod defns.

apiVersion: apps/v1

kind: Deployment

metadata:

  name: rss-site

  labels:

    app: web

spec:

  replicas: 2

  selector:

    matchLabels:

      app: web

  template:

    metadata:

      labels:

        app: web

    spec:

      containers:

        - name: front-end

          image: nginx

          ports:

            - containerPort: 80

        - name: rss-reader

          image: nickchase/rss-php-nginx:v1

          ports:

            - containerPort: 88

* *ClusterIP* (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.
* *NodePort* - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.
* *LoadBalancer* - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.
* *ExternalName* - Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up. This type requires v1.7 or higher of kube-dns, or CoreDNS version 0.0.8 or higher.

1. Run kubectl create -f deployment.yaml
2. Kubectl get deployments

**You will see that the deployment is created.**

**Now, describe the deployment using**

1. Kubectl describe deployments rss-site

**You will see that two replicas are created.**

**Update the deployment using**

1. Kubectl apply -f deployment.yaml

Scale the deployment using

1. kubectl scale deployment rss-site --replicas=3
2. kubectl autoscale deployment rss-site --min=3 --max=5 --cpu-percent=10

# Creating a ML service from the scratch

1. minikube start

# Get the image and tag it

1. docker pull subbu0319/placement-app
2. docker image tag subbu0319/placement-app placement-app:v1
3. docker -p 9696:9696 placement-app:v1
4. python test.py (check results)

# Create service

Use the image name from ‘docker images ‘ and update in deployment.yaml file

1. kubectl create -f deployment.yaml
2. kubectl create -f service.yaml
3. minikube service placement-app
4. Will see the port no in the command line and also on browser
5. Change the port no in predict\_test.py and run it.
6. Check the logs if docker image is not picked or for any error
7. Scale the deployment kubectl placement-app --replicas=10

Deployment.yaml:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: placement-app

spec:

  selector:

    matchLabels:

      app: placement-app

  template:

    metadata:

      labels:

        app: placement-app

    spec:

      containers:

      - name: placement-app

        image: srisoudamini/placement-app

        resources:

          limits:

            memory: "128Mi"

            cpu: "500m"

        ports:

        - containerPort: 9696

Service.yaml:

apiVersion: v1

kind: Service

metadata:

  name: placement-app

spec:

  type: LoadBalancer

  selector:

    app: placement-app

  ports:

    - protocol: "TCP"

      port: 80

      targetPort: 9696

### **CheatSheet**

<https://kubernetes.io/docs/reference/kubectl/>

**-o=yaml – Output format in YAML.**

kubectl get pods -o=yaml

**-o=wide – Output in the plain-text format with any additional information, and for pods, the node name is included.**

kubectl get pods **-o=wide**

**-n – Alias for namespace.**

kubectl get pods -n=<namespace\_name>

**-f – Filename, directory, or URL to files to use to create a resource.**

kubectl create -f ./<file name>

**-l – Filter using a specified label.**

kubectl logs -l name=<label name>

**-h – Help for kubectl.**

kubectl -h

#### Configuration Files (Also referred to as Manifest or YAML Files)

Configuration Files (Also referred to as Manifest or YAML Files.) – Kubernetes objects can be created, updated, and deleted by storing multiple object configuration files in a directory and using kubectl apply to recursively create and update those objects as needed. This method retains writes made to live objects without merging the changes back into the object configuration files. kubectl diff also gives you a preview of what changes apply will make.

kubectl apply -f <configuration file>

kubectl create -f <configuration file> – Create objects.

kubectl create -f <configuration file directory> – Create objects in all manifest files in a directory.

kubectl create -f <‘url’> – Create objects from a URL.

kubectl delete -f <configuration file> – Delete an object.

#### Cluster Management and Context

**Cluster management refers to querying information about the K8S cluster itself.**

kubectl cluster-info – Display endpoint information about the master and services in the cluster.

kubectl version – Display the Kubernetes version running on the client and server.

kubectl config view – Get the configuration of the cluster.

kubectl config view -o jsonpath='{.users[\*].name}' – Get a list of users.

kubectl config current-context – Display the current context.

kubectl config get-contexts – Display a list of contexts.

kubectl config use-context <cluster name> – Set the default context.

kubectl api-resources – List the API resources that are available.

kubectl api-versions – List the API versions that are available.

-A – List pods, services, daemonsets, deployments, replicasets, statefulsets, jobs, and CronJobs in all namespaces, not custom resource types. Note the alias for --all-namespaces is -A

kubectl get all --all-namespaces

#### Daemonsets

**Daemonsets – A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.**

kubectl get daemonset – List one or more daemonsets.

kubectl edit daemonset <daemonset\_name> – Edit and update the definition of one or more daemonset.

kubectl delete daemonset <daemonset\_name> – Delete a daemonset.

kubectl create daemonset <daemonset\_name> – Create a new daemonset.

kubectl rollout daemonset – Manage the rollout of a daemonset.

kubectl describe ds <daemonset\_name> -n <namespace\_name> – Display the detailed state of daemonsets within a namespace.

#### Deployments

**Deployments – A Deployment provides declarative updates for Pods and ReplicaSets. You describe a desired state in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments. See StatefulSet vs. Deployment.**

kubectl get deployment – List one or more deployments.

kubectl describe deployment <deployment\_name> – Display the detailed state of one or more deployments.

kubectl edit deployment <deployment\_name> – Edit and update the definition of one or more deployments on the server.

kubectl create deployment <deployment\_name> – Create a new deployment.

kubectl delete deployment <deployment\_name> – Delete deployments.

kubectl rollout status deployment <deployment\_name> – See the rollout status of a deployment.

kubectl set image deployment/<deployment name> <container name>=image:<new image version> – Perform a rolling update (K8S default), set the image of the container to a new version for a particular deployment.

kubectl rollout undo deployment/<deployment name> – Rollback a previous deployment.

kubectl replace --force -f <configuration file> – Perform a replace deployment — Force replace, delete and then re-create the resource.

#### Events

kubectl get events – List recent events for all resources in the system.

kubectl get events --field-selector type=Warning – List Warnings only.

kubectl get events --sort-by=.metadata.creationTimestamp – List events sorted by timestamp.

kubectl get events --field-selector involvedObject.kind!=Pod – List events but exclude Pod events.

kubectl get events --field-selector involvedObject.kind=Node, involvedObject.name=<node\_name> – Pull events for a single node with a specific name.

kubectl get events --field-selector type!=Normal – Filter out normal events from a list of events.

#### Logs

**Logs – System component logs record events happening in cluster, which can be very useful for debugging. You can configure log verbosity to see more or less detail. Logs can be as coarse-grained as showing errors within a component, or as fine-grained as showing step-by-step traces of events (like HTTP access logs, pod state changes, controller actions, or scheduler decisions).**

kubectl logs <pod\_name> – Print the logs for a pod.

kubectl logs --since=6h <pod\_name> – Print the logs for the last 6 hours for a pod.

kubectl logs --tail=50 <pod\_name> – Get the most recent 50 lines of logs.

kubectl logs -f <service\_name> [-c <$container>] – Get logs from a service and optionally select which container.

kubectl logs -f <pod\_name> – Print the logs for a pod and follow new logs.

kubectl logs -c <container\_name> <pod\_name> – Print the logs for a container in a pod.

kubectl logs <pod\_name> pod.log – Output the logs for a pod into a file named ‘pod.log’.

kubectl logs --previous <pod\_name> – View the logs for a previously failed pod.

#### Nodes

**Nodes – Kubernetes runs your workload by placing containers into Pods to run on Nodes. A node may be a virtual or physical machine, depending on the cluster. Each node is managed by the control plane and contains the services necessary to run Pods. Typically you have several nodes in a cluster; in a learning or resource-limited environment, you might have only one node. The components on a node include the kubelet, a container runtime, and the kube-proxy.**

kubectl taint node <node\_name> – Update the taints on one or more nodes.

kubectl get node – List one or more nodes.

kubectl delete node <node\_name> – Delete a node or multiple nodes.

kubectl top node <node\_name> – Display Resource usage (CPU/Memory/Storage) for nodes.

kubectl get pods -o wide | grep <node\_name> – Pods running on a node.

kubectl annotate node <node\_name> – Annotate a node.

kubectl cordon node <node\_name> – Mark a node as unschedulable.

kubectl uncordon node <node\_name> – Mark node as schedulable.

kubectl drain node <node\_name> – Drain a node in preparation for maintenance.

kubectl label node – Add or update the labels of one or more nodes.

#### Pods

**Pods – Pods are the smallest deployable units of computing that you can create and manage in Kubernetes. A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers. A Pod’s contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific “logical host”: it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host. As well as application containers, a Pod can contain init containers that run during Pod startup. You can also inject ephemeral containers for debugging if your cluster offers this**.

kubectl get pod – List one or more pods.

kubectl get pods --sort-by='.status.containerStatuses[0].restartCount' – List pods Sorted by Restart Count.

kubectl get pods --field-selector=status.phase=Running – Get all running pods in the namespace.

kubectl delete pod <pod\_name> – Delete a pod.

kubectl describe pod <pod\_name> – Display the detailed state of a pods.

kubectl create pod <pod\_name> – Create a pod.

kubectl exec <pod\_name> -c <container\_name> <command> – Execute a command against a container in a pod. Read more: Using Kubectl Exec: Connect to Your Kubernetes Containers

kubectl exec -it <pod\_name> /bin/sh – Get an interactive shell on a single-container pod.

kubectl top pod – Display Resource usage (CPU/Memory/Storage) for pods.

kubectl annotate pod <pod\_name> <annotation> – Add or update the annotations of a pod.

kubectl label pods <pod\_name> new-label=<label name> – Add or update the label of a pod.

kubectl get pods --show-labels – Get pods and show labels.

kubectl port-forward <pod name> <port number to listen on>:<port number to forward to> – Listen on a port on the local machine and forward to a port on a specified pod.

#### Secrets

Secrets – A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don’t need to include confidential data in your application code.

kubectl create secret – Create a secret.

kubectl get secrets – List secrets.

kubectl describe secrets – List details about secrets.

kubectl delete secret <secret\_name> – Delete a secret.

#### Services

**Services – An abstract way to expose an application running on a set of Pods as a network service. With Kubernetes you don’t need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them.**

kubectl get services – List one or more services.

kubectl describe services – Display the detailed state of a service.

kubectl expose deployment [deployment\_name] – Expose a replication controller, service, deployment, or pod as a new Kubernetes service.

kubectl edit services – Edit and update the definition of one or more services.