# Intro

- Kubernetes is an open-source **container orchestration** tool developed by Google
- It helps us manage containerized applications in different deployment environments

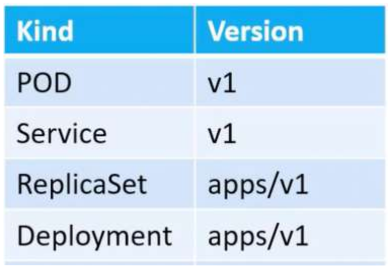
## Container Orchestration

- The automated deployment and management of containers is called **container orchestration**
- Containers act as perfect hosts for micro-services (running independently)
- The rise of micro-services architecture led to applications using 1000s of containers that need to be smartly managed.
- Container orchestration offers:
    - **High Availability** (no downtime)
    - **Horizontal Scalability**
    - **Disaster Recovery**

# YAML Definition File

```
apiVersion: v1 kind: Pod metadata: name: myapp-pod labels: app: myapp type: f
ront-end annotations: # misc info (not used by K8s) podVersion: 1.2 spec: con
tainers: - name: nginx-container image: nginx
```

- YAML config file to create and maintain objects in a Kubernetes cluster
- It has 4 top level required fields
    - `apiVersion` - string
        - depending upon the kind of K8s object being created, use the right API version



| Kind | Version |
|---|---|
| POD | v1 |
| Service | v1 |
| ReplicaSet | apps/v1 |
| Deployment | apps/v1 |

    - `kind` - the kind of K8s object being created (string)
    - `metadata` - metadata about the K8s object being created (dictionary)
        - `name` - name of the K8s object (string)
        - `labels` - key-value pairs for grouping K8s objects after creation (dictionary)
        - `annotations` - used to specify misc information not directly used by K8s but could be used by other tools running on the k8s cluster (dictionary)
    - `spec` - specifications (dictionary)
        - specific to the K8s object (need to refer the docs)
- Once you have the YAML file, you can deploy the K8s object by `kubectl apply -f <filename>.yaml`

- To override the `Dockerfile` `CMD`, specify the command as `args` in JSON array format. Similarly, to override the `ENTRYPOINT`, specify the new docker entrypoint as `command` in the config file.

  Example:

```
FROM Ubuntu ENTRYPOINT sleep
CMD 5
```

Dockerfile

```
apiVersion: v1 kind: Pod metadat
a: name: ubuntu-sleeper-pod spe
c: containers: - name: ubuntu-sl
eeper image: ubuntu-sleeper comm
and: ['sleeper'] args: ['100']
```

pod.yaml

# Kubectl Commands

- `kubectl` can be replaced with `k` by using an alias
- `kubectl` makes calls to the `kube-apiserver` behind the scenes

## Get or List Resources

- List all K8s objects - `k get all`
- List nodes - `k get nodes`
- List pods - `k get pods`
- List pods with additional info - `k get pods -o wide`
- List services - `k get services` or `k get svc`
- List deployments - `k get deployments`
- List replica sets - `k get replicasets`
- List replica sets with additional information - `k get replicasets -o wide`
- List deployments in another namespace - `k get deployments --namespace=<namespace>`
- List pods in all the namespaces - `k get pods --all-namespaces`
- List service accounts - `k get serviceaccount`
- List pods with label filter - `k get pod --selector key=value`
- Get pods with multiple label filters - `k get pods --selector env=prod,bu=finance,tier=frontend`
- List all jobs - `k get jobs`
- List all cron jobs - `k get cronjobs`
- List all network policies - `k get networkpolicies`
- List all ingress resources - `k get ingress`
- List persistent volumes - `k get persistentvolume`
- List persistent volume claims - `k get persistentvolumeclaim`
- List all DaemonSets - `k get daemonsets`
- List cluster events - `k get events -o wide`

- Get what node each pod is deployed on - `k get pods -o wide`

## Describe

- Describe a pod - `k describe pod <pod-name>`
- Describe a deployment - `k describe deployment <deployment-name>`
- Describe a service account - `k describe serviceaccount <service-account-name>`
- Describe an ingress - `k describe ingress <ingress-name>`
- Describe a DaemonSet - `k describe daemonset <daemonset-name>`

## Create

- Create a pod from an image - `k run <pod-name> --image <image-name>`
- Apply a K8s config.yaml file (declarative) - `k apply -f <filename>.yaml`
- Create a namespace - `k create namespace <namespace>`
- Create service account - `k create serviceaccount <service-account-name>`

## Delete

- Delete a pod - `k delete pod <pod-name>`
- Delete a replica set - `k delete replicaset <replicaset-name>`
- Delete a deployment - `k delete deployment <deployment-name>`
- Delete a K8s resource from a config file - `k delete -f config.yaml`

## Logs

- View logs of a pod - `k logs <pod-name>`
- View live logs of a pod - `k logs -f <pod-name>`
- View logs of a container in a multi-container pod (must specify the container name, otherwise the command will fail) - `k logs <pod-name> <container-name>`

- View logs of the previous container if the pod failed - `k logs <pod-name> --previous`

## Rollouts and Update

- `k rollout status deployment/<deployment-name>` - view the status of rollout for a deployment
- `k rollout history deployment/<deployment-name>` - view the history of rollouts for a deployment
- `k rollout history deployment/<deployment-name> --revision=1` - view the history of rollouts for a deployment revision
- `k rollout undo deployment/<deployment-name>` - rollback a deployment
- `k rollout undo deployment/<deployment-name> --to-revision=1` - rollback a deployment to a specific revision

## Node Commands

- Cordon a node (mark it unschedulable) - `k cordon <node-name>`
- Un-cordon a node (mark it schedulable) - `k uncordon <node-name>`
- Drain a node of all the pods - `k drain <node-name>`
- Drain a node of all the pods except DaemonSets - `k drain <node-name> --ignore-daemonsets`

## Auth

- View roles - `k get roles`
- View role bindings - `k get rolebindings`
- Check if you have access to perform an operation in K8s:
  - Create deployments - `k auth can-i create deployments`
  - Delete nodes in dev ns - `k auth can-i delete nodes -n dev`

- Check if another user has access to perform an operation in K8s:
  - Create deployments - `k auth can-i create deployments --as dev-user`
  - Delete nodes in dev namespace - `k auth can-i delete nodes --as dev-user -n dev`
- Check if a service account has access to perform an operation in K8s - `k auth can-i get pods --as=system:serviceaccount:default:<serviceaccount-name>`

## Config

- List all clusters - `k config get-clusters`
- Set context for a cluster - `k config use-context <cluster-name>`

### Certificates

- List all CSR - `k get csr`
- View a CSR in YAML format - `k get csr <csr-name> -o yaml`
- Delete a CSR - `k delete csr <csr-name>`
- Approve a CSR - `k certificate approve <csr-name>`
- Deny a CSR - `k certificate deny <csr-name>`

## Output formats

`-o json` - json output

`-o name` - only the resource name

`-o wide` - extra info in plain text

`-o yaml` - yaml output

## Useful Imperative Commands

- Create a pod - `k run <pod-name> --image=<image-name> --command -- <command-with-args>`

- Create a deployment - `k create deployment <deployment-name> --image <image-name> --replicas 3`
- Create a service for a deployment - `k expose deployment <deployment-name> --port 80 --name <service-name>`
- Create a service for a pod - `k expose pod <pod-name> --port 80 --name <service-name>`
- Edit a deployment (in memory) - `k edit deployment <deployment-name>`
- Scale a deployment - `k scale deployment <deployment-name> --replicas 3`
- Update the image in a deployment - `k set image deployment <deployment-name> <old-image>=<new-image>`
- Create resources from a config file - `k create -f filename.yaml`
- Update resources from a config file - `k replace -f filename.yaml`
- Delete and recreate resources from a config file - `k replace --force -f filename.yaml`
- Edit a deployment (doesn't work for pods as they require recreation) - `k edit deployment <deployment-name>`

## Handy Commands

- Get the config YAML of a pod - `k get pod <pod-name> -o yaml > pod.yaml`
- Get the config YAML of a deployment - `k get deployment <deployment-name> -o yaml > deployment.yaml`
- Create a sample config YAML file for a pod - `k run <pod-name> --image <image-name> --dry-run=client -o yaml > filename.yaml`
- Create a sample config YAML file for a deployment - `k create deployment <deployment-name> --image <image-name> --dry-run=client -o yaml > filename.yaml`
- Create a sample config YAML file for a service - `k expose deployment <deployment-name> --port 80 --dry-run=client -o yaml > filename.yaml`

## JSON Path in Kubectl

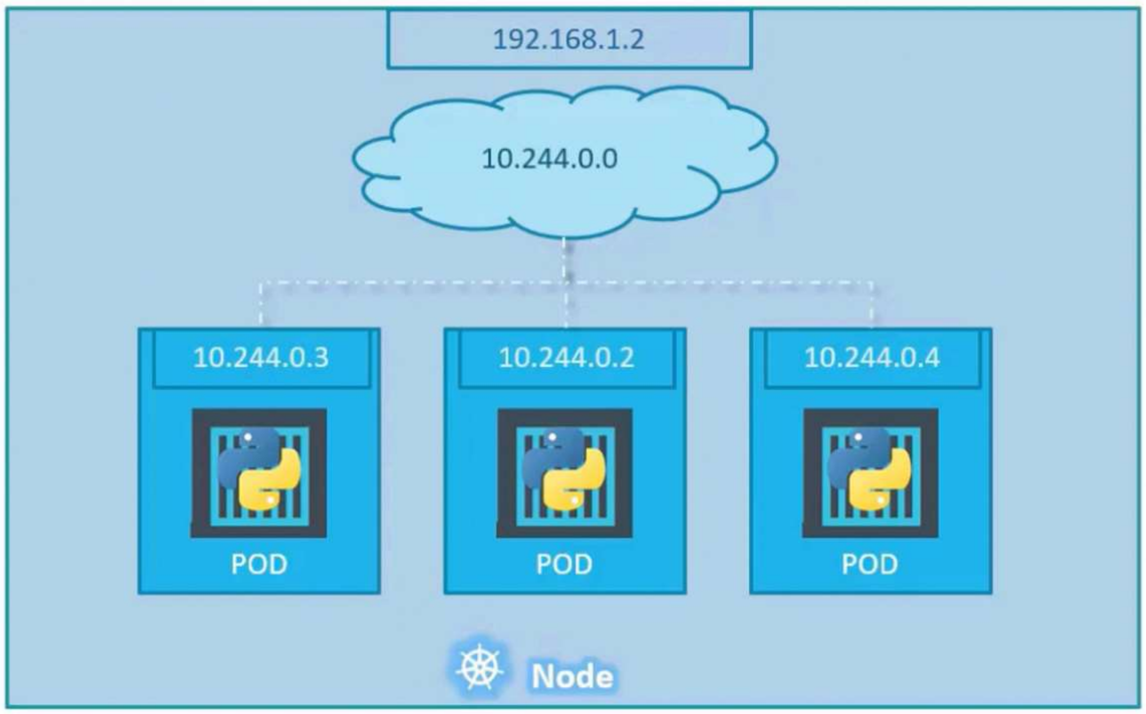
- Get images of all the pods - `k get pods -o jsonpath='{.items[0].spec.containers[0].image}'`

## Metrics

- Get top nodes by memory - `k top nodes --sort-by memory`
- Get top nodes by CPU for a context - `k --context <context> top nodes --sort-by cpu`

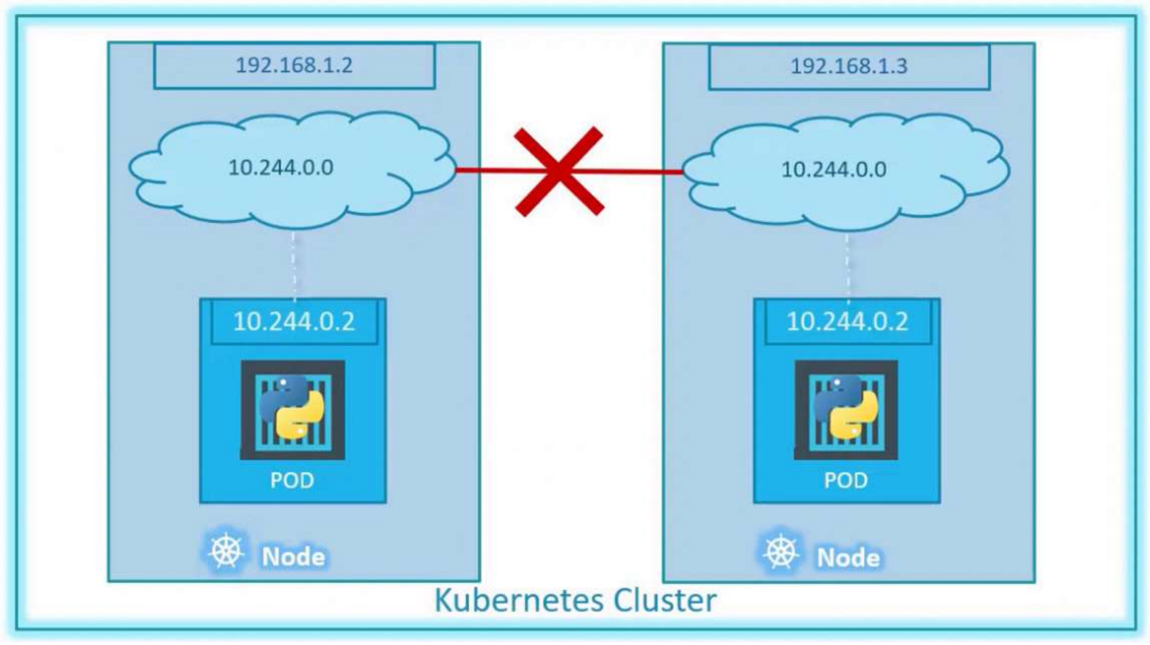
# Networking in Kubernetes

## Networking in a single node

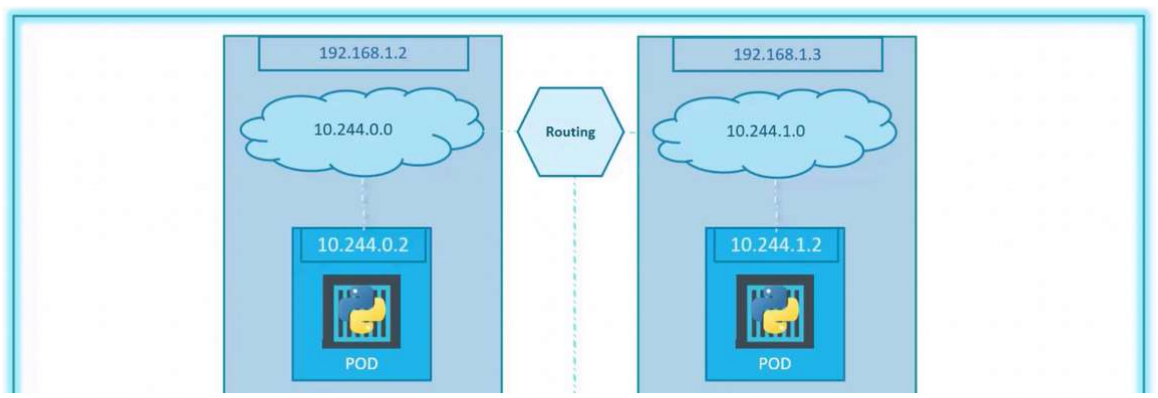


- When K8s is installed on a host, it creates an internal private network in the range `10.244.0.0/16` and every pod on the host gets an IP address on this network.
- The pods can communicate with each other using IP addresses but it's not a good idea as IP addresses change when pods are restarted.
- **Every node gets an IP address on the external network** (not the K8s internal network)

## Networking in a cluster



- When we have multiple nodes in the cluster, each of them have a unique IP in the external network.
- Each node sets up their own internal private network, which means they could be using overlapping CIDRs. This could lead to IP conflicts between pods. In this case, pods won't be able to communicate across nodes.
- When we setup a cluster, Kubernetes expects us to setup the networking to fulfill certain requirements:
    - All pods should be able to communicate with each other without configuring a NAT
    - All nodes should be able to communicate with all the pods and vice-versa without a NAT
- There are multiple third party networking solutions for K8s networking that we can use. This makes each node use a different CIDR for its internal private network thus allows each proper communication between the pods.


192.168.1.2
10.244.0.0
10.244.0.2
POD
Routing
192.168.1.3
10.244.1.0
10.244.1.2
POD

# Labels, Selectors and Annotations

Labels and Selectors allow us to label every K8s resource and later select them to apply an action.

- Get pods with label filter - `k get pod --selector key=value`
- Get pods with multiple label filters - `k get pods --selector env=prod,bu=finance,tier=frontend`

## Labels in Pod

```
apiVersion: v1 kind: Pod metadata: labels: name: frontend annotations: podVer
sion: 1.2 spec: containers: - name: httpd image: httpd:2.4-alpine
```

## Labels in Deployment

Labels in the template section is used to label the Pods created by the deployment. These labels are then selected by the deployment to group these pods under the same deployment. If any other pod that was created before this deployment and it matches the label selector criteria, it will be controlled by this deployment too.

The labels for the deployment is not directly involved in labelling or selecting the pods.

```
apiVersion: apps/v1 kind: Deployment metadata: name: httpd-frontend labels: n
ame: frontend annotations: buildVersion: 1.2 spec: replicas: 3 selector: matc
hLabels: name: frontend template: metadata: labels: name: frontend spec: cont
ainers: - name: httpd image: httpd:2.4-alpine
```

## Annotations

**Annotations are used to provide miscellaneous information to the Pod.** They are not used by K8s internally but they can be used by other tools running on the K8s cluster.

```
apiVersion: networking.k8s.io/v1 kind: Ingress metadata: annotations: nginx.i
ngress.kubernetes.io/rewrite-target: / nginx.ingress.kubernetes.io/ssl-redire
ct: "false" name: ingress-pay namespace: critical-space spec: rules: - http:
paths: - backend: service: name: pay-service port: number: 8282 path: /pay pa
thType: Prefix
```