BITS Pilani

Cloud Computing

Session 5

Container Orchestration

Shwetha Vittal

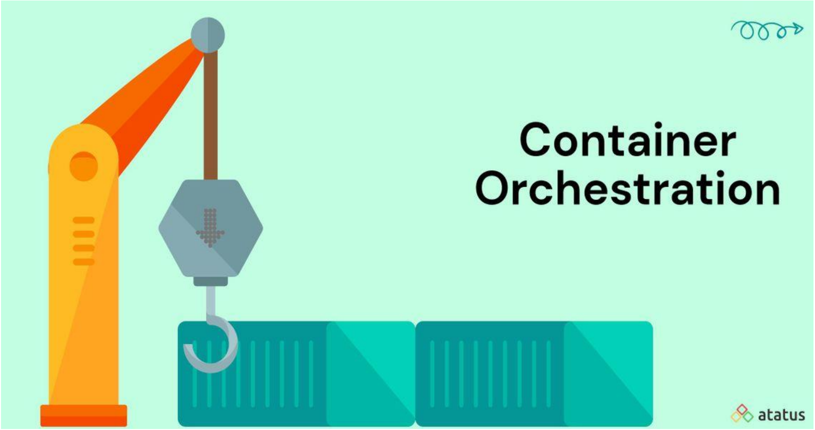
Agenda

- ❑ Need for Container Orchestration
- ❑ Kubernetes (K8s) Features
- ❑ K8s Objects
- ❑ Use cases of K8s
- ❑ What K8s is not?

Why Container Orchestration

Container

- Logically distinct piece of software
- Built, deployed, maintained, managed and, scaled on own without unduly affecting other parts of the system.



Container Orchestration

- Bigger system requires multiple containers
- Need interaction among themselves.
- E.g application servers need to interact with backend DB.
- Come together as a single scalable, reliable, and resilient system
- **Container orchestration** comes in for rescue
  - E.g. Kubernetes

Kubernetes – What is it ?

- Kubernetes (K8s) - An open source system for
  - Automating deployment: Dynamically pushing configuration files to running jobs;
  - Service discovery and load balancing
  - Auto-scaling
  - Management of containerized applications.

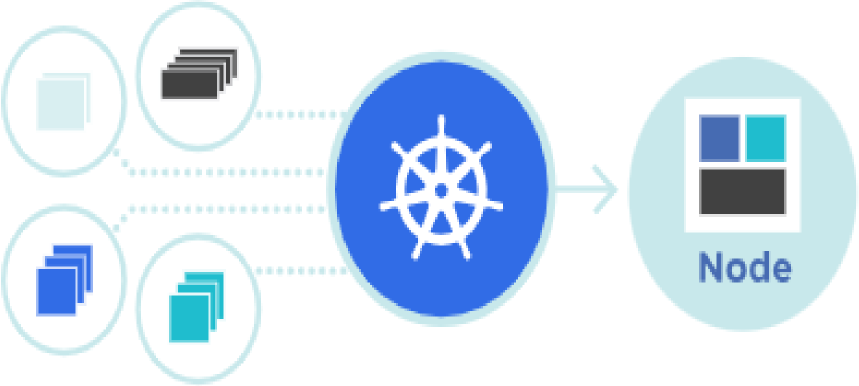


Image Courtesy: Kubernetes

15 years of experience of running production workloads at Google

Groups containers that make up an application into logical units for easy management and discovery.

Kubernetes – What is it ?

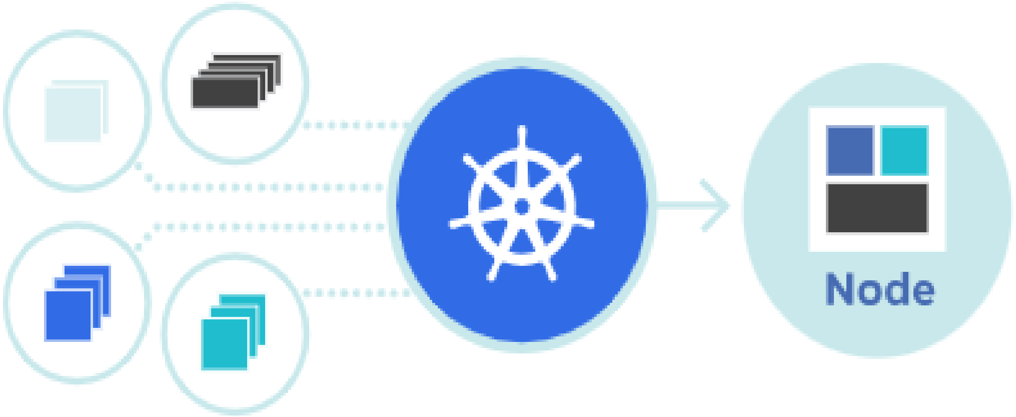


Image Courtesy: Kubernetes

- High Availability service:
  - K8s provides you with a framework to run distributed systems resiliently.
  - It takes care of scaling and failover for your application

Features of K8s

- **Automatic bin packing**
  - K8s can fit containers onto the specified nodes to make the best use of your resources.
- **Self-healing**
  - K8s restarts containers that fail, replaces containers, kills containers that don't respond to user-defined health check

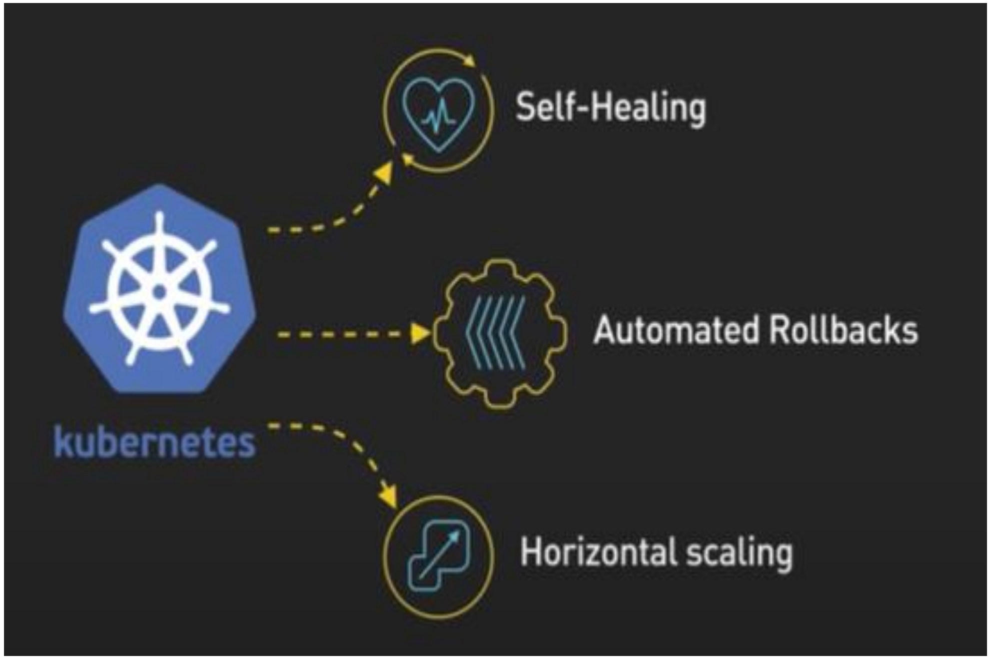


Image Courtesy: Bytebytego

Features of K8s

- **Service Discovery and Load balancing**
  - **Discovery**: Kubernetes can expose a container using the DNS name or using their own IP address.
  - **Load balancing**: If traffic to a container is high, K8s distributes the network traffic

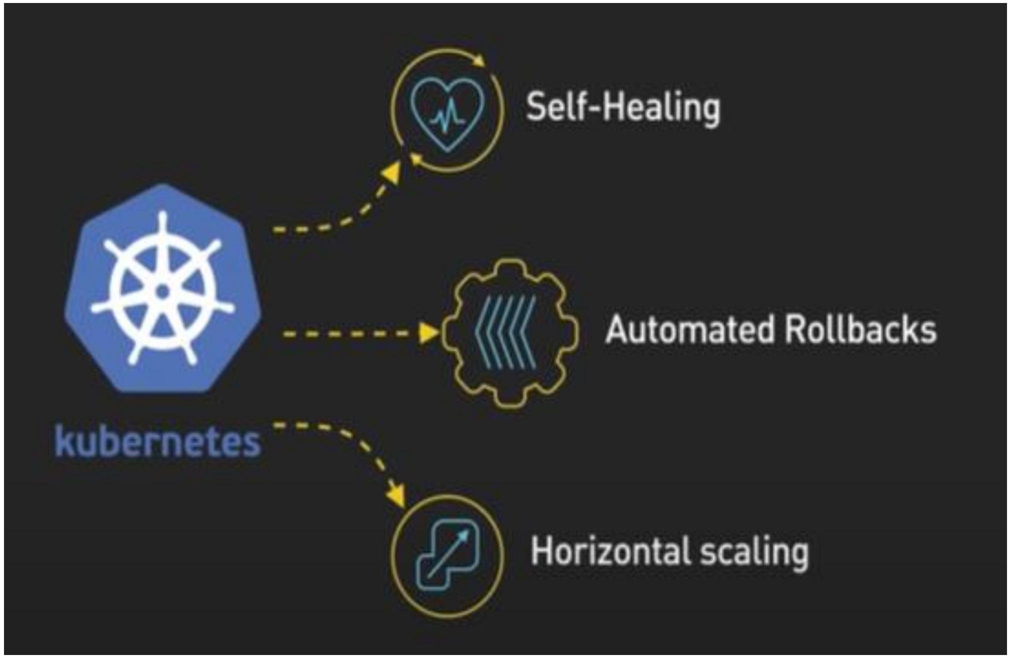


Image Courtesy: Bytebytego

Features of K8s

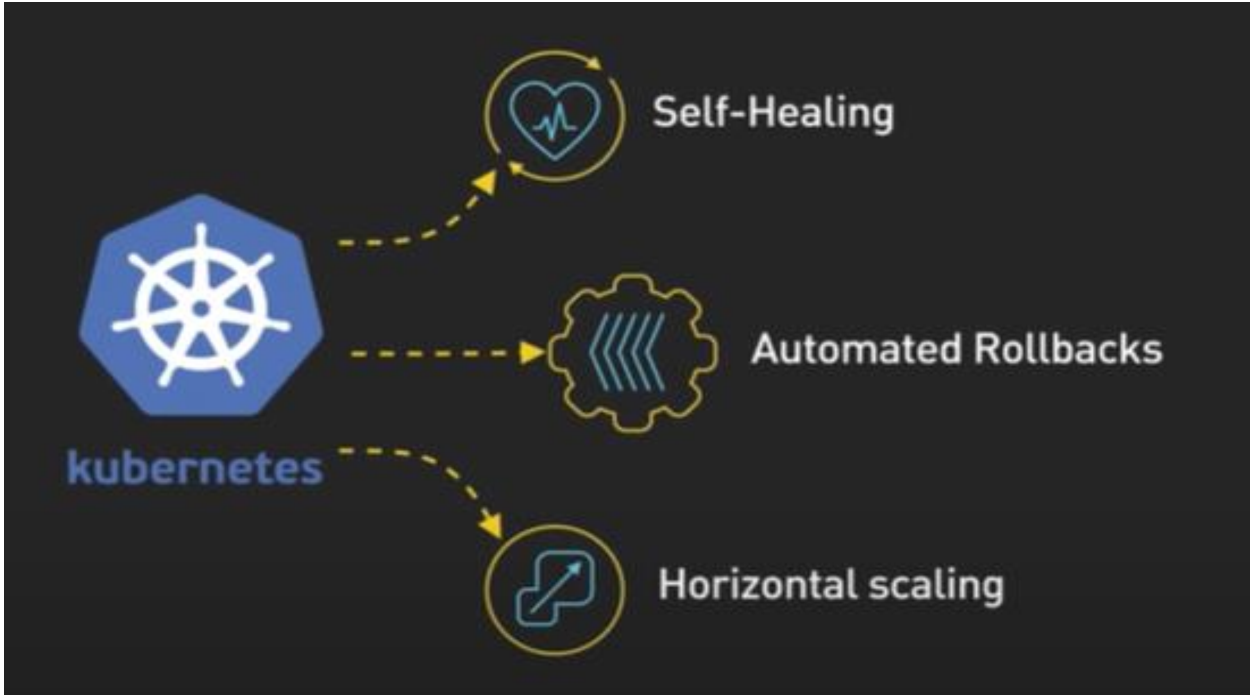


Image Courtesy: Bytebytego

Automated rollouts and rollbacks

- Change the actual state of the deployed container to the desired state at a controlled rate.
- E.g.: Remove existing containers and adopt all their resources to the new container.

Features of K8s

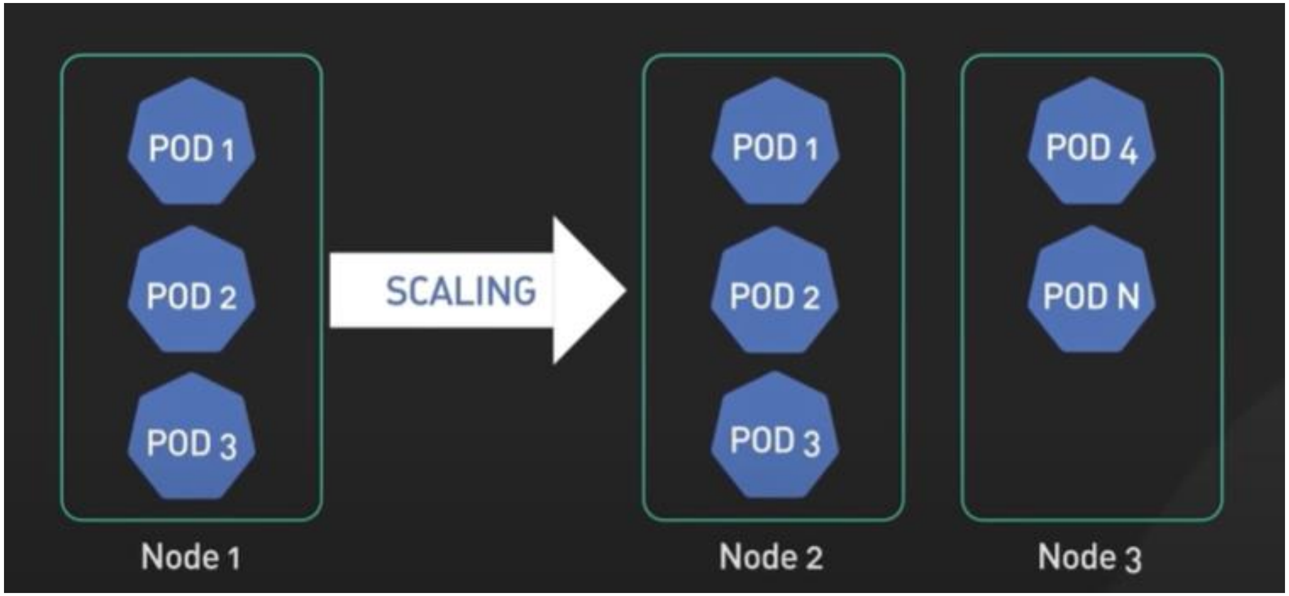


Image Courtesy: Bytebytego

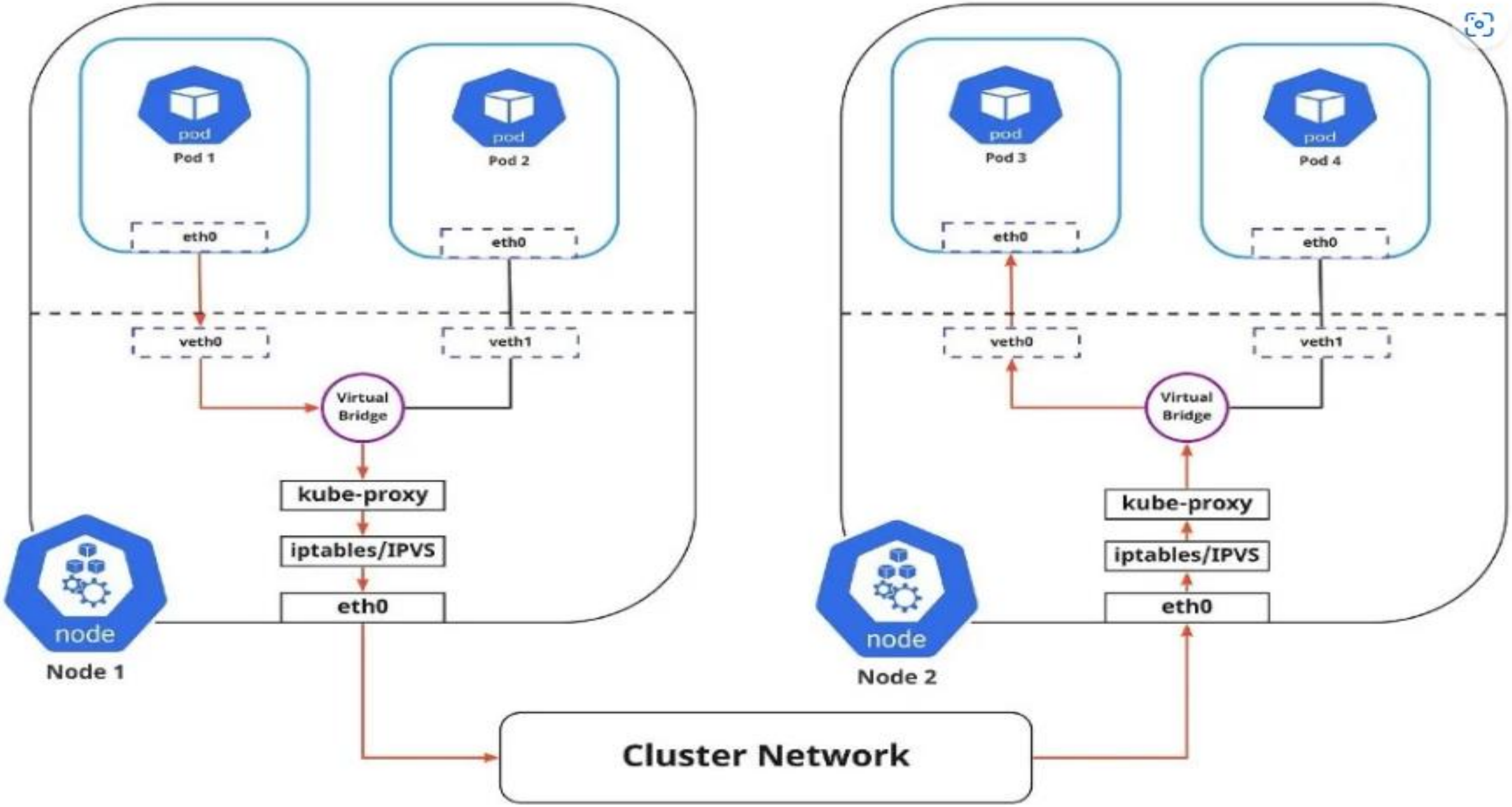
- **Horizontal scaling**
    - Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
- **Designed for extensibility**
    - Add features to cluster without changing upstream source code.

K8s Nodes & Cluster

- **Node**
  - A node is a host system, whether physical or virtual, with required components.
  - Allows an execution of one or more containers inside it.
  - Two types of Nodes
  1. Master
  2. Worker

- **Cluster**
  - A K8s cluster consists of one or more nodes.
  - K8s system components, that is connected to a network that allows it to reach other nodes in the cluster.

K8s Nodes and Cluster

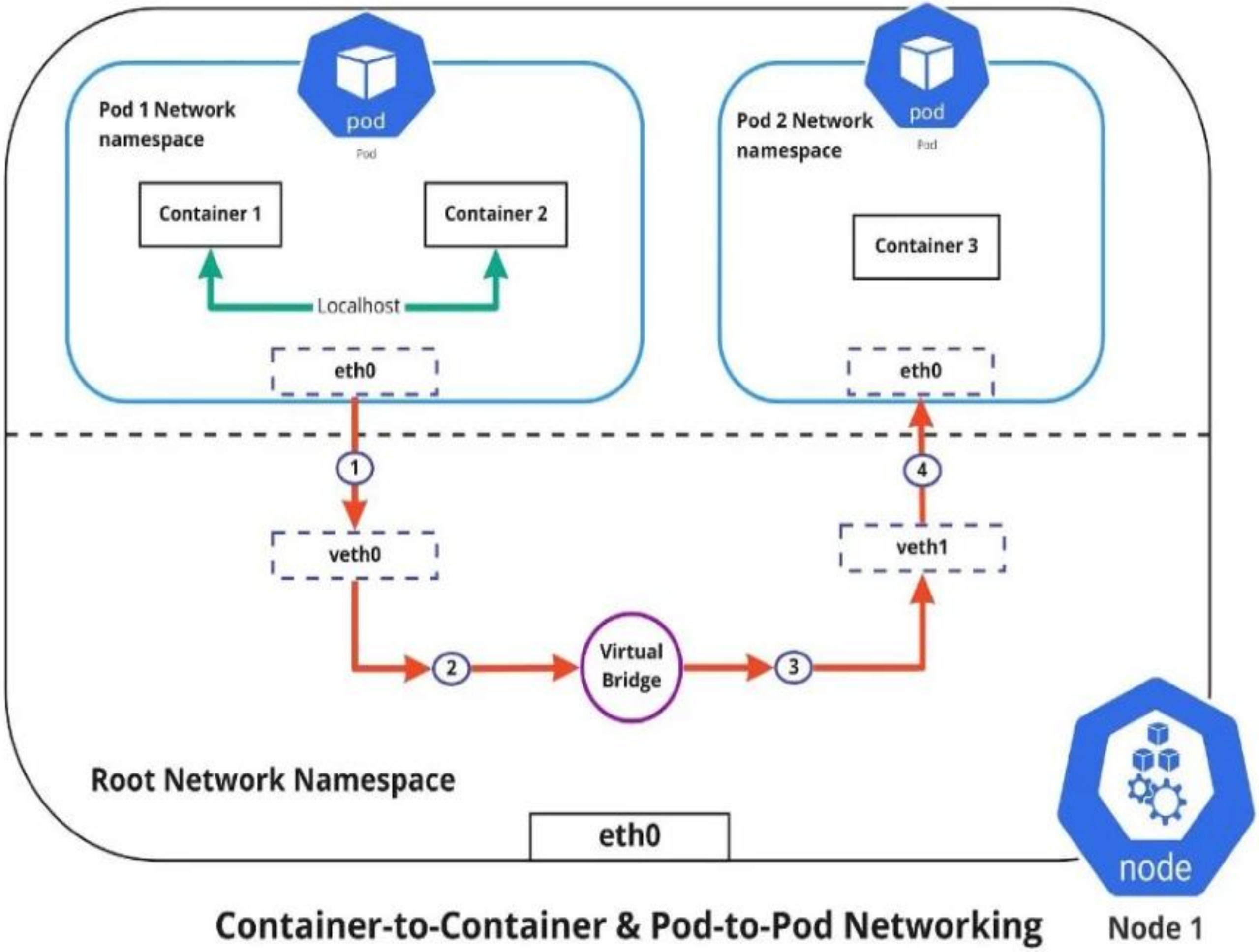


(Nived Velayudhan, CC BY-SA 4.0)

K8s Network Model

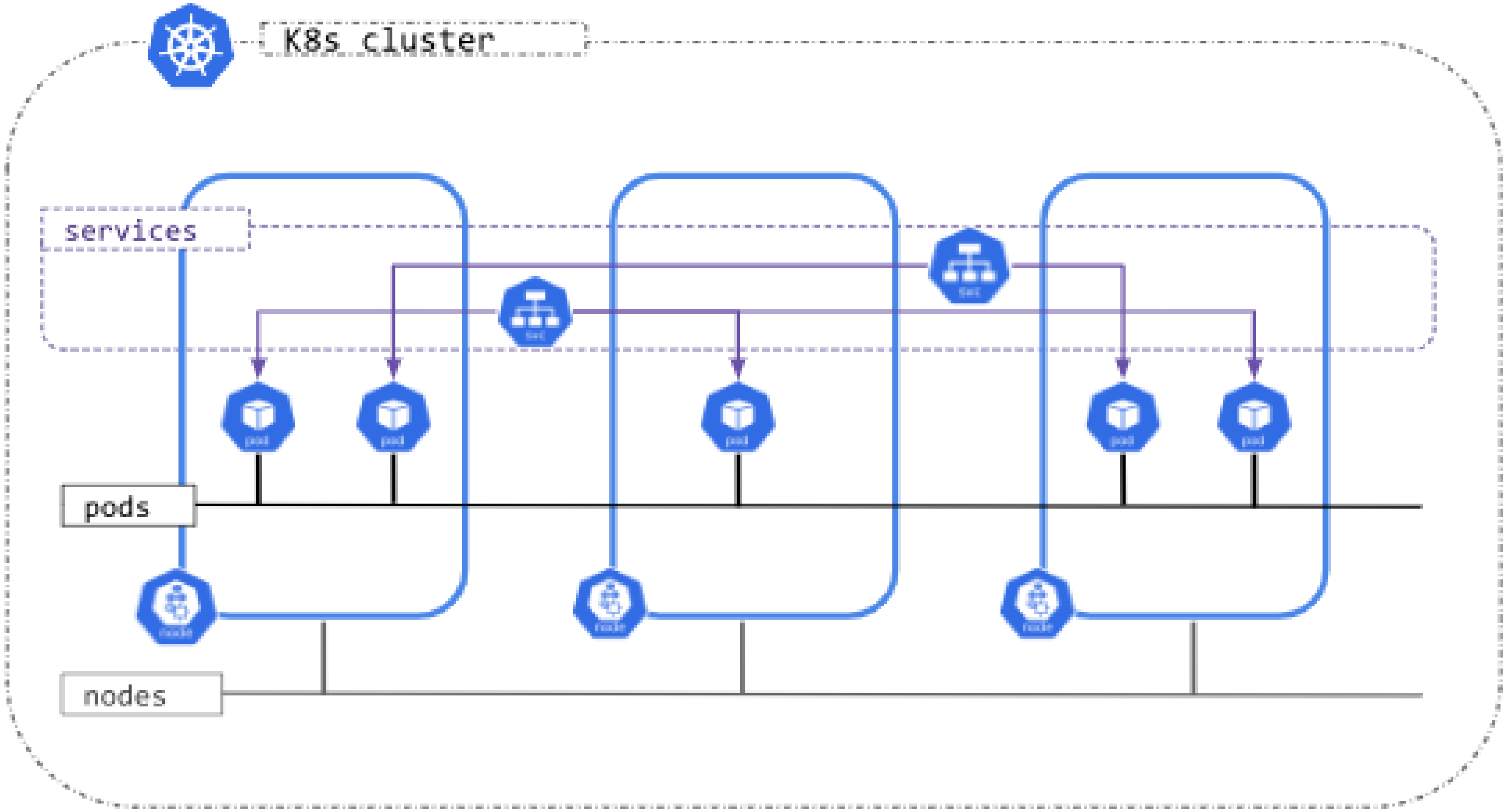
- **Pod**
  - Smallest deployable unit for executing back end code /application.
  - Gets its own unique cluster-wide IP address.
  - Has its own private network namespace which is shared by all of the containers within the pod.
  - The pod network handles communication between pods.

K8s Network Model



(Nived Velayudhan, CC BY-SA 4.0)

K8s Service



- Cluster
- Services
- Pods
- Nodes

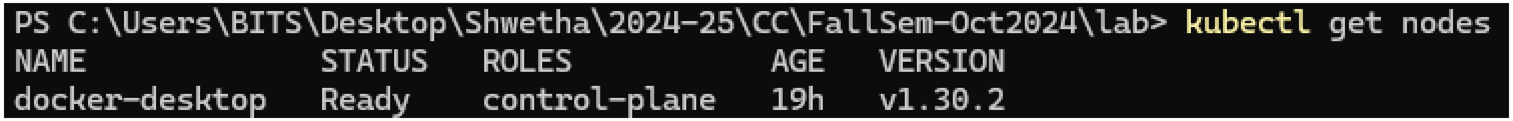
K8s Service

- Service is a method for exposing a network application that is running as one or more pods in the cluster.
- The Service API is an abstraction to expose groups of Pods over a network.
- Run application code in Pods

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

Kubectl Commands

- kubectl get nodes

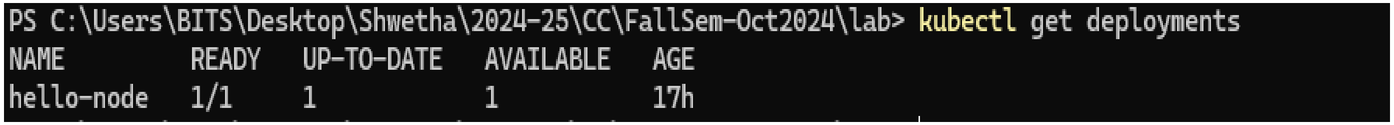


```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl get nodes
NAME             STATUS   ROLES           AGE   VERSION
docker-desktop   Ready    control-plane   19h   v1.30.2
```

- Status of node: Ready, NotReady, unknown
  - **Ready:** The node is running healthy and pods can be run inside.
  - **NotReady:** The node is not yet ready to run the pods
  - **Unknown:** If the node is not responding. If the master node cannot communicate with that node

Kubectl Commands

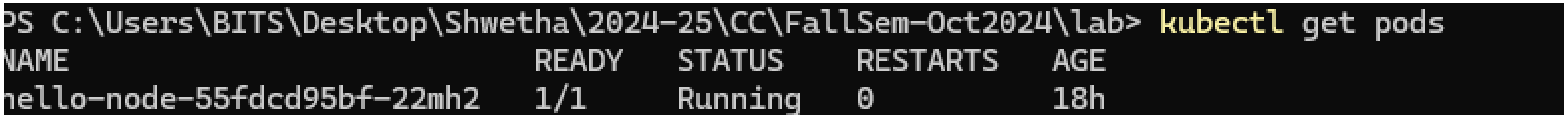
- kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.39 -- /agnhost netexec --http-port=8080
- kubectl get deployments



```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
hello-node   1/1     1            1           17h
```

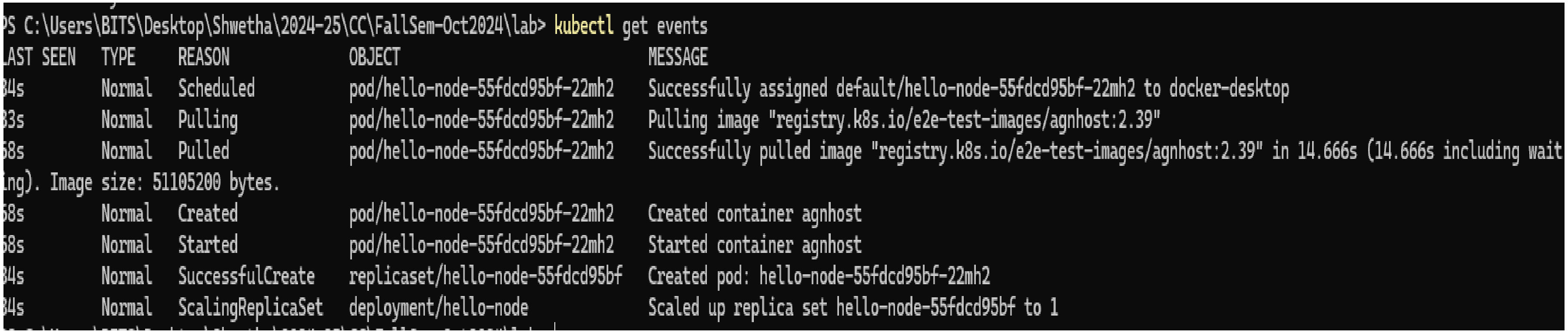
Kubectl Commands

kubectl get pods



```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-node-55fdcd95bf-22mh2   1/1     Running   0          18h
```

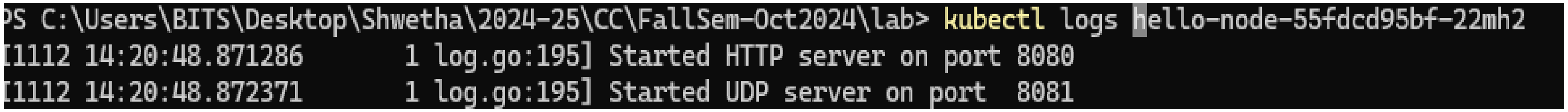
kubectl get events



```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl get events
LAST SEEN   TYPE     REASON              OBJECT                             MESSAGE
34s         Normal   Scheduled           pod/hello-node-55fdcd95bf-22mh2    Successfully assigned default/hello-node-55fdcd95bf-22mh2 to docker-desktop
33s         Normal   Pulling             pod/hello-node-55fdcd95bf-22mh2    Pulling image "registry.k8s.io/e2e-test-images/agnhost:2.39"
58s         Normal   Pulled              pod/hello-node-55fdcd95bf-22mh2    Successfully pulled image "registry.k8s.io/e2e-test-images/agnhost:2.39" in 14.666s (14.666s including wait
ing). Image size: 51105200 bytes.
58s         Normal   Created             pod/hello-node-55fdcd95bf-22mh2    Created container agnhost
58s         Normal   Started             pod/hello-node-55fdcd95bf-22mh2    Started container agnhost
34s         Normal   SuccessfulCreate    replicaset/hello-node-55fdcd95bf   Created pod: hello-node-55fdcd95bf-22mh2
34s         Normal   ScalingReplicaSet   deployment/hello-node              Scaled up replica set hello-node-55fdcd95bf to 1
```

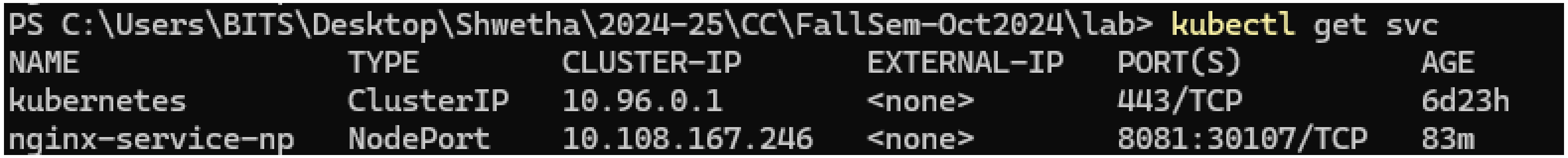
Kubectl Commands

- Kubectl get logs <pod name>



```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl logs hello-node-55fdcd95bf-22mh2
I1112 14:20:48.871286       1 log.go:195] Started HTTP server on port 8080
I1112 14:20:48.872371       1 log.go:195] Started UDP server on port  8081
```

- Kubectl get services



```
PS C:\Users\BITS\Desktop\Shwetha\2024-25\CC\FallSem-Oct2024\lab> kubectl get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP          6d23h
nginx-service-np   NodePort    10.108.167.246   <none>        8081:30107/TCP   83m
```

YML Specification

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

YML Specification

```
name: deploy a web server
   k8s:
    api_version: v1
    namespace: my-namespace
    definition:
      kind: Deployment
        metadata:
        labels:
        app: nginx
        name: nginx-deploy
        spec:
        replicas: 1
        selector:
        matchLabels:
        app: nginx
        template:
        metadata:
        labels:
        app: nginx
```

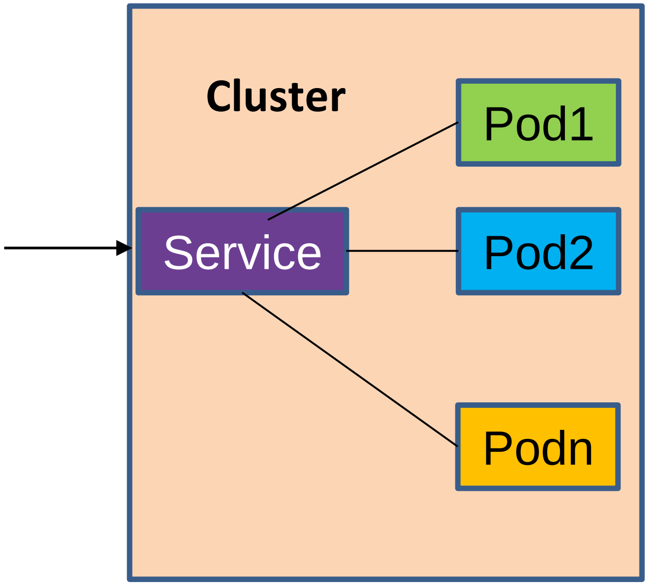
YML Specification

```
apiVersion: apps/v1 # Specifies the API version for the deployment resource.
kind: Deployment # Declares the type of Kubernetes object being created; in this case, a Deployment.
metadata:
name: nginx-test-deployment # The name of the deployment. Used for identification within the cluster.
labels: # Labels are key-value pairs for organizing and selecting resources.
app: nginx # Label with key 'app' and value 'nginx'. Helps in identifying related resources.
spec:
replicas: 1 # Specifies the number of desired pod replicas for this deployment.
selector: # Defines how the Deployment finds which Pods to manage.
matchLabels:
app: nginx # The Deployment will manage Pods with this label.
template: # Template section defines the Pod specification for the deployment.
metadata:
labels:
app: nginx # Labels assigned to Pods created by this Deployment.
spec: # The Pod specification, which defines the containers and other settings.
containers: # List of containers to be run in the Pod.
- name: nginx # The name of the container, useful for identifying the container within the Pod.
image: nginx:latest # The container image to be used. 'nginx:latest' specifies the latest version of nginx.
ports:
- containerPort: 80 # The port that this container exposes. It maps to port 80 inside the container.
```

Note: This file is placed for understanding the terminology only. Does not have indentation correct. Hence, cannot be used with K8s apply/create to create the deployment

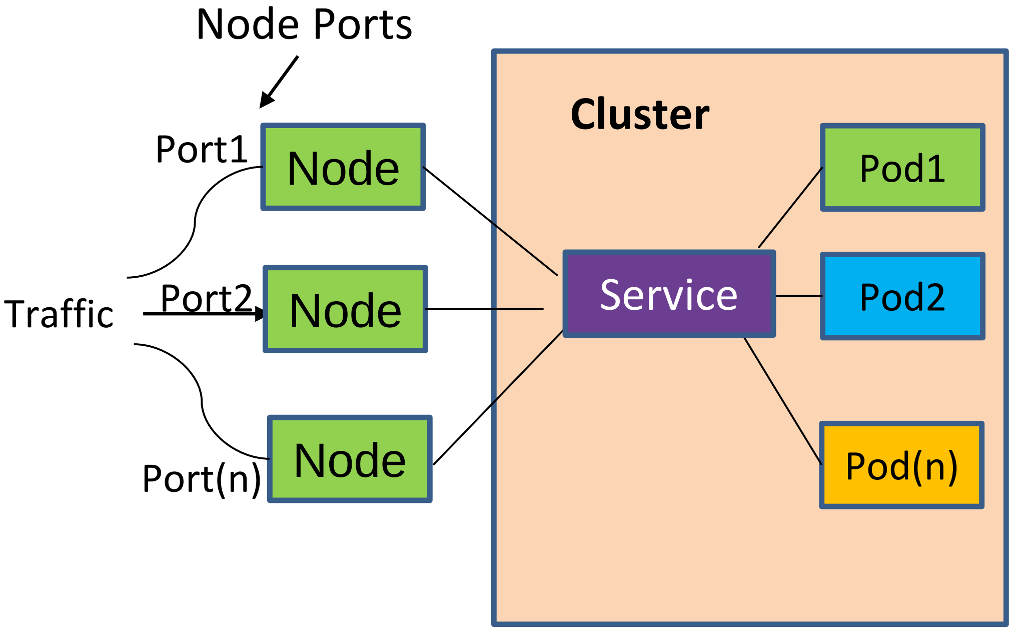
K8s Service Types – Cluster IP

- Default Service Type
- Internal Communication – Inside Cluster
- Exposes an internal IP within the cluster
- No external access allowed

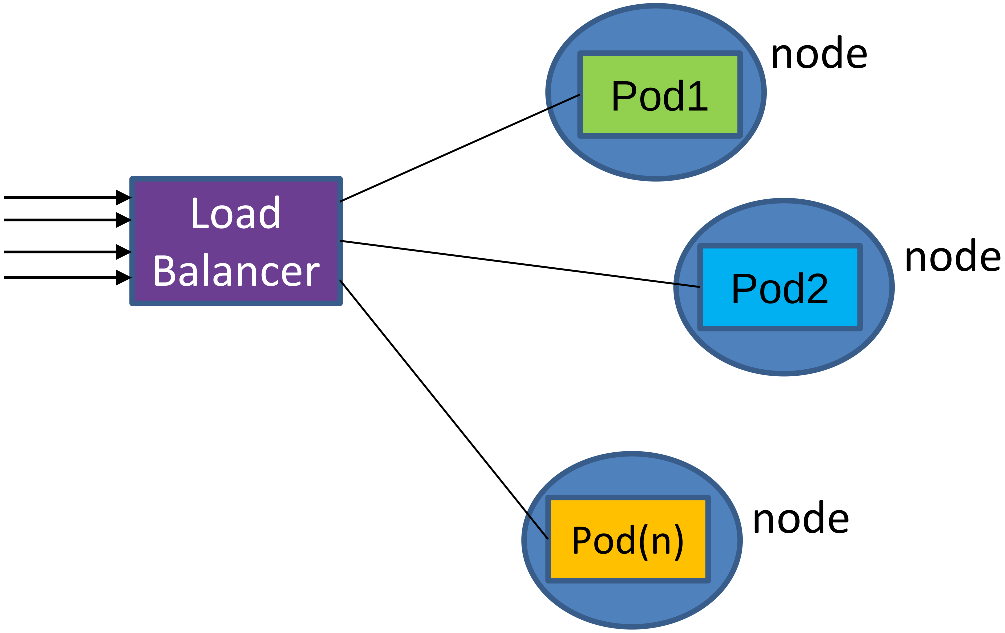


K8s Service Types – NodePort

- Exposes a specific application to the outside world with a port on the node
- External access to the service
- Forwards the traffic from the specified port on each node to the corresponding port on the pods targeted by the service.
- When application are to be accessed from outside the cluster without requiring a load balancer
- Note: One service per port
- Node port is static – 30000 to 32767



K8s Service Types – Load Balancer



- Provides external communication
- Exposes the application to the outside world – public IP
- Provides load balancing functions too
- E.g.: Web server

Use cases of K8s

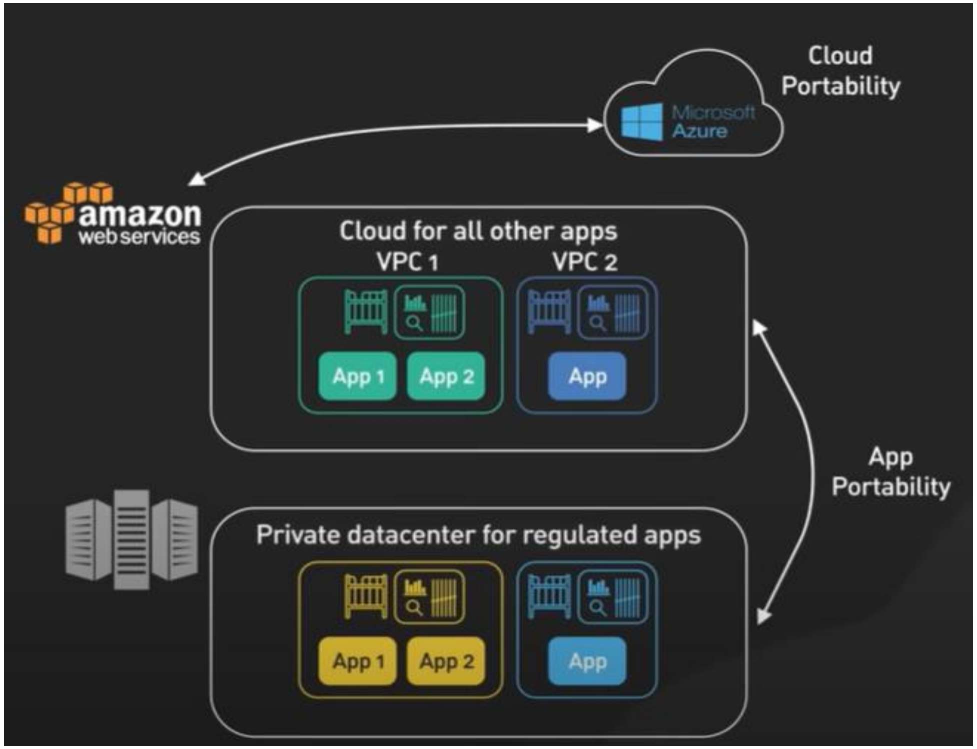


Image Courtesy: Bytebytego

- **Deployment and orchestration of microservices**
  - Ensures seamless deployment and communication.

Use cases of K8s

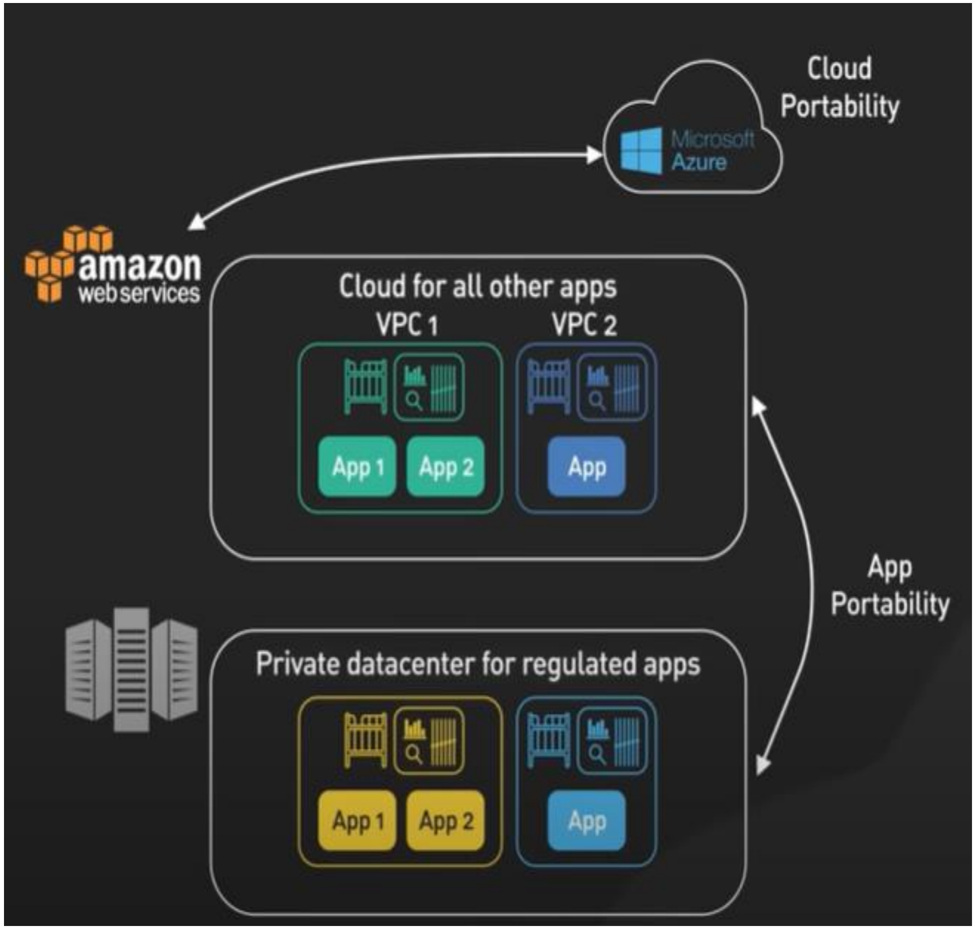


Image Courtesy: Bytebytego

- **Resource Optimization**
  - Dynamic allocation of resources based on workload
  - Maximizing system utilization
  - Reduces operational costs.
- **CI/CD Pipelines**
  - By automating deployment processes, K8s aids continuous integration and delivery.

What K8s is not

- Does not deploy source code and does not build your application
- Does not provide application-level services
- Does not dictate logging, monitoring, or alerting solutions

Summary

- Need for Container Orchestration
- K8s Orchestration Features & Functionalities
- K8s Objects – Nodes, Cluster, Pods, Deployments, Services
- K8s Service Types – ClusterIP, NodePort, LoadBalancer

Refer: Overview | Kubernetes

Lab – nginx Service on K8s Cluster

Deploy nginx web service on https://labs.play-with-k8s.com/

1. Create K8s cluster with two nodes
2. Experiment with deployment and different service types with two nodes on a K8s.
3. Capture all your observations and discuss with peers and instructor.

nginx deployment – nginx.yml
Nginx service as ClusterIP – clusteripservice.yml
Nginx service as NodePort – nodeport.yml
Nginx service as Loadbalancer – lb.yml

Lab – K8s Cluster creation with Two Nodes

Get into https://labs.play-with-k8s.com/

Create two instances here

On the first instance – (master node)

1. Initializes cluster master node:

```
kubeadm init --apiserver-advertise-address $(hostname -i) --pod-network-cidr 10.5.0.0/16
```

2. Initialize cluster networking:

```
kubectl apply -f https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/daemonset/kubeadm-kuberouter.yaml
```

3. Create an nginx deployment:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/nginx-app.yaml
```

On the second instance terminal (worker node)

Then you can join any number of worker nodes by running the following on each as root:

```
kubeadm join 192.168.0.18:6443 --token 4goo7z.k6laq8hufybdrnen \
  --discovery-token-ca-cert-hash sha256:2c4b669dcab8e1342befb6a22ff882dea58a501f5a8b13d5aa260f7f5a3298fd
```

nginx Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.23.3 #latest version on DockerHub
          ports:
            - containerPort: 80
```

```
kubectl apply –f nginx.yml

kubectl get deployments

kubectl get pods

kubectl logs <pod name>
```

nginx clusterIP Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
      app: nginx
  ports:
      - protocol: TCP
        port: 8081 -> Service accessible on 8081
        targetPort: 80 -> Container inside the pod uses port 80
```

Kubectl apply –f clusteripservice.yml

Kubectl get svc

Note the cluster IP from the output

Go to other instance and execute curl <cluster ip of the service>: 8081

nginx NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-np
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8081 Service accessible on 8081
      targetPort: 80 -> Container inside the pod uses port 80
      nodePort: 32002   ->Static port assigned to the node
```

Kubectl apply –f nodeport.yml

Kubectl get svc

Note the cluster IP from the output

Go to other instance and give
curl <cluster ip of the service>: 8081

Also
Curl <node ip>:32002

nginx LoadBalancer Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service-lb
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 80
      nodePort: 32002
```

Kubectl apply –f lb.yml

Kubectl get svc

Note the cluster IP from the output

Go to other instance and give
curl <cluster ip of the service>: 8081

Also
Curl <node ip>:32002

What additional things can you do on this service ?

Additional Slides

Additional Features of K8s

- **Configuration and Secret management**
  - Store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys.
  - Deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

- **Batch Execution**
  - Manage batch and CI workloads

K8s Architecture – Control Plane

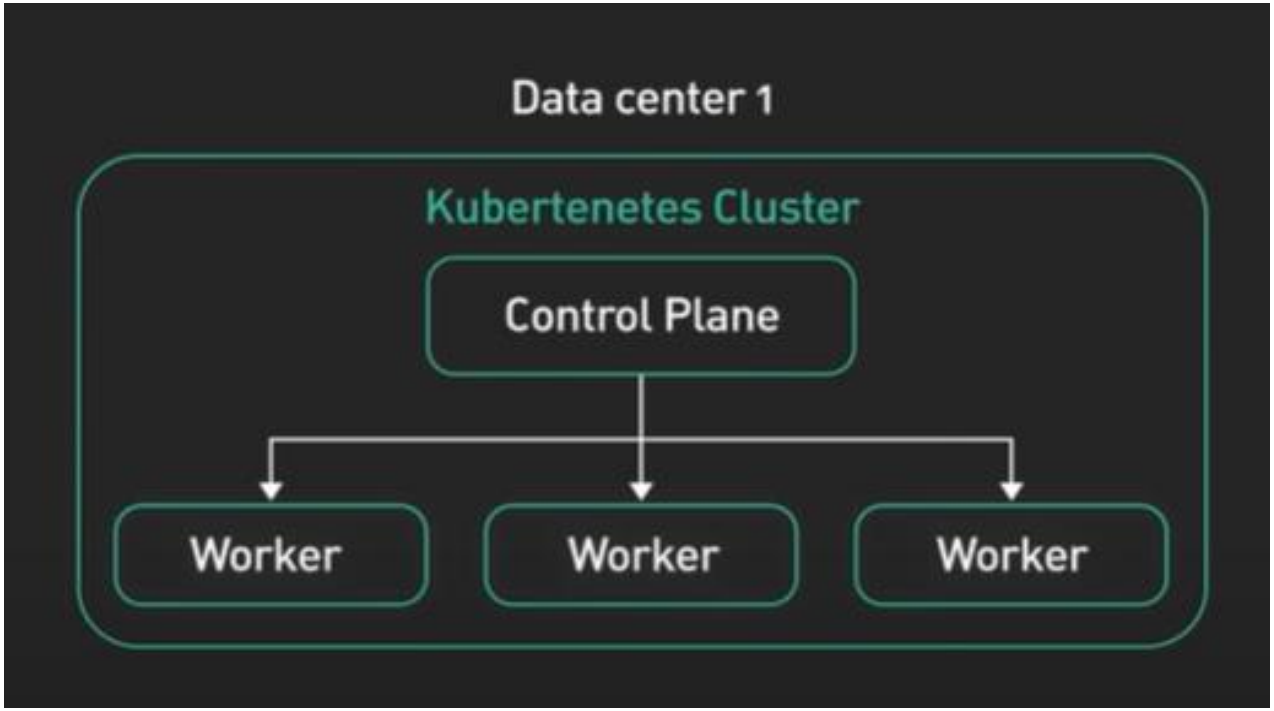


Image Courtesy: Bytebytego

Two Core Parts

1. **Control Plane (Master)**

   Responsible for managing the state of cluster

2. Worker Node(s)

Run the containerized applications workloads

K8s Architecture – Control Plane

- **API Server:**
  - Primary interface between the control plane and rest of the cluster.
- **Etcd:**
  - Stores the cluster's persistent state
  - What resources are available
  - Health information of cluster
  - Other components use /update this information about the cluster

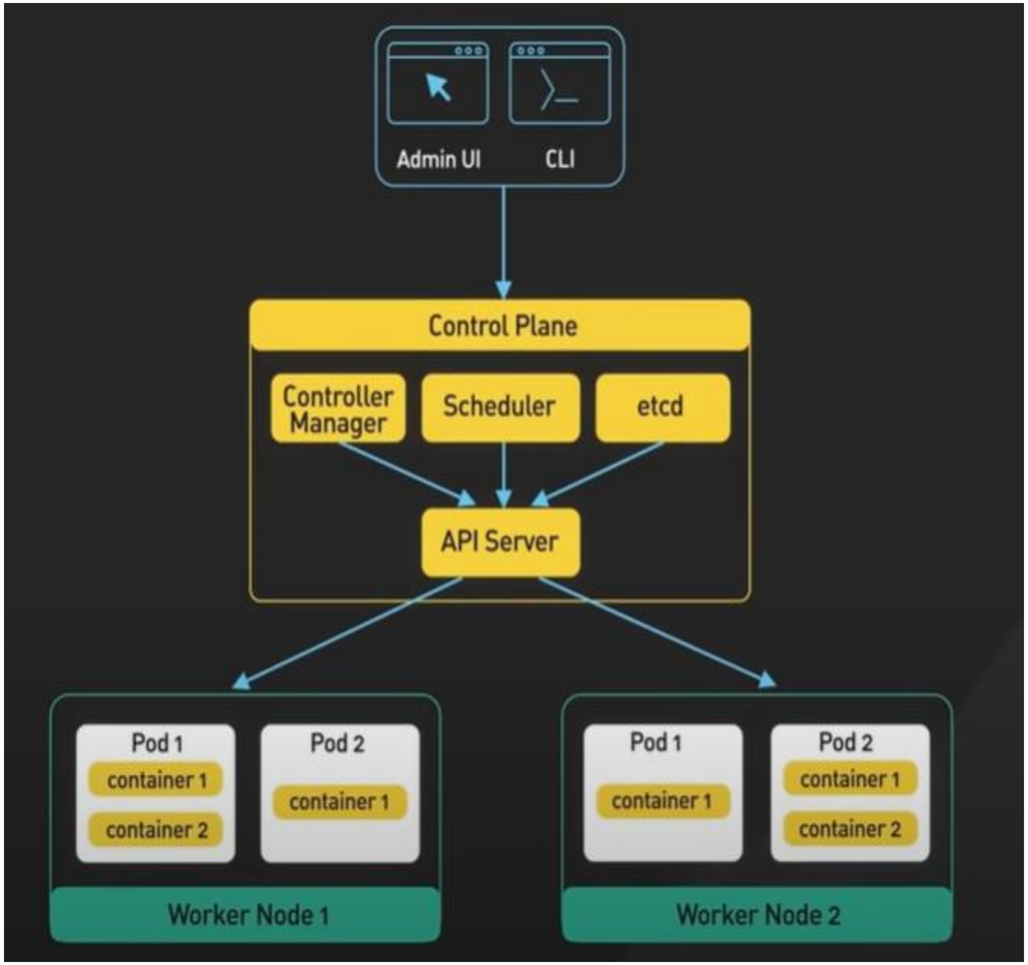


Image Courtesy: Bytebytego

K8s Architecture – Control Plane

- **Scheduler:**
  - Responsible for scheduling pods on the worker nodes
  - Uses available and required resources information for the pods to perform its job.

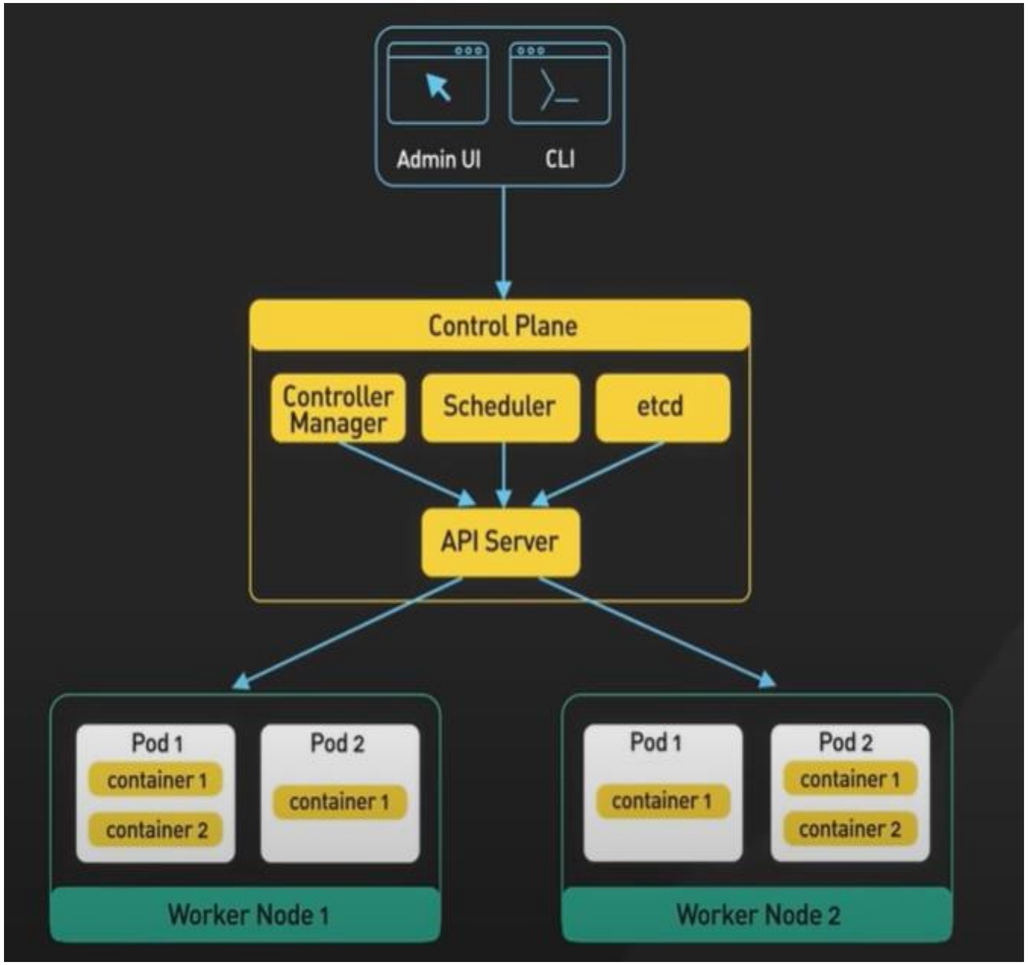
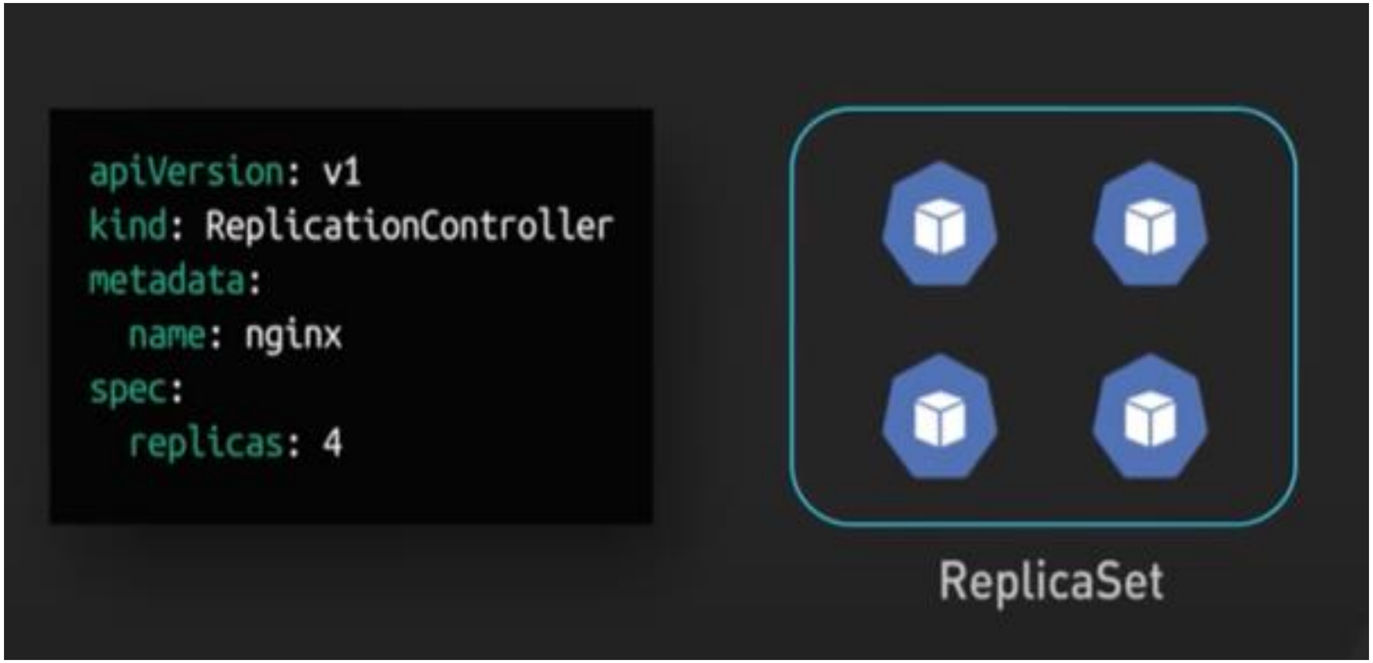


Image Courtesy: Bytebytego

K8s Architecture – Control Plane



```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 4
```

Image Courtesy: Bytebytego

- **Controller Manager**
  - Responsible for running those containers that manage the state of the cluster.
  - E.g: ReplicationController

K8s Architecture – Worker Node

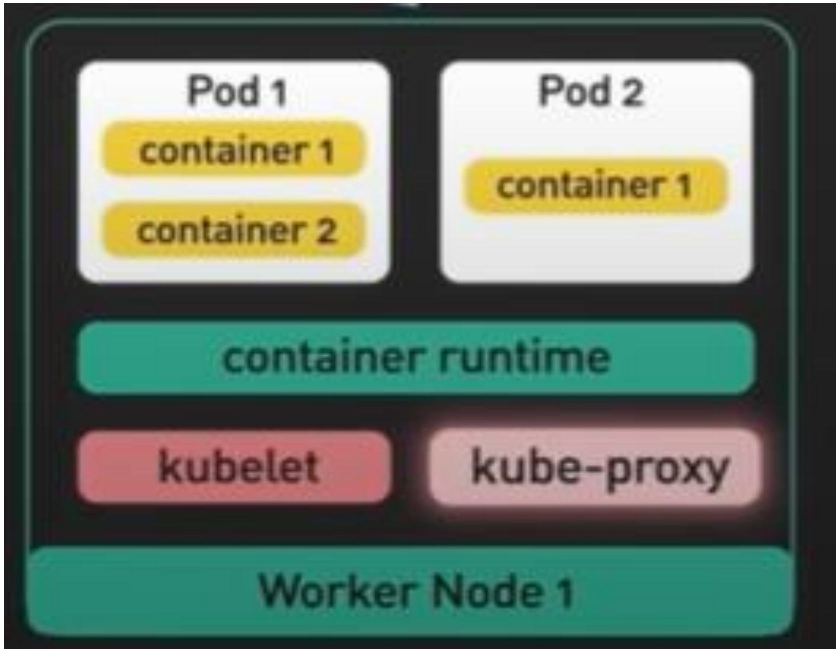


Image Courtesy: Bytebytego

Kubelet

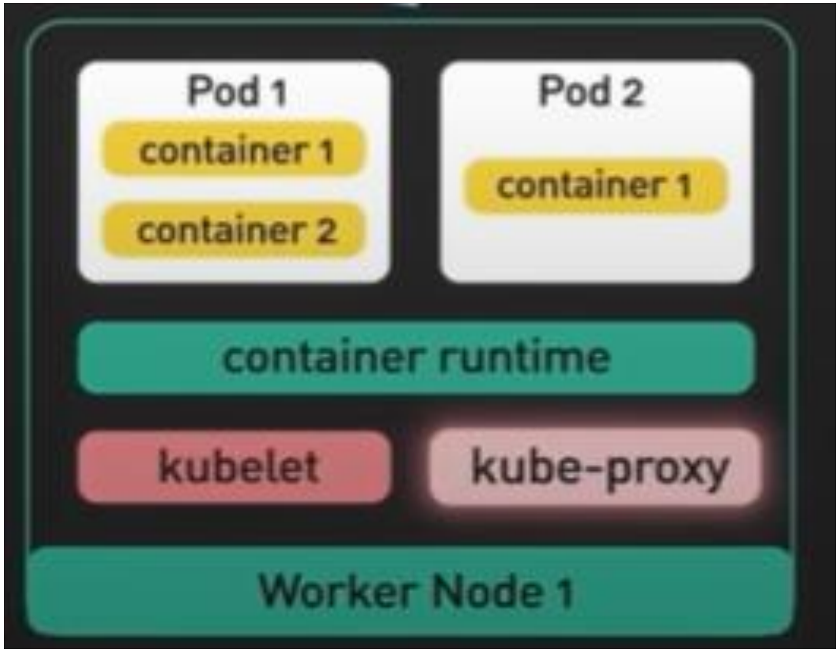
– Daemon running on each worker node

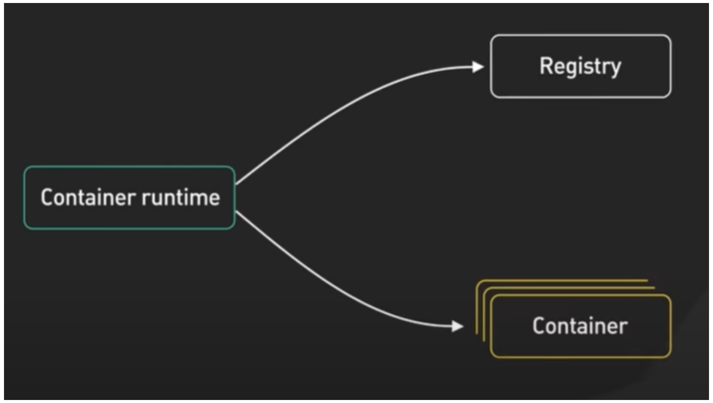
– Communicates with control plane.

– Receives information about which pods to run on the node and desired state of the pod is maintained.

KubeProxy: Routes the traffic to the correct pod, load balancing for even distribution of traffic across the pods

K8s Architecture – Worker Node





All Image Courtesy: Bytebytego.com

Container Runtime

Runs the container on the worker nodes