Container Orchestration Using Kubernetes

By

Venkat

Objectives of Prod Infrastructure?

1) High Availability

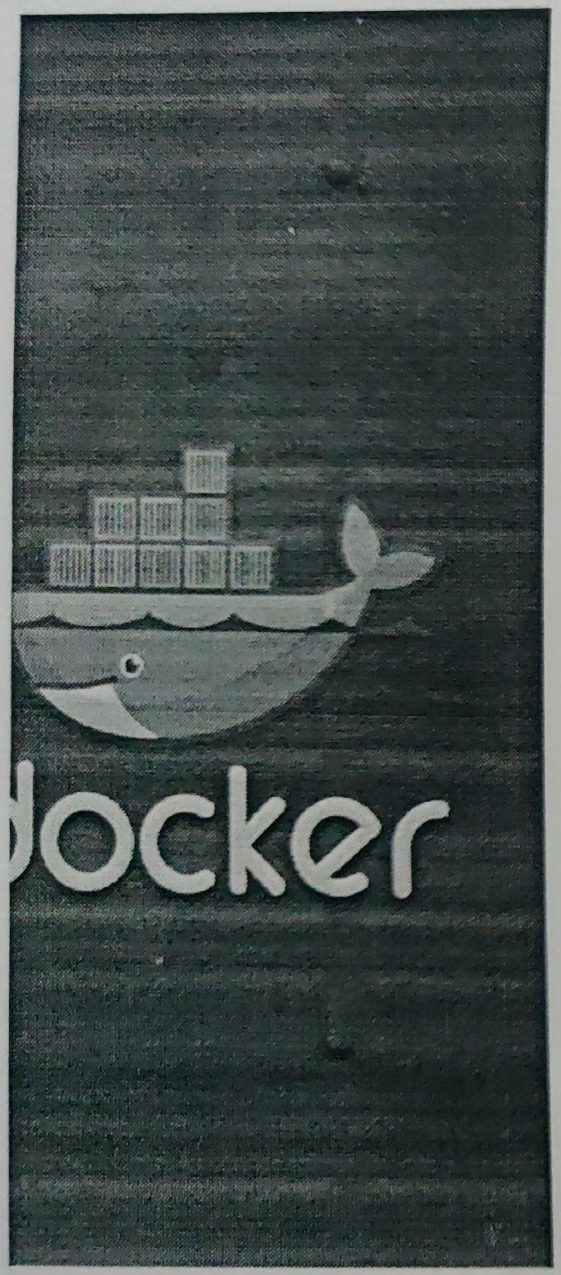
2) Fail Over

3) Load Balancing

4) Security

5) Scalability

6) Application Deployments (Updates)



What has Docker Done for Us?

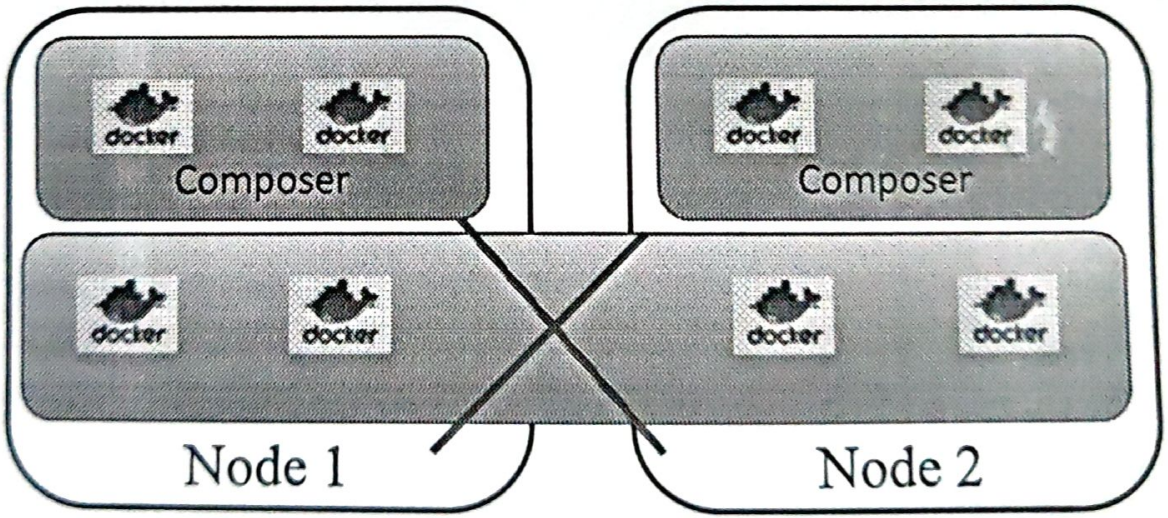
- Docker has solved the problem of packaging, deploying and running containerized applications

- Docker is great for managing a few containers running on a Single machine

The need for a container Orchestration Engine (COE):

→Docker limitations

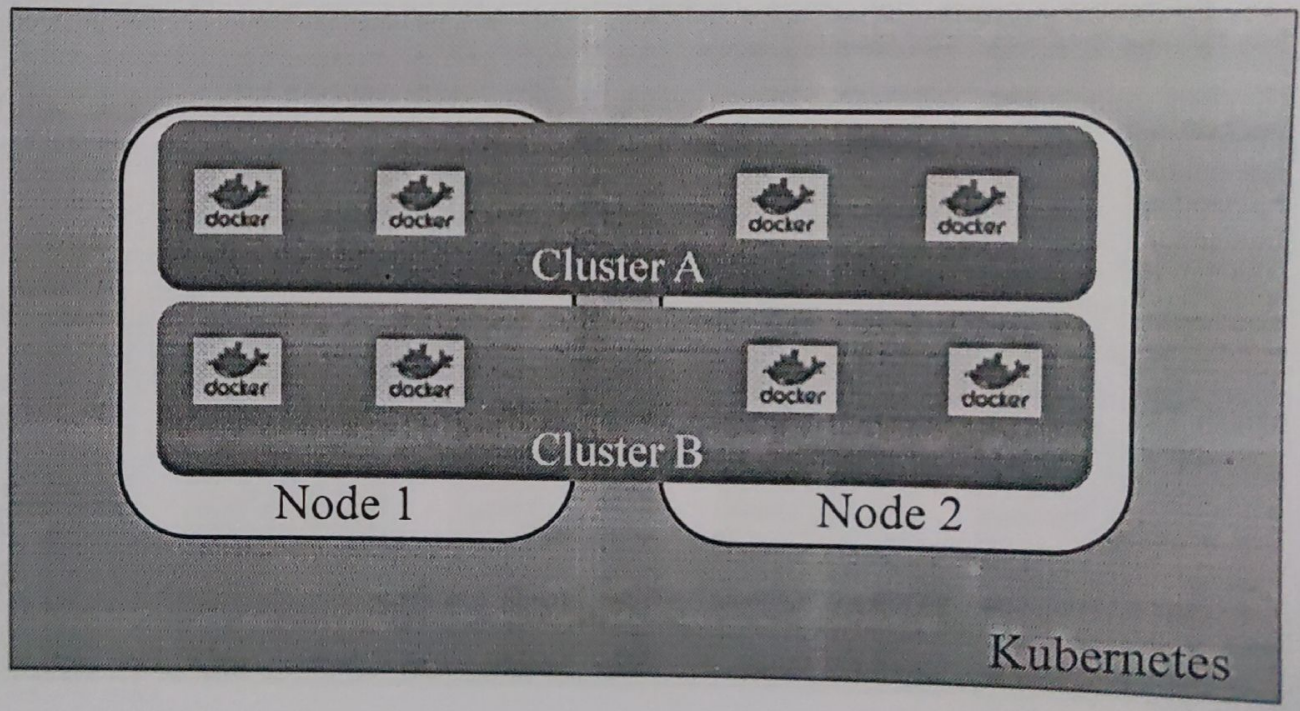
→Docker composers limitations



The need for a container Orchestration Engine (COE):

Features:

1) High Availability
2) Clustering
3) Fail Over
4) Load Balancing
5) Security
6) Scalability
7) Rolling Updates
8) Service Registry

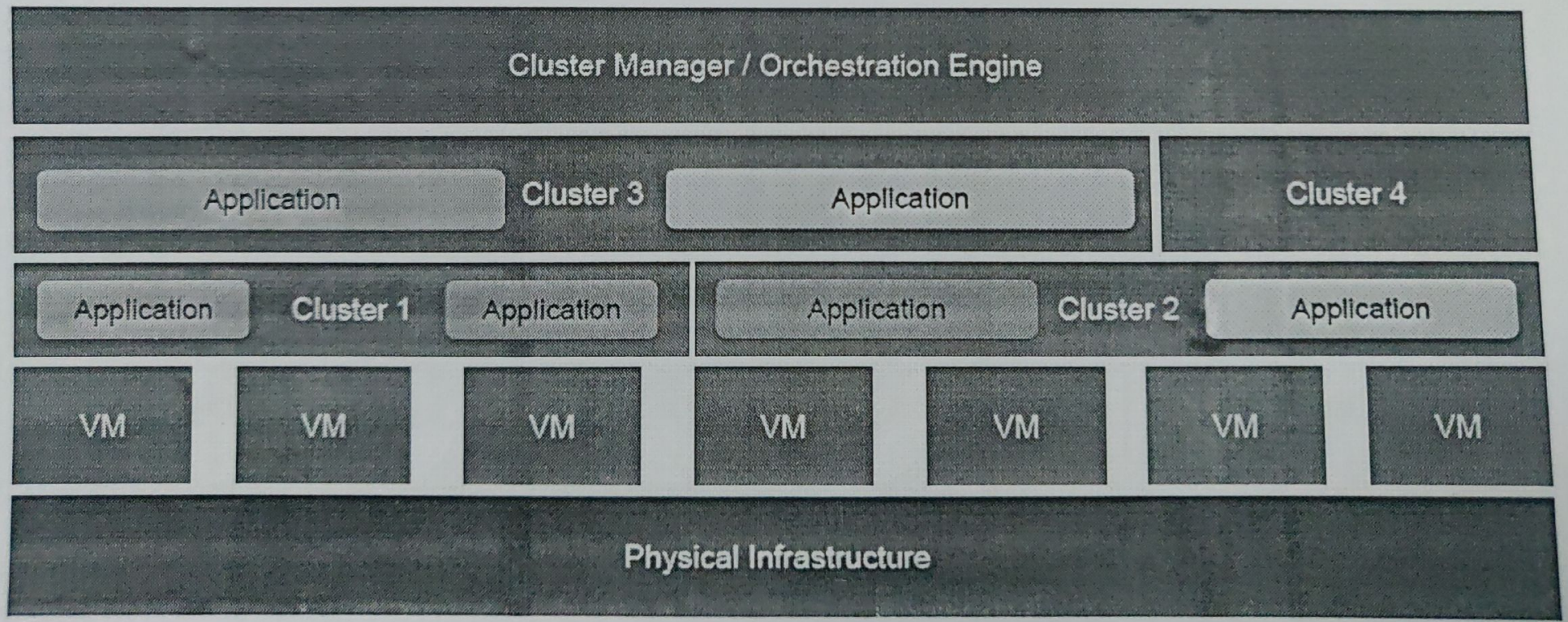


The Changing Face of the Datacenter

- The unit of deployment is changing from a machine to a container
  - Infrastructure has become immutable
  - Emphasis on treating the datacenter as a large server (cluster)
- Tools are evolving to manage the new datacenter infrastructure
  - Docker Swarm
  - Kubernetes
  - Mesosphere
- Manage the lifecycle of containerized applications running in production
- Automate the distribution of applications
- Ensure higher levels of utilization and efficiency

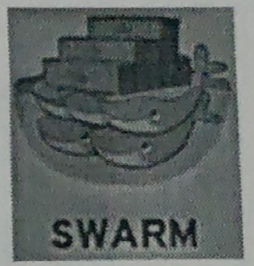
The Changing Face of the Datacenter

- A DATACENTER is not a collection of Computers, A DataCenter is a computer.



Battles of COEs; which one to choose

1) Swarm from Docker –



2) K8s from Google - very poweful and more features compare with Swarm

3) Mesos from Apache - This is not only for Dokcer, it has having the frameworks to run other apps also.

Why Kubernetes why not others:

→It is supporting by google from their infrastructure which is already implemented

→ K8s already using in the infrastructure.

--> Open - You can use k8s not for docker containers, it can be used for other containers also rocket

--> Great community

--> More features

What is Kubernetes?

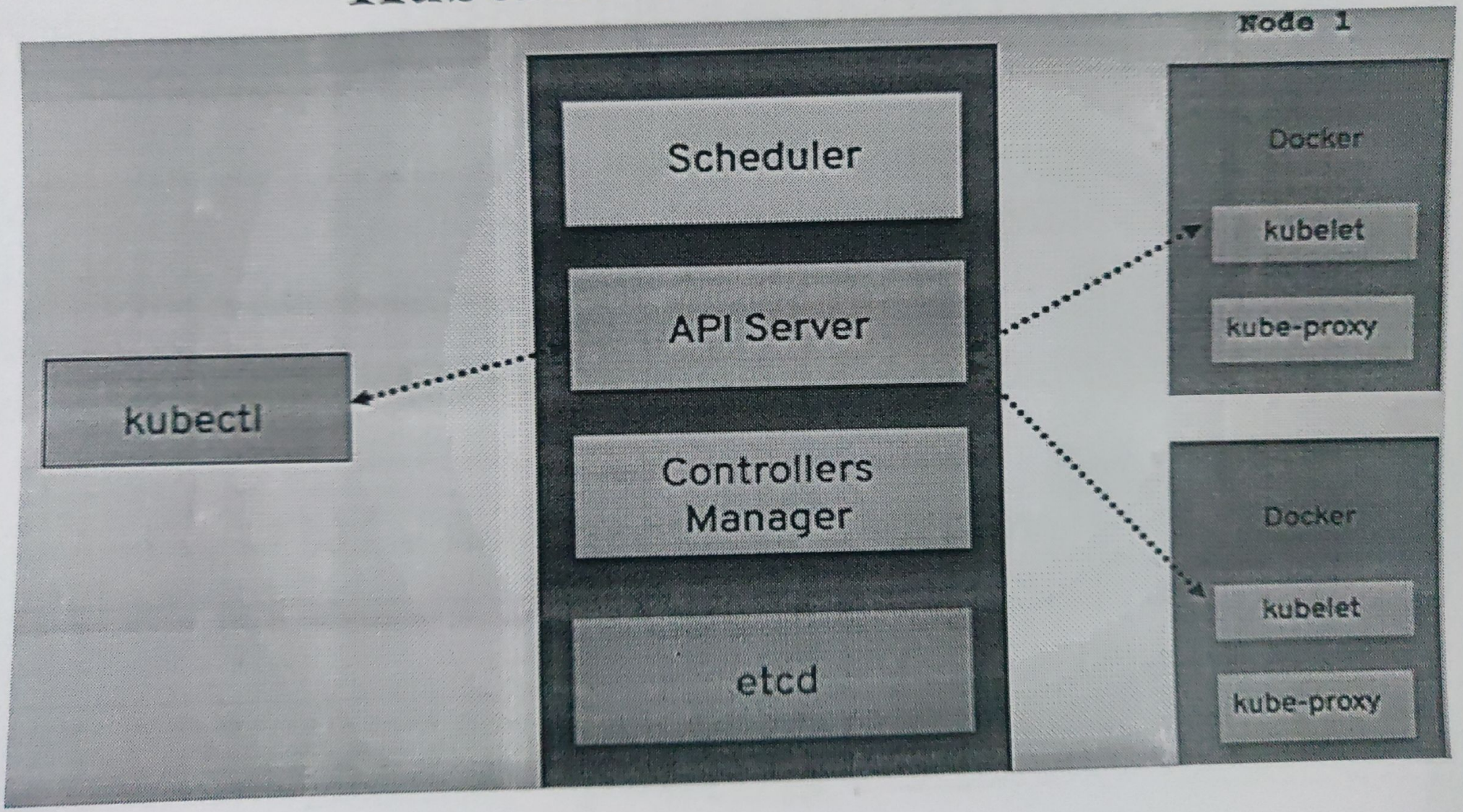
- Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services.
- It facilitates both declarative configuration and automation.
- Google open-sourced the Kubernetes project in 2014.

What Does Kubernetes do?

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

- Improves reliability
  - Continuously monitors and manages your containers
  - Will scale your application to handle changes in load
- Better use of infrastructure resources
  - Helps reduce infrastructure requirements by gracefully scaling up and down your entire platform
- Coordinates what containers run where and when across your system
- Easily coordinate deployments of your system
  - Which containers need to be deployed
  - Where should the containers be deployed

Kubernetes Architecture



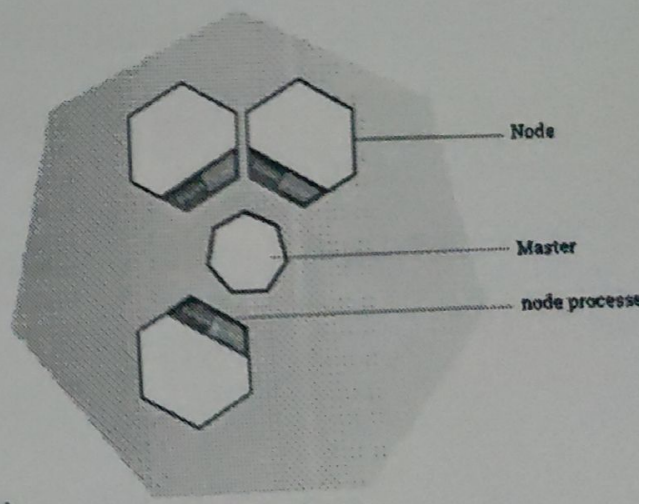
Kubernetes – Clusters

-- Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

-- Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.

-- A Kubernetes cluster consists of two types of resources·

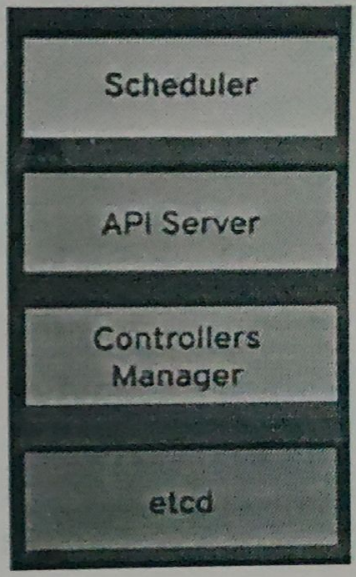
- The ***Master*** coordinates the cluster
- ***Nodes*** are the workers that run applications



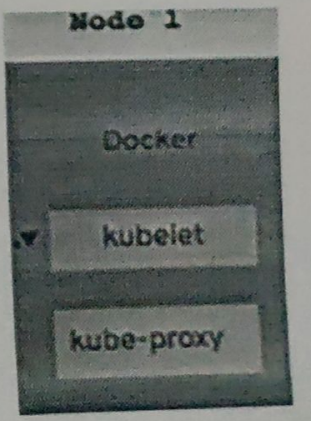
Kubernetes cluster

Kubernetes Components

- Master Components
  - Master components provide the cluster's control plane.
  - Master components make global decisions about the cluster (for example, scheduling), and detecting and responding to cluster events (starting up a new pod when a replication controller's 'replicas' field is unsatisfied).

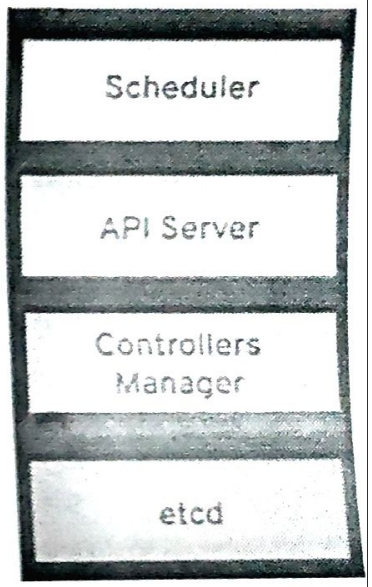


- Node Components
  - Node components run on every node
  - maintaining and running pods and providing the Kubernetes runtime environment.



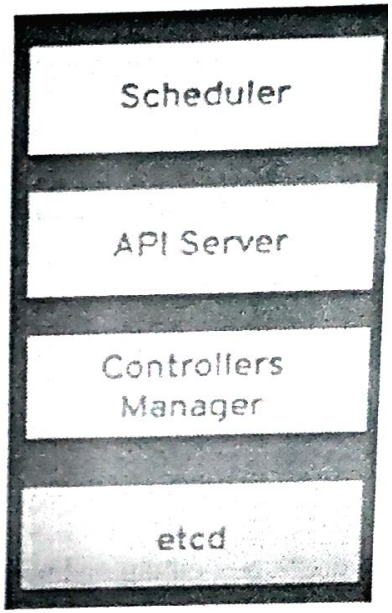
Master Components

- ***kube-apiserver***
  - Component on the master that exposes the Kubernetes API. It is the front-end for the Kubernetes control plane.
  - It is designed to scale horizontally – that is, it scales by deploying more instances.
- ***etcd***
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
  - Always have a backup plan for etcd's data for your Kubernetes cluster
- ***kube-scheduler***
  - Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.
  - Factors taken into account hardware/software specifications.

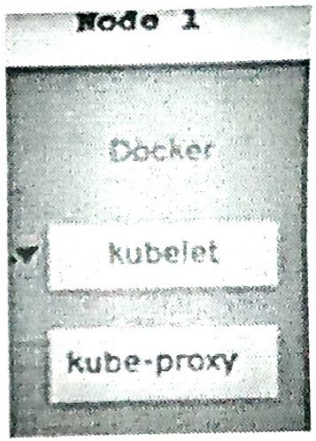


Master Components

- ***kube-controller-manager***
  - Component on the master that runs controllers.
  - Run as a process.
  - These controllers include:
    - ***Node Controller:*** Responsible for noticing and responding when nodes go down.
    - ***Replication Controller:*** Responsible for maintaining the correct number of pods for every replication controller object in the system.
    - ***Endpoints Controller:*** Populates the Endpoints object (that is, joins Services & Pods).
    - ***Service Account & Token Controllers:*** Create default accounts and API access tokens for new namespaces



Node Components



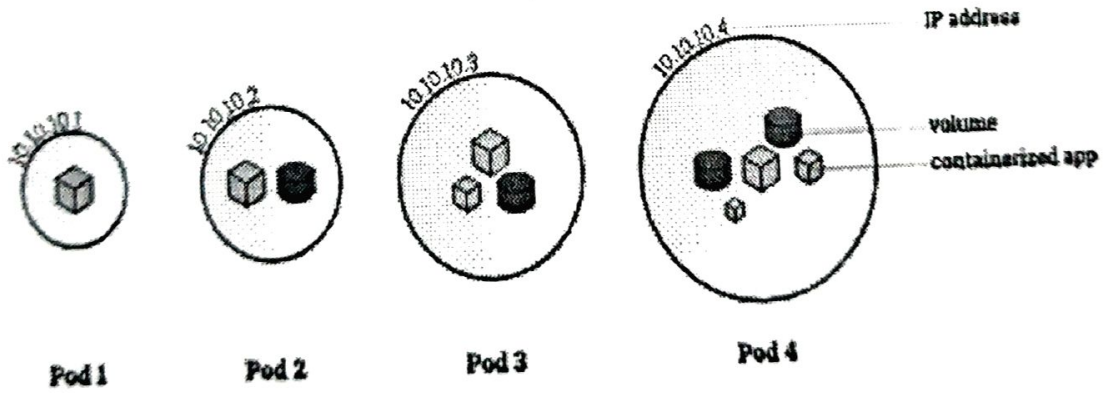
- ***kubelet***
  - An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
  - The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.
- ***kube-proxy***
  - kube-proxy enables the Kubernetes service abstraction by maintaining network rules on the host and performing connection forwarding.
- ***Container Runtime***
  - The container runtime is the software that is responsible for running containers. Kubernetes supports several runtimes: Docker, rkt, runc.

K8s – Application Resources

- PODS
  - Group of Containers
- Worker Nodes
- Replication Controllers or Sets
- Deployments
- Services
- Scaling Features
- Rolling Update

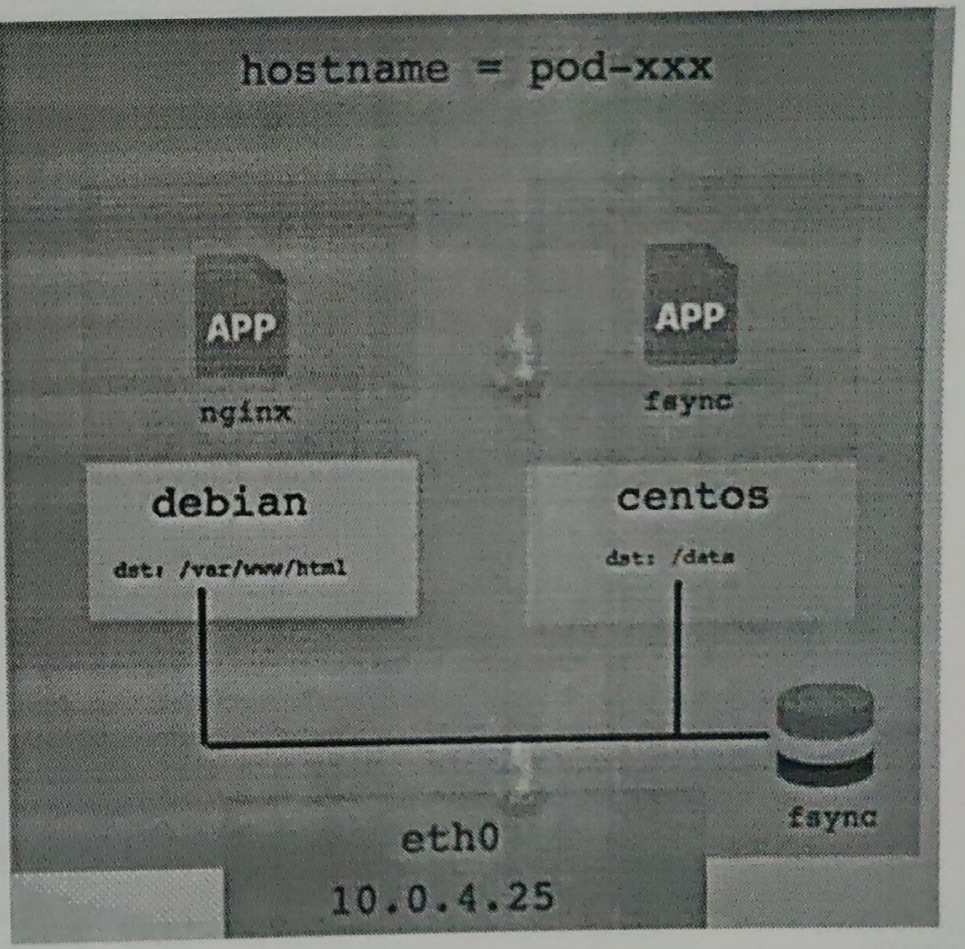
Kubernetes Pods

- The Pod is the core component of Kubernetes
- Collection of 1 or more containers
- Each pod should focus on one container, however multiple containers can be added to enhance features of the core container
- Pod can't be shared on multiple hosts. Pod is belong to the single node.



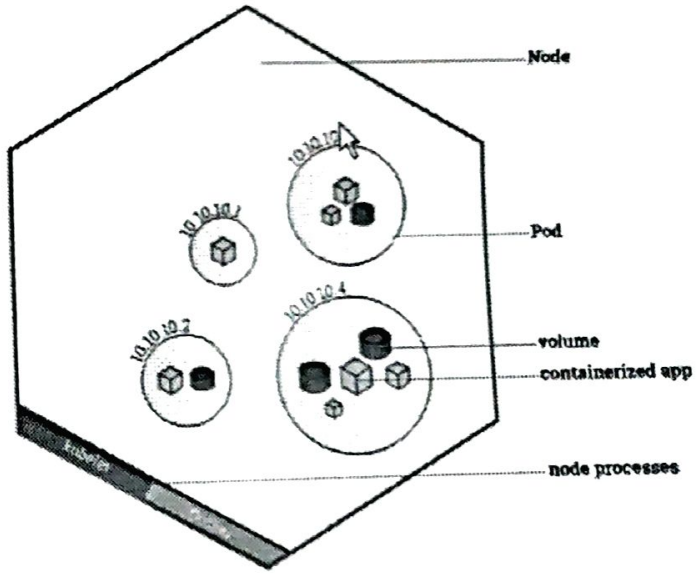
Kubernetes Pods

- Isolated:
  - Process or pid namespace
  - Root file system namespace
  - User namespace
- Common:
  - Network namespace
  - Hostname namespace
  - Volumes



Kubernetes Nodes

- A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the Master.



- A Node can have multiple pods, and the Kubernetes master automatically handles scheduling the pods across the Nodes in the cluster.

- The Master's automatic scheduling takes into account the available resources on each Node.

Replication Controller and Replication Sets

- A *Replication Controller* ensures that a specified number of pod replicas
running at any one time.

- How it works:
  - If there are too many pods, the Replication Controller terminates the extra pods.
  - If there are too few, the Replication Controller starts more pods.
  - Unlike manually created pods, the pods maintained by a Replication Controller are automatically replaced if they fail, are deleted, or are terminated.

Alternatives to Replication Controller - ReplicaSet

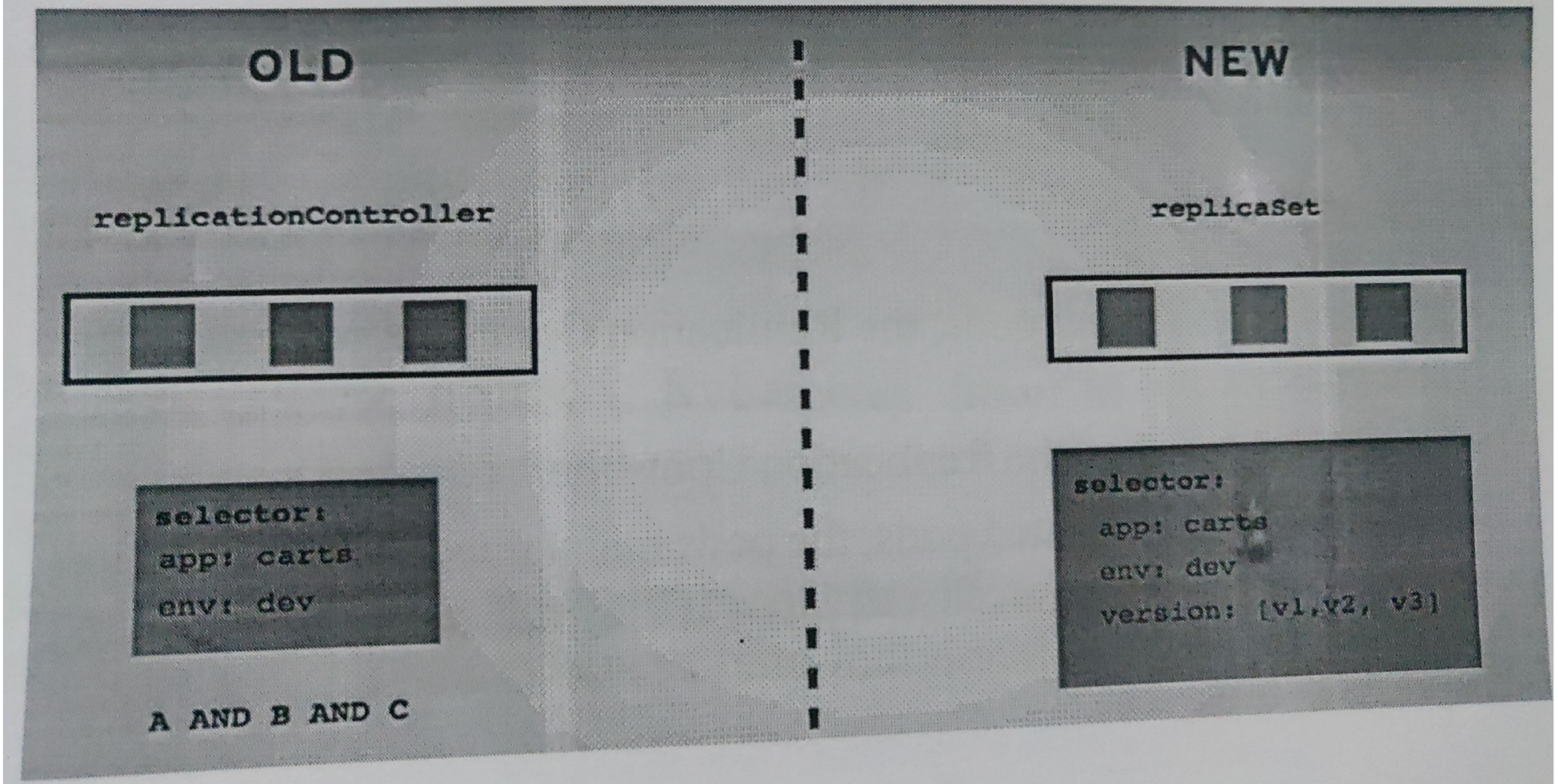
- ReplicaSet is the next-generation Replication Controller that supports the new set-based label selector.

- It's mainly used by Deployment as a mechanism to orchestrate pod creation, deletion and updates.

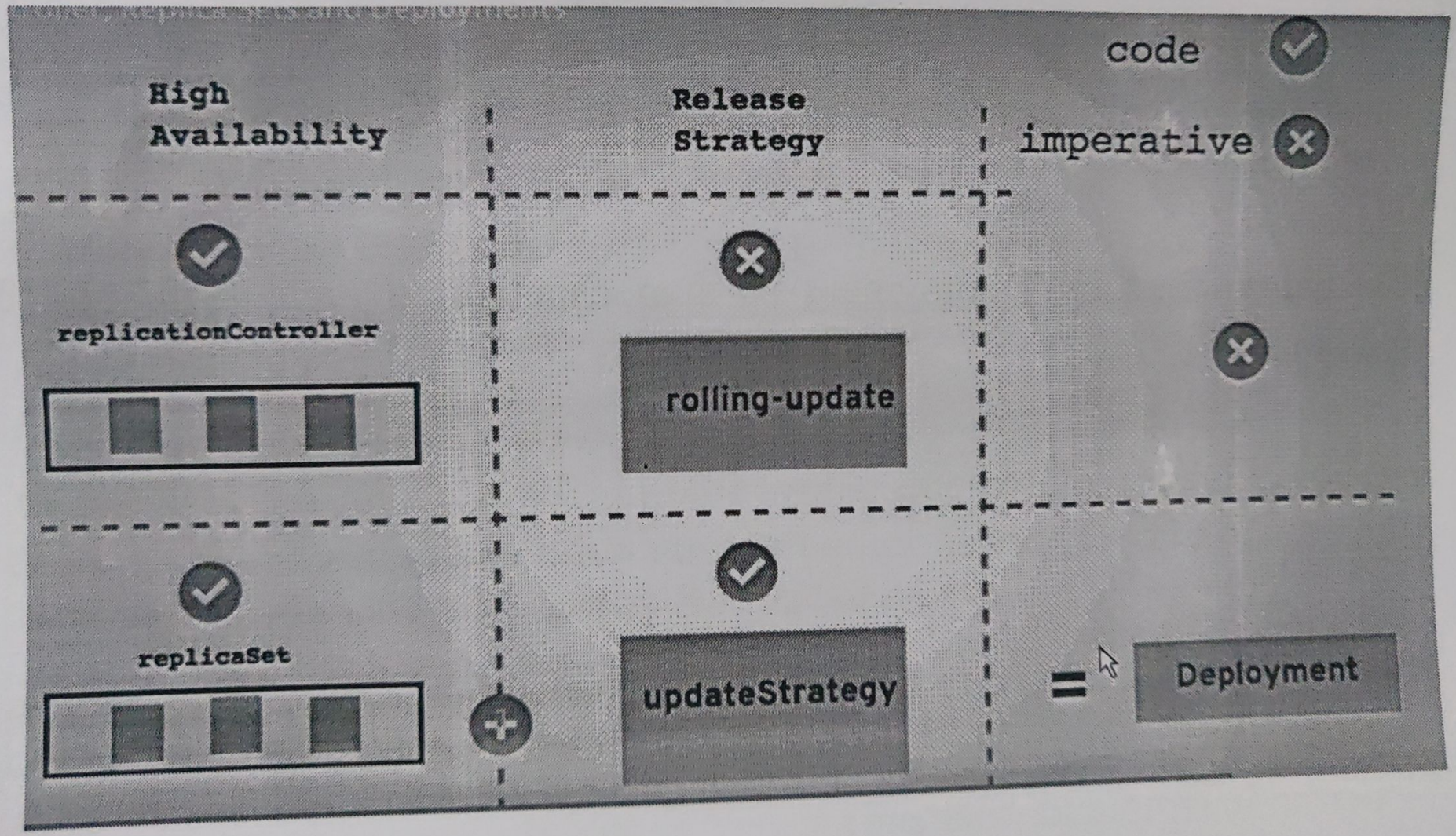
- *Set-based* label requirements allow filtering keys according to a set of values. Three kinds of operators are supported: in, notin and exists.

- For example:
  - environment in (production, qa)
  - tier notin (frontend, backend)

Diff Replication Controller and ReplicaSet



Diff Replication Controller and ReplicaSet



Kubernetes Deployments

- *A Deployment is responsible for creating and updating instances of your application.*

- Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment** configuration.

- The Deployment instructs Kubernetes how to create and update instances of your application.

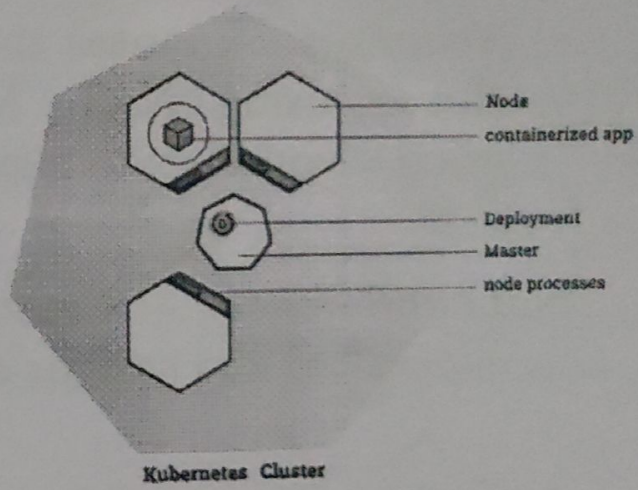
- Once you've created a Deployment, the Kubernetes master schedules mentioned application instances onto individual Nodes in the cluster.

Kubernetes Deployments

- Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances.

- If the Node hosting an instance goes down or is deleted, the Deployment controller replaces it.

- This provides a self-healing mechanism to address machine failure or maintenance.



Using a Service to Expose Your App

- A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them.

- Services enable a loose coupling between dependent Pods.

- The set of Pods targeted by a Service is usually determined by a *LabelSelector*
- Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service.

- Services allow your applications to receive traffic.
- Services can be exposed in different ways by specifying a type in the ServiceSpec.

Services Types

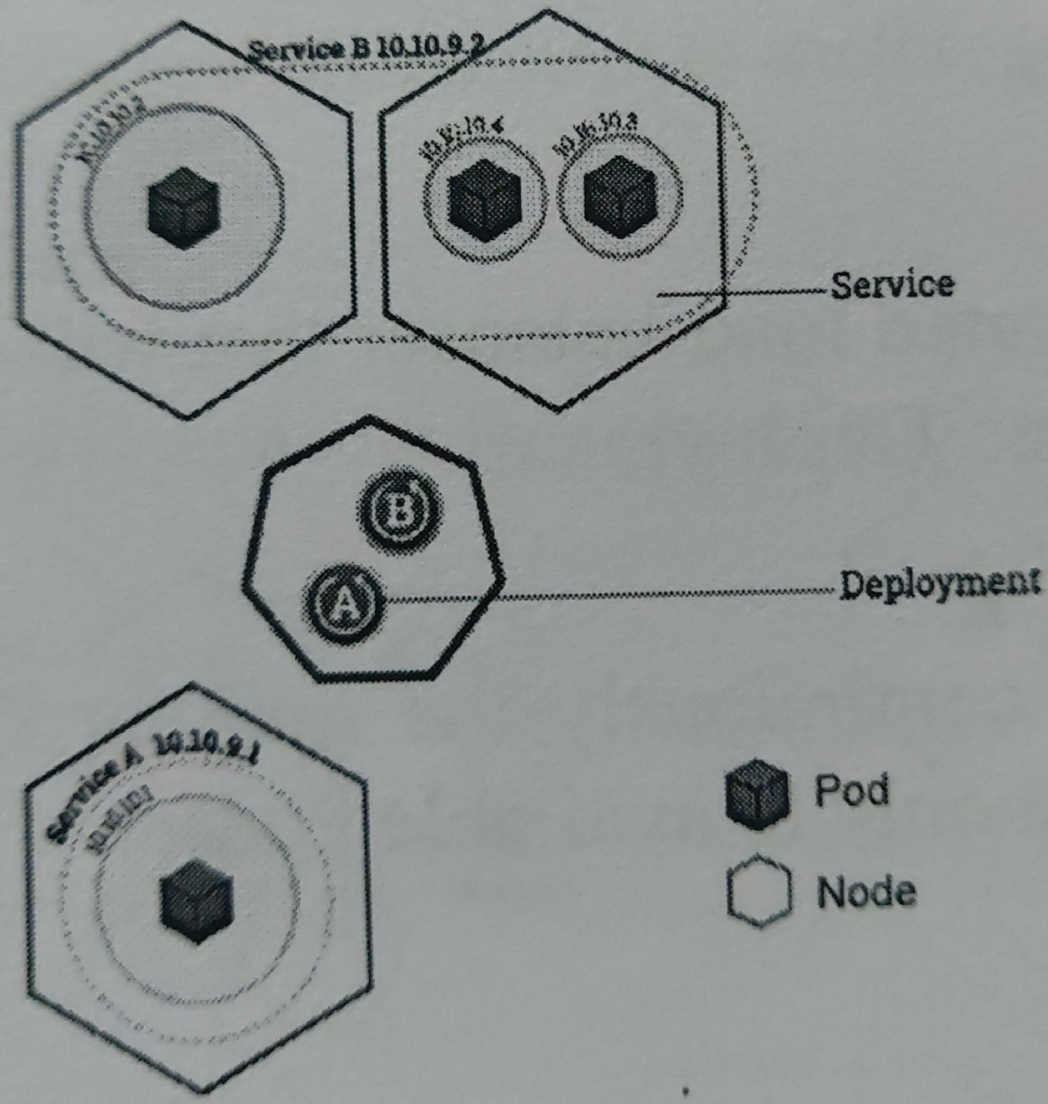
- ***ClusterIP (default)*** - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.

- ***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*** - Exposes the Service using an arbitrary name (specified by externalName in the spec) by returning a CNAME record with the name. No proxy is used. This type requires v1.7 or higher of kube-dns.

Kubernetes Service

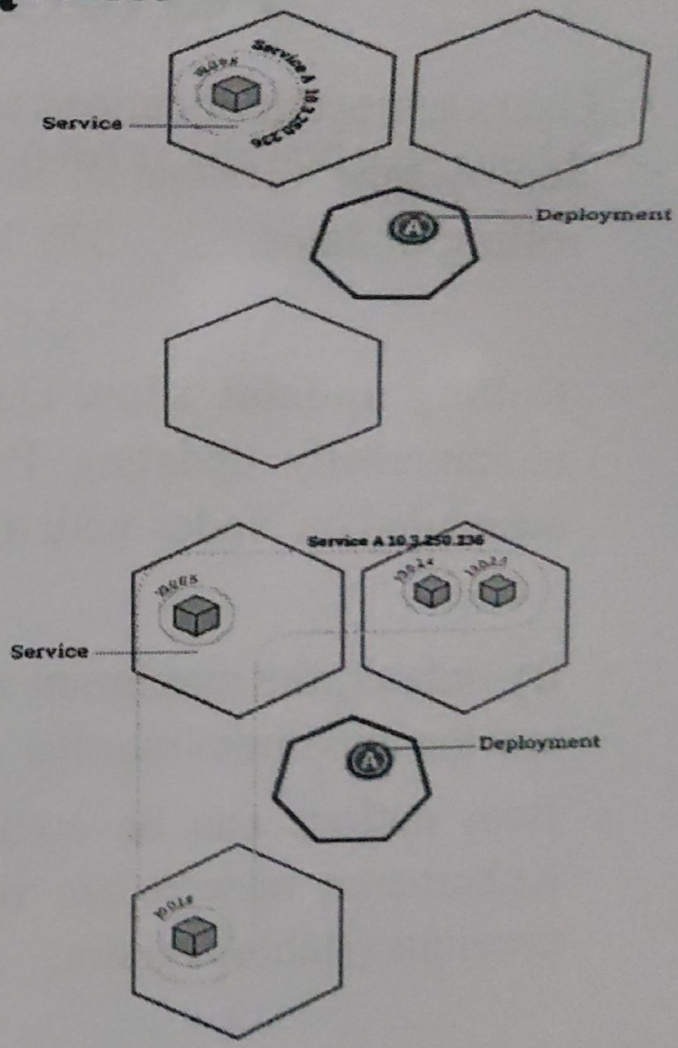


Scaling an application

- **Scaling** is accomplished by changing the number of replicas in a Deployment.

- Scaling out a Deployment will ensure new Pods are created and scheduled to Nodes with available resources. Scaling will increase the number of Pods to the new desired state.

- Scaling to zero is also possible, and it will terminate all Pods of the specified Deployment.



Scaling an application

- Running multiple instances of an application will require a way to distribute the traffic to all of them.
- Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment.
- Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.

Performing a Rolling Update

- Users expect applications to be available all the time and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates.

- **Rolling updates** allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.

- By default, the maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one.
- Both options can be configured to either numbers or percentages (of Pods). In Kubernetes, updates are versioned and any Deployment update can be reverted to previous (stable) version.

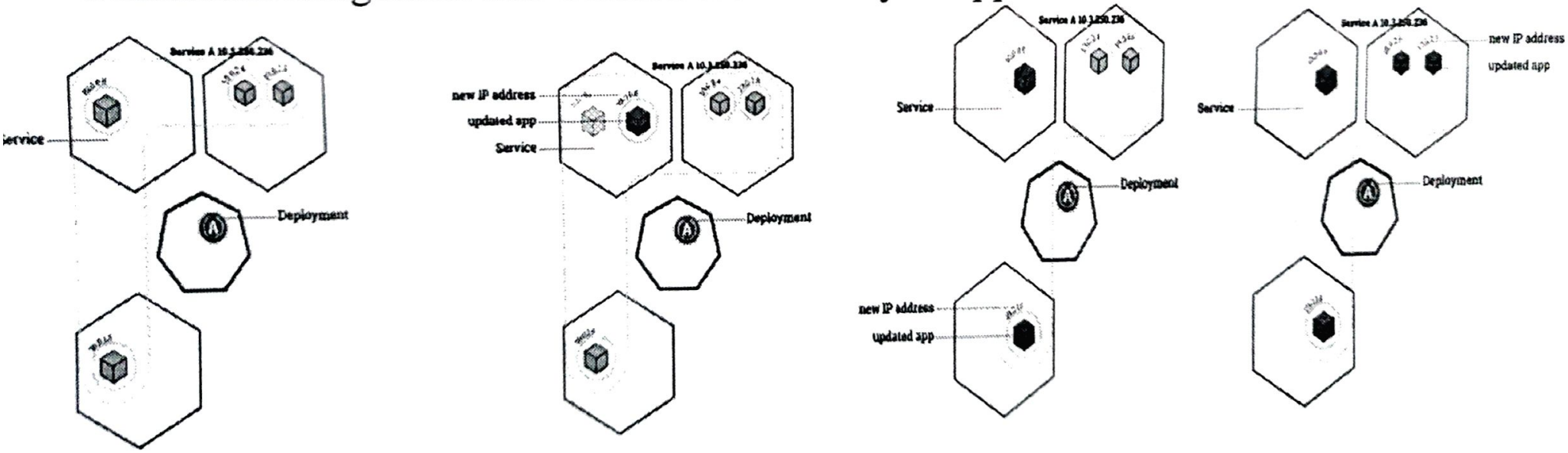
Rolling updates overview

Rolling updates allow the following actions:

-- Promote an application from one environment to another (via container image updates)

-- Rollback to previous versions

-- Continuous Integration and Continuous Delivery of applications with zero downtime



Resource Config

- Each entity created with kubernetes is a resource including pod, service, deployments, replication controller etc
- Resources can be defined as YAML or JSON.

- **AKMS** => Resource Config Specs

  apiVersion: v1

  kind:

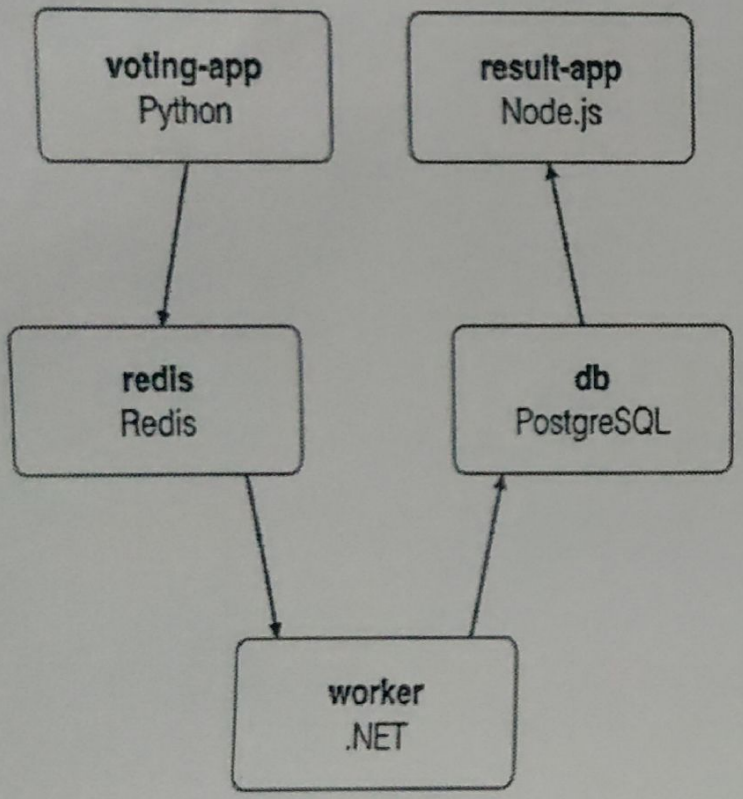
  metadata:

  spec:

Sample Pod Config

```
apiVersion: v1
kind: Pod
metadata:
    name: vote
    labels:
        app: python
        role: vote
        version: v1
spec:
    containers:
        - name: app
          image: gvenkat/vote:v1
          ports:
              - containerPort: 80
                protocol: TCP
```

Architecture



- A Python webapp which lets you vote between two options
- A Redis queue which collects new votes
- A .NET worker which consumes votes and stores them in…
- A Postgres database backed by a Docker volume
- A Node.js webapp which shows the results of the voting in real time

K8s - Resources