kubernetes

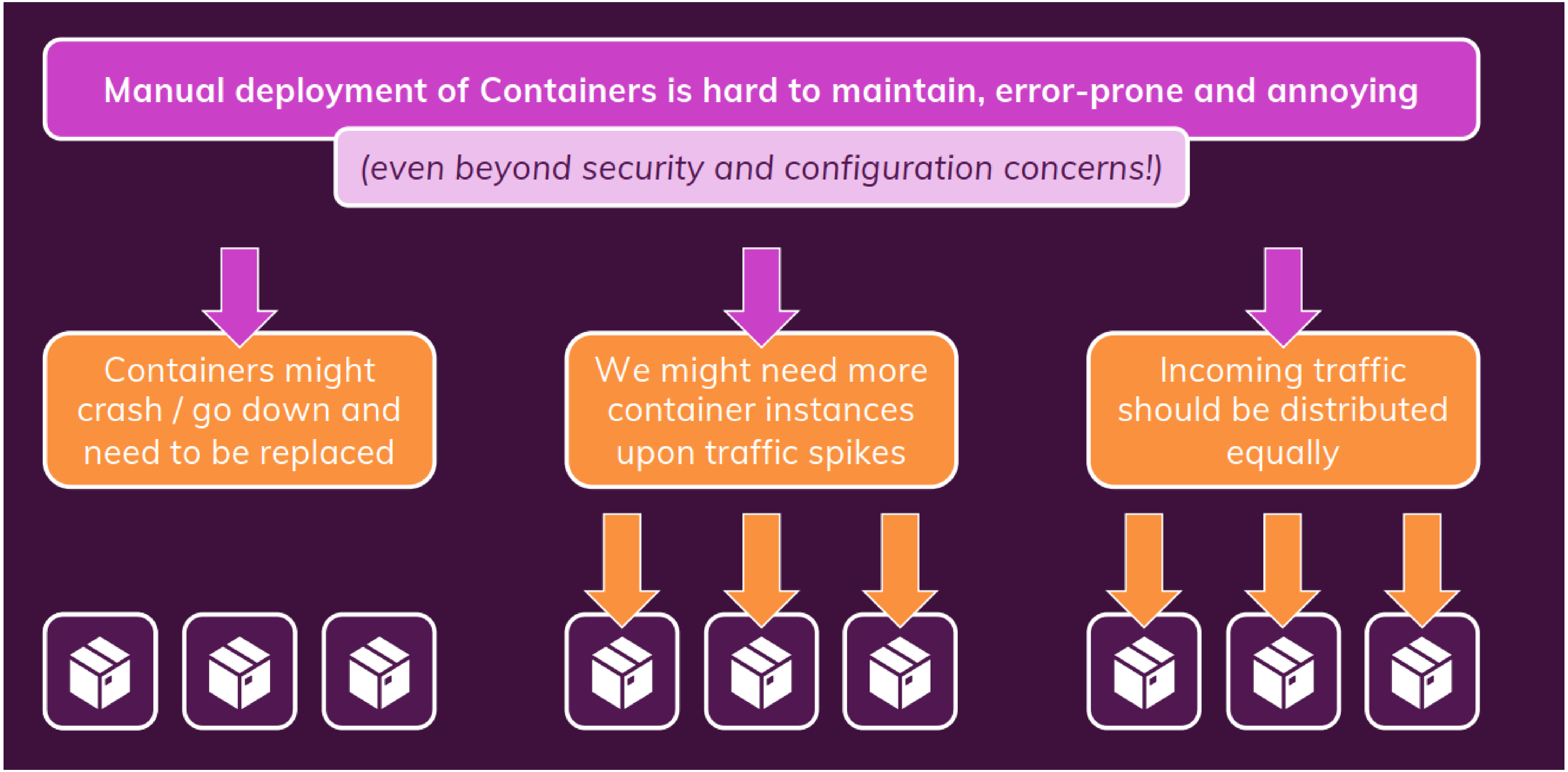
Kubernetes Overview

Kubernetes in Depth

Kubernetes overview

- portable, extensible, open source platform for managing containerized applications (container orchestration)
  - supports declarative configuration and automation
- Open sourced by Google in 2014
  - combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community
  - has a large, rapidly growing ecosystem

Why Kubernetes ?



Manual deployment of Containers is hard to maintain, error-prone and annoying

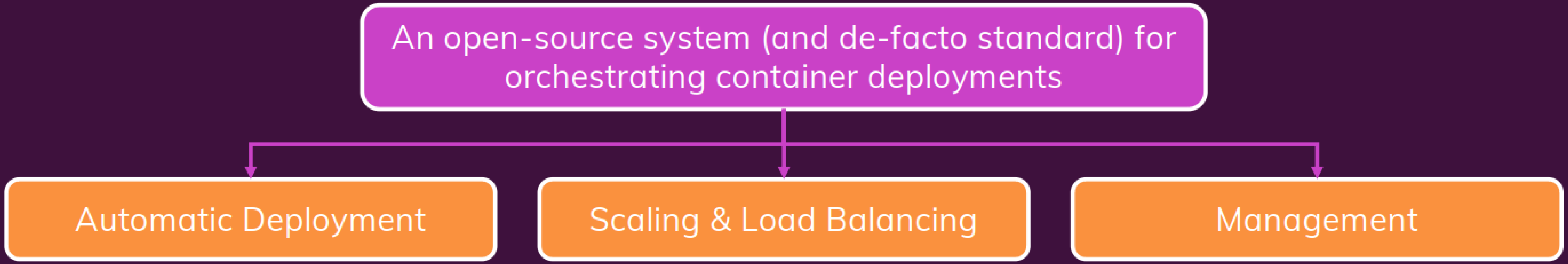
(even beyond security and configuration concerns!)

Containers might crash / go down and need to be replaced

We might need more container instances upon traffic spikes

Incoming traffic should be distributed equally

Kubernetes



An open-source system (and de-facto standard) for orchestrating container deployments

- Automatic Deployment
- Scaling & Load Balancing
- Management

Kubernetes key features

- ❖ Service discovery and load balancing
  - can expose a container using the DNS name or using their own IP address
  - load balance and distribute incoming network traffic to cluster of containers

- ❖ Storage orchestration
  - supports mounting a storage system of your choice, such as local storages, public cloud providers, and more.

Kubernetes key features

- ❖ Automated rollouts and rollbacks
  - automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

- ❖ Self-healing
  - Kubernetes restarts containers that fail
  - Replaces containers and kills containers that don't respond to health checks

Kubernetes key features

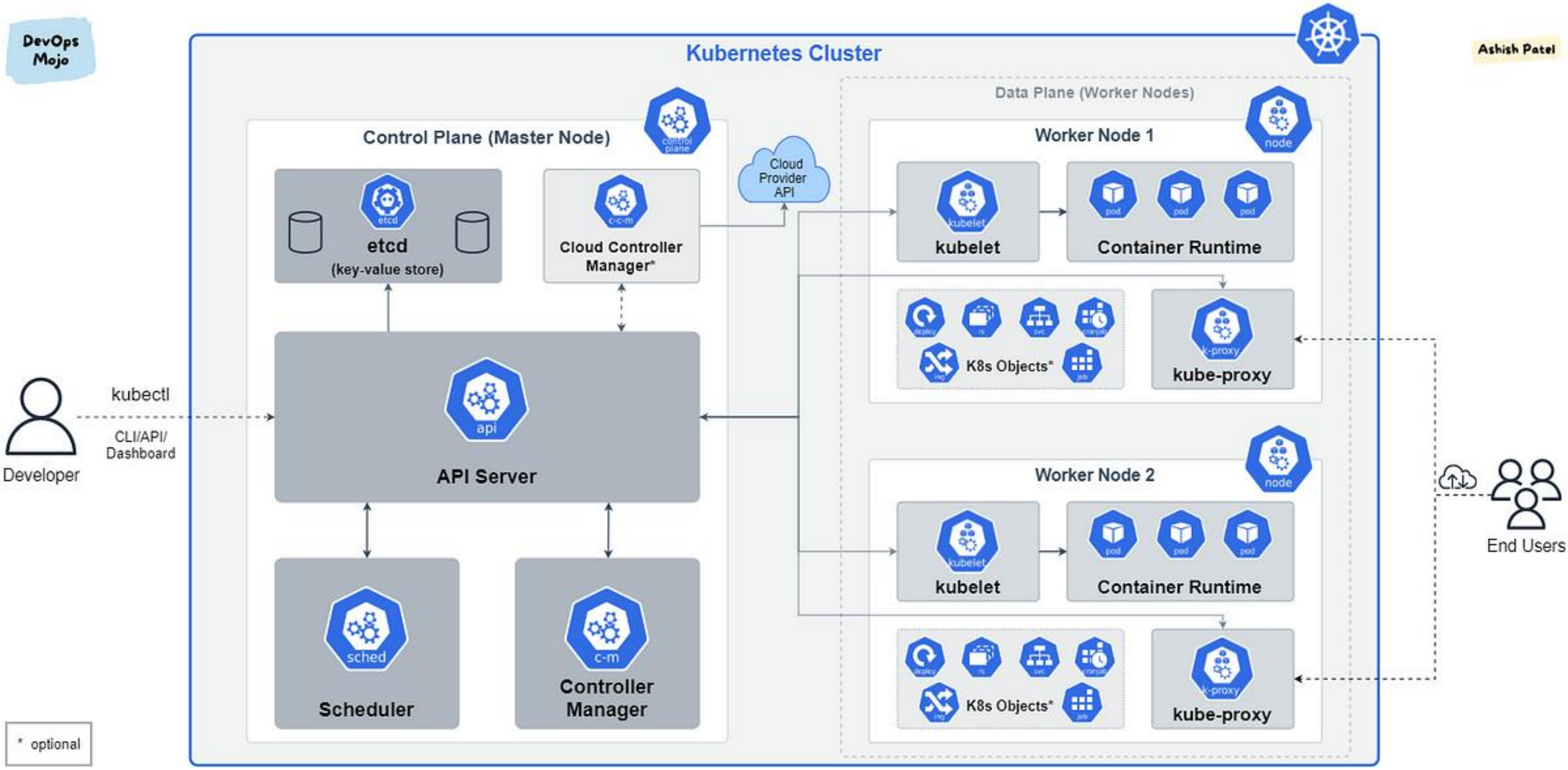
- Secret and configuration management
  - lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys
  - deploy and update secrets and application configuration without rebuilding your container images

Kubernetes Architecture

Kubernetes Overview

Kubernetes architecture

- ❖A Kubernetes cluster consists of a set of worker nodes coordinated by a master node (control plane)
  - Each cluster has at least one worker node
  - The worker node(s) host the Pods that are the components of the application workload
- ❖The control plane manages the worker nodes and the Pods in the cluster
  - In production environments, the control plane usually runs across multiple servers to ensure fault-tolerance and high availability


DevOps Mojo
Kubernetes Cluster
Ashish Patel
Control Plane (Master Node)
control plane
etcd
etcd
(key-value store)
c-c-m
Cloud Controller Manager*
Cloud Provider API
api
API Server
sched
Scheduler
c-m
Controller Manager
Developer
kubectl
CLI/API/ Dashboard
Data Plane (Worker Nodes)
Worker Node 1
node
kubelet
kubelet
pod
pod
pod
Container Runtime
deploy
rs
svc
cronjob
ing
K8s Objects*
job
k-proxy
kube-proxy
Worker Node 2
node
kubelet
kubelet
pod
pod
pod
Container Runtime
deploy
rs
svc
cronjob
ing
K8s Objects*
job
k-proxy
kube-proxy
End Users
* optional

Control plane components

- make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events
- Consists of:
  - kube-apiserver
  - etcd
  - kube-scheduler
  - kube-controller-manager
  - cloud-controller-manager

Control plane components

- kube-apiserver
  - exposes the Kubernetes API and serves as the front end for the Kubernetes control plane
  - kube-apiserver is designed to scale horizontally by simply deploying more instances
- etcd
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

Control plane components

- kube-scheduler
  - watches for newly created Pods with no assigned node, and selects a node for them to run on
  - scheduling decisions based on individual and collective resource requirements

Control plane components

- kube-controller-manager
  - Control plane component that runs controller processes.
  - There are many different types of controllers
  - Node controller: Responsible for noticing and responding when nodes go down.
  - Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
  - ServiceAccount controller: Create default ServiceAccounts for new namespaces.

Control plane components

- cloud-controller-manager
  - Control plane component that embeds cloud-specific control logic
  - Link local cluster into cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

Control plane components

❖ cloud-controller-manager
- There are many different types of controllers
- Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route controller: For setting up routes in the underlying cloud infrastructure
- Service controller: For creating, updating and deleting cloud provider load balancers

Node components

- Node components run on every worker node
  - maintaining running pods and providing the Kubernetes runtime environment
- Key components
  - kubelet
  - kube-proxy
  - Container runtime

Node components

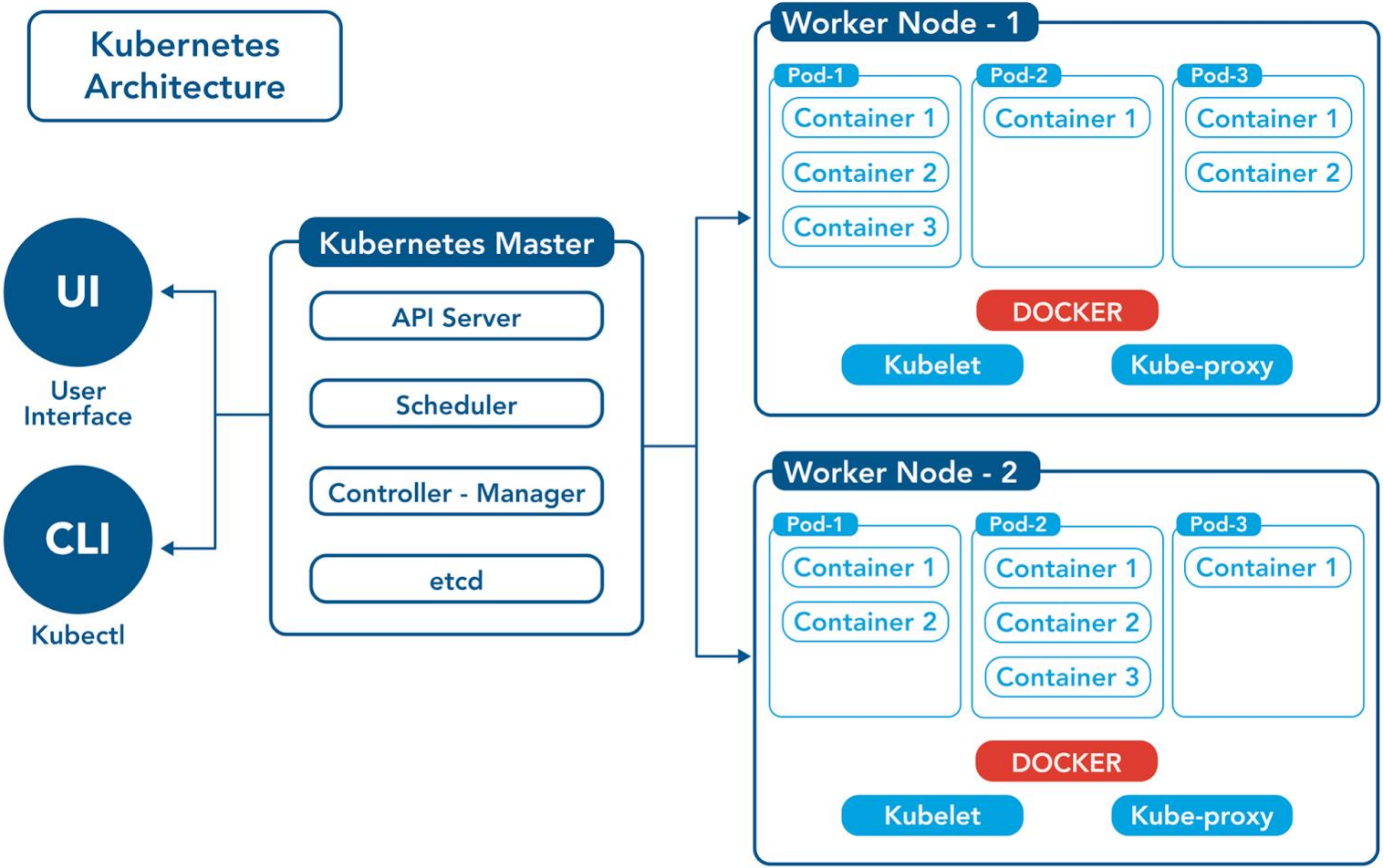
- kubelet
  - An agent that runs on each node in the cluster
  - It makes sure that containers are running in a Pod.
  - takes a set of PodSpecs and ensures that the containers described in those PodSpecs are running and healthy.
- kube-proxy
  - network proxy that runs on each node in your cluster, implementing the Kubernetes Service concept.
  - maintains network rules on nodes to allow network communication to your Pods from network sessions inside or outside of your cluster

Node components

❖Container runtime

- allows Kubernetes to run containers effectively
- responsible for managing the execution and lifecycle of containers within the Kubernetes environment
- supports container runtimes such as containerd (default runtime for Docker), CRI-O, and any other implementation of the Kubernetes CRI (Container Runtime Interface).

Kubernetes Architecture



UI

User Interface

CLI

Kubectl

Kubernetes Master

API Server

Scheduler

Controller - Manager

etcd

Worker Node - 1

Pod-1

Container 1

Container 2

Container 3

Pod-2

Container 1

Pod-3

Container 1

Container 2

DOCKER

Kubelet

Kube-proxy

Worker Node - 2

Pod-1

Container 1

Container 2

Pod-2

Container 1

Container 2

Container 3

Pod-3

Container 1

DOCKER

Kubelet

Kube-proxy

Addons

- These use Kubernetes resources to implement cluster features
- Key addons
  - DNS
  - WebUI (Dashboard)
  - Container Resource Monitoring
  - Cluster-level Logging
  - Network Plugins

Addons

- Cluster DNS
  - All Kubernetes clusters should have cluster DNS
  - This is a DNS server which serves DNS records for Kubernetes services.
  - Containers started by Kubernetes automatically include this DNS server in their DNS searches
- Web UI (Dashboard)
  - general purpose, web-based UI for Kubernetes clusters
  - allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

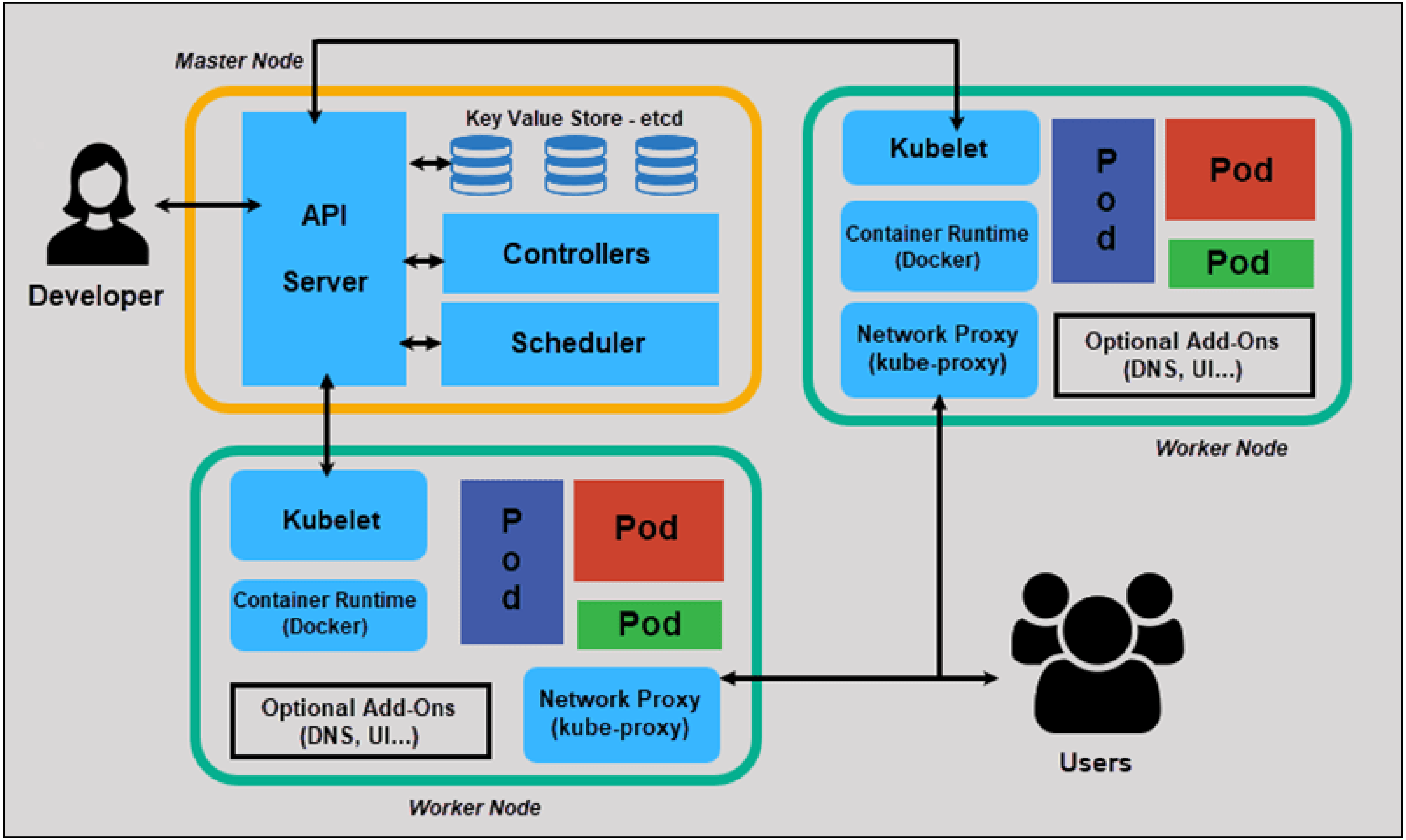
Addons

- Container Resource Monitoring
  - records generic time-series metrics about containers in a central database
  - provides a UI for browsing that data.
- Cluster-level Logging
  - saving container logs to a central log store with search/browsing interface

Addons

❖Network Plugins

- implement the container network interface (CNI) specification
- responsible for allocating IP addresses to pods and enabling them to communicate with each other within the cluster


Master Node
Key Value Store - etcd
Developer
API
Server
Controllers
Scheduler
Kubelet
Container Runtime
(Docker)
Pod
Pod
Pod
Network Proxy
(kube-proxy)
Optional Add-Ons
(DNS, UI...)
Worker Node
Kubelet
Container Runtime
(Docker)
Pod
Pod
Pod
Optional Add-Ons
(DNS, UI...)
Network Proxy
(kube-proxy)
Users
Worker Node

kubernetes

Objects and Workloads

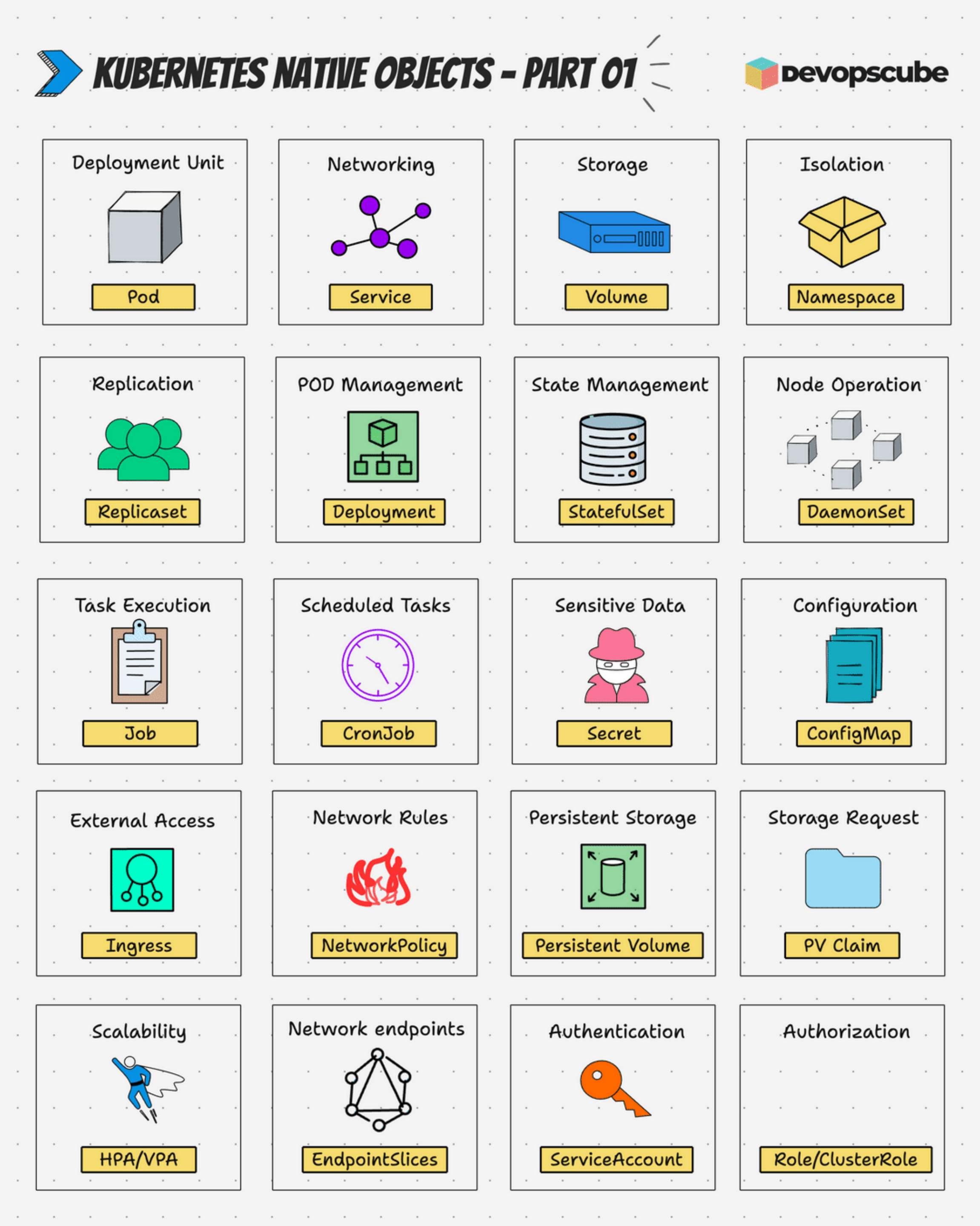
Kubernetes Overview

Kubernetes objects

- persistent entities used to describe the desired state of your cluster (record of intent), this includes:
  - which containerized applications are running and on which nodes
  - The resources available to those applications
  - The policies that govern application behavior, such as restart policies, upgrades, and fault-tolerance
- Once an object is created
  - Kubernetes will ensure it is persisted and try to configure the cluster state to reflect the object description

Working with Kubernetes objects

- Objects are managed via the Kubernetes API
  - primarily through the kubectl CLI which makes the API calls.
- Every object includes two important fields for its configuration
  - spec - description of the desired state of a particular resource (pod, application, etc)
  - status - describes the current state of the object
- The Kubernetes control plane continually and actively manages every object's actual state to match the desired spec as far as possible



KUBERNETES NATIVE OBJECTS - PART 01

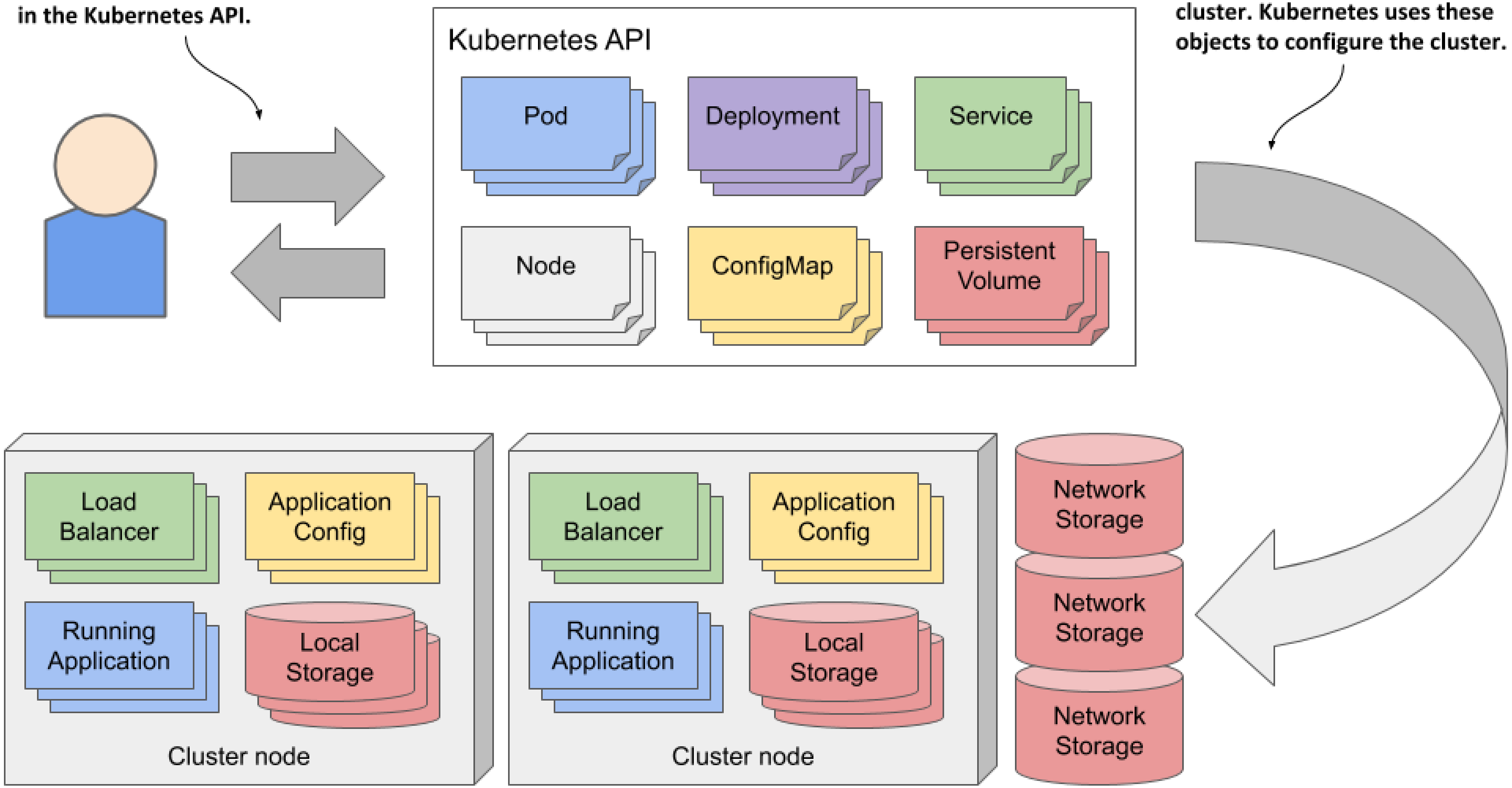
devopscube

Deployment Unit Pod	Networking Service	Storage Volume	Isolation Namespace
Replication Replicaset	POD Management Deployment	State Management StatefulSet	Node Operation DaemonSet
Task Execution Job	Scheduled Tasks CronJob	Sensitive Data Secret	Configuration ConfigMap
External Access Ingress	Network Rules NetworkPolicy	Persistent Storage Persistent Volume	Storage Request PV Claim
Scalability HPA/VPA	Network endpoints EndpointSlices	Authentication ServiceAccount	Authorization Role/ClusterRole

Objects to configure cluster state

You control most aspects of the cluster by manipulating objects in the Kubernetes API.

API objects represent virtually everything that exists in the cluster. Kubernetes uses these objects to configure the cluster.



Commonly used objects

- Pods
- ReplicaSets
- Deployments
- Services
- Volumes
- Namespaces

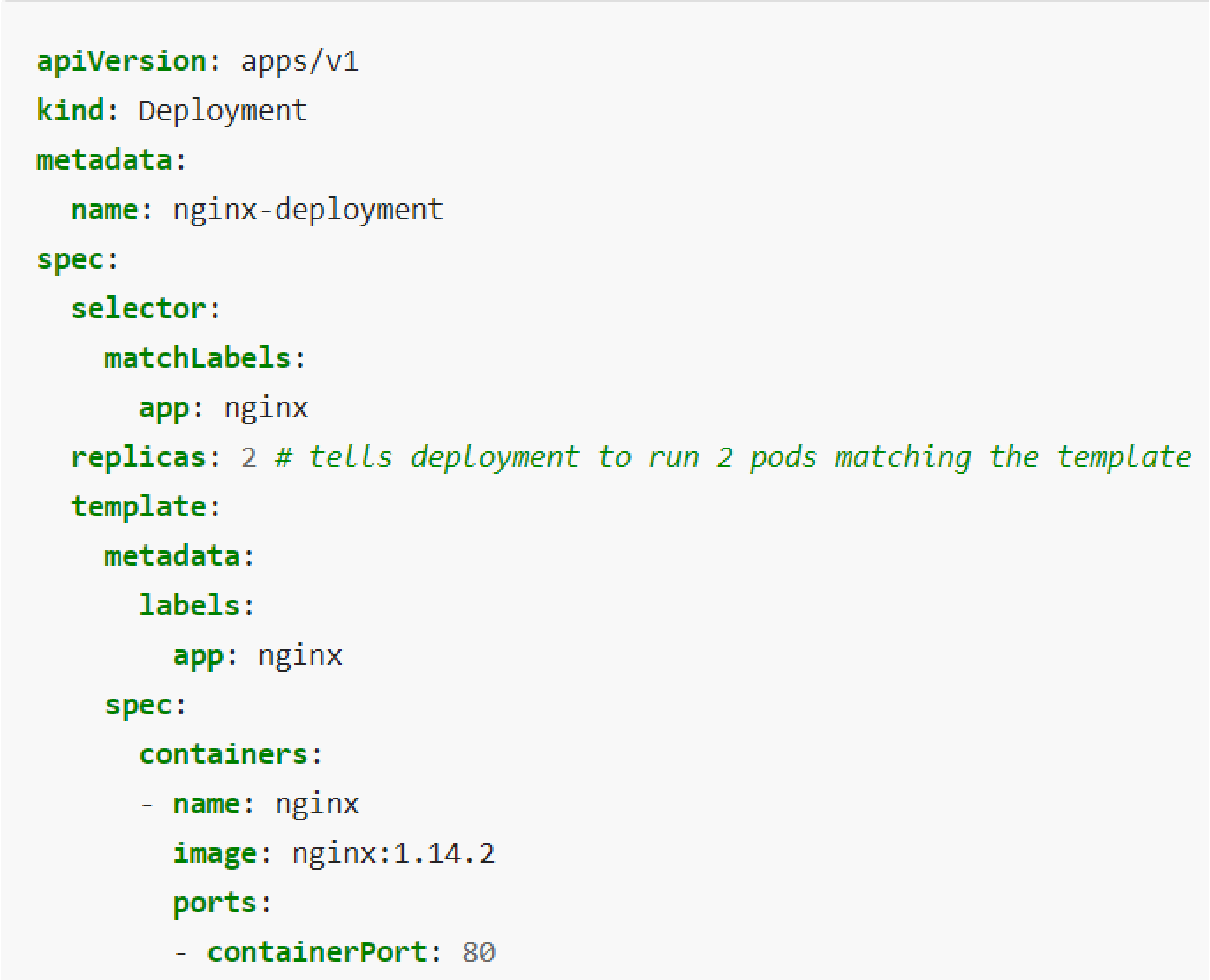
Workload Resources

- A workload is an application running on Kubernetes
  - They are run inside a pod, where each pod contains one or more containers
  - Workload resources are created to manage a set of pods rather than managing pods individually
- Most commonly used Workload Resources (which are also objects)
  - Deployment and ReplicaSet
  - StatefulSet
  - DaemonSet
  - Job and CronJob

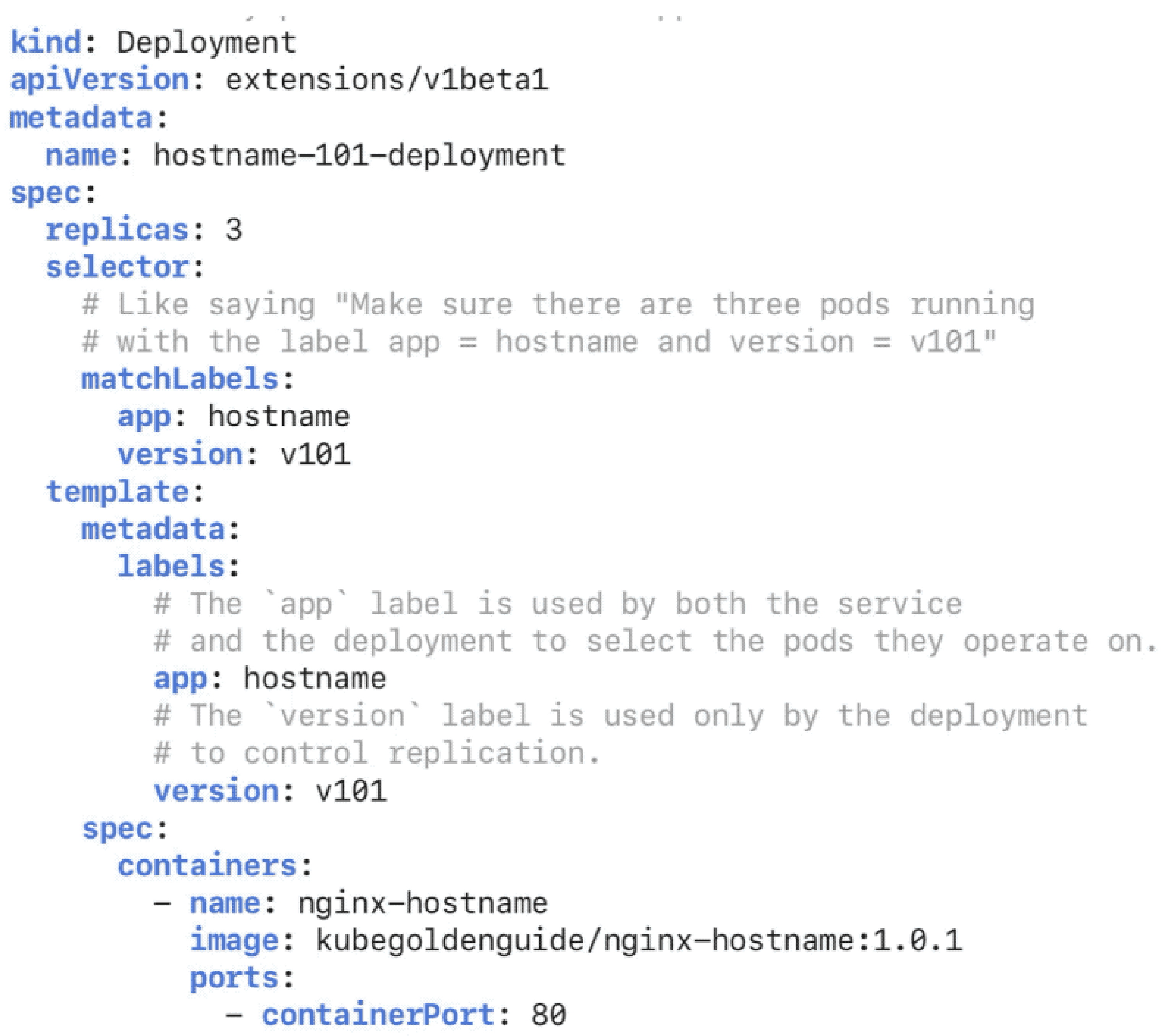
Describing Kubernetes objects

- Creating an object requires providing:
  - spec - description of desired state
  - basic info - name, etc
- Most common way of providing description
  - manifest YAML file

Sample YAML manifest for deployment



```yaml
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
```



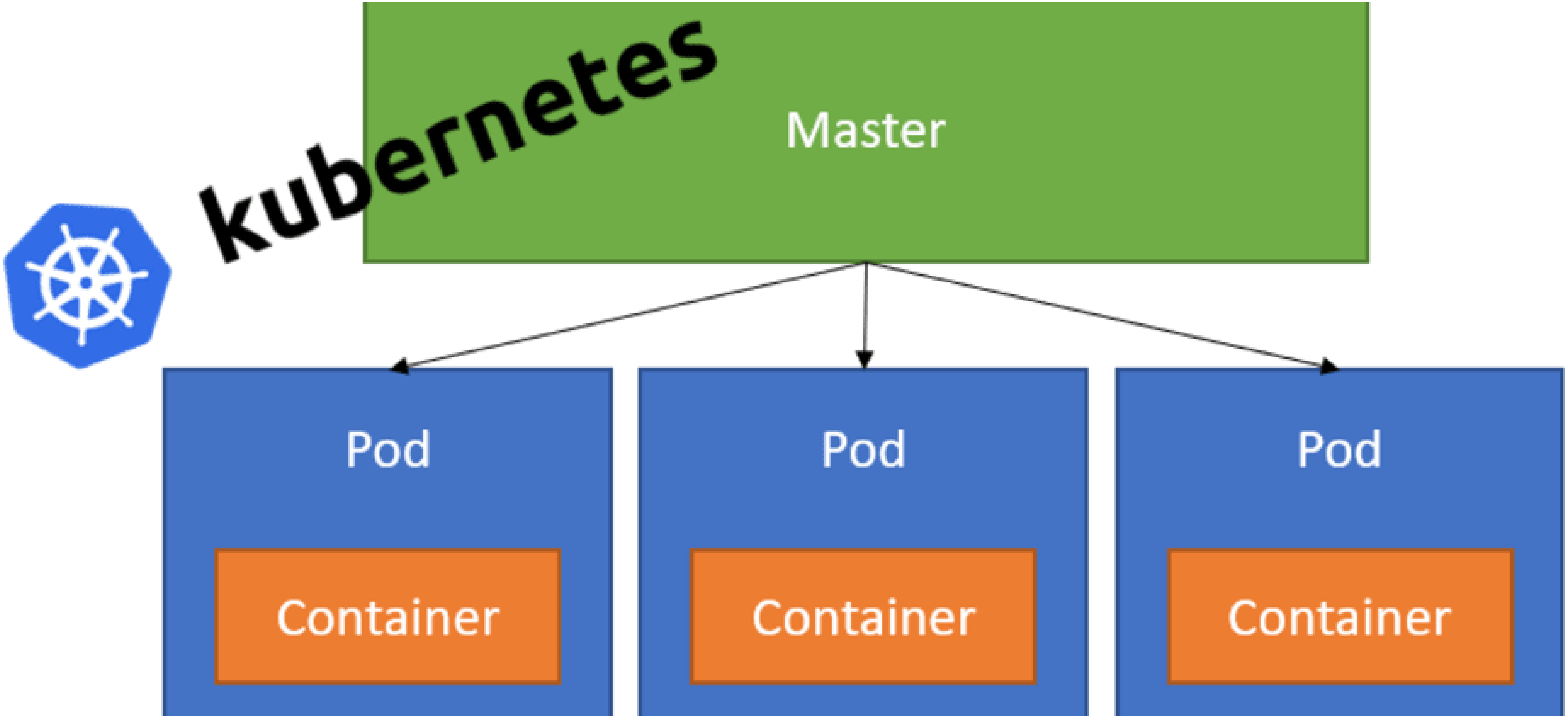
```yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: hostname-101-deployment
spec:
  replicas: 3
  selector:
    # Like saying "Make sure there are three pods running
    # with the label app = hostname and version = v101"
    matchLabels:
      app: hostname
      version: v101
  template:
    metadata:
      labels:
        # The `app` label is used by both the service
        # and the deployment to select the pods they operate on.
        app: hostname
        # The `version` label is used only by the deployment
        # to control replication.
        version: v101
    spec:
      containers:
        - name: nginx-hostname
          image: kubegoldenguide/nginx-hostname:1.0.1
          ports:
            - containerPort: 80
```

Pods

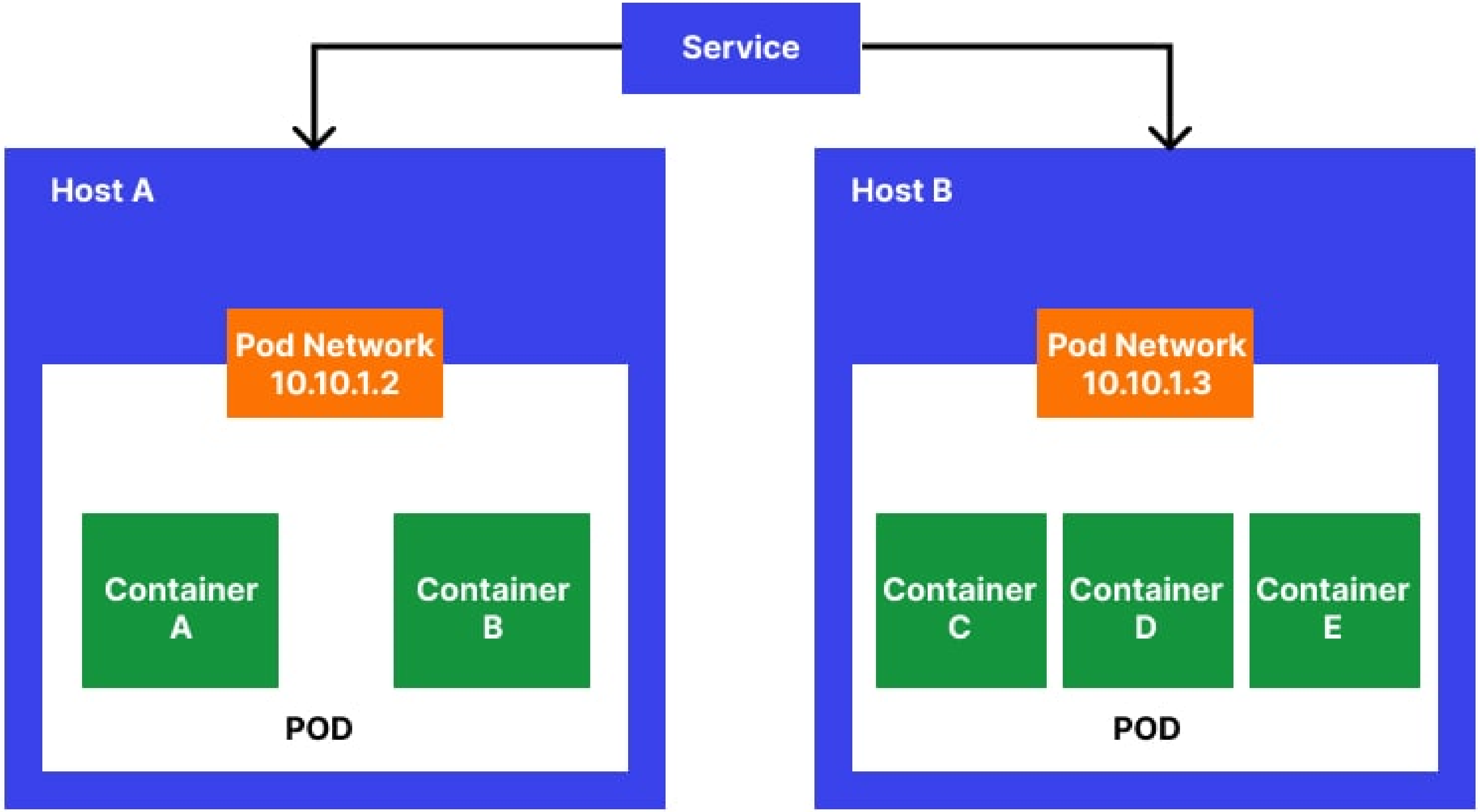
- ❖smallest deployable unit of computing in Kubernetes
  - a pod contains one more containers with shared storage and network resources
  - reside on worker nodes and have their own IP addresses
  - includes configuration on running its containers
- ❖pods are ephemeral
  - can fail or be explicitly removed
  - Kubernetes automatically replaces each failed pod with a new pod replica to keep cluster running

Containers in pods

- one-container-per-Pod
  - the most common use case; Pod acts as a wrapper around a single container
- multiple containers within a Pod
  - encapsulate an application composed of multiple co-located containers that are tightly coupled and need to share network / storage resources (advanced use case)


kubernetes
Master
Pod
Container
Pod
Container
Pod
Container

Kubernetes pod architecture



Working with Pods

- Workload resource objects are used for managing pods
  - Examples: Deployment and replicaset, StatefulSet, etc
- Each Pod is meant to run a single instance of a given application.
  - To scale application horizontally, use multiple replicated Pods one for each instance.
  - Replicated Pods are usually created and managed as a group by a workload resource and its controller.

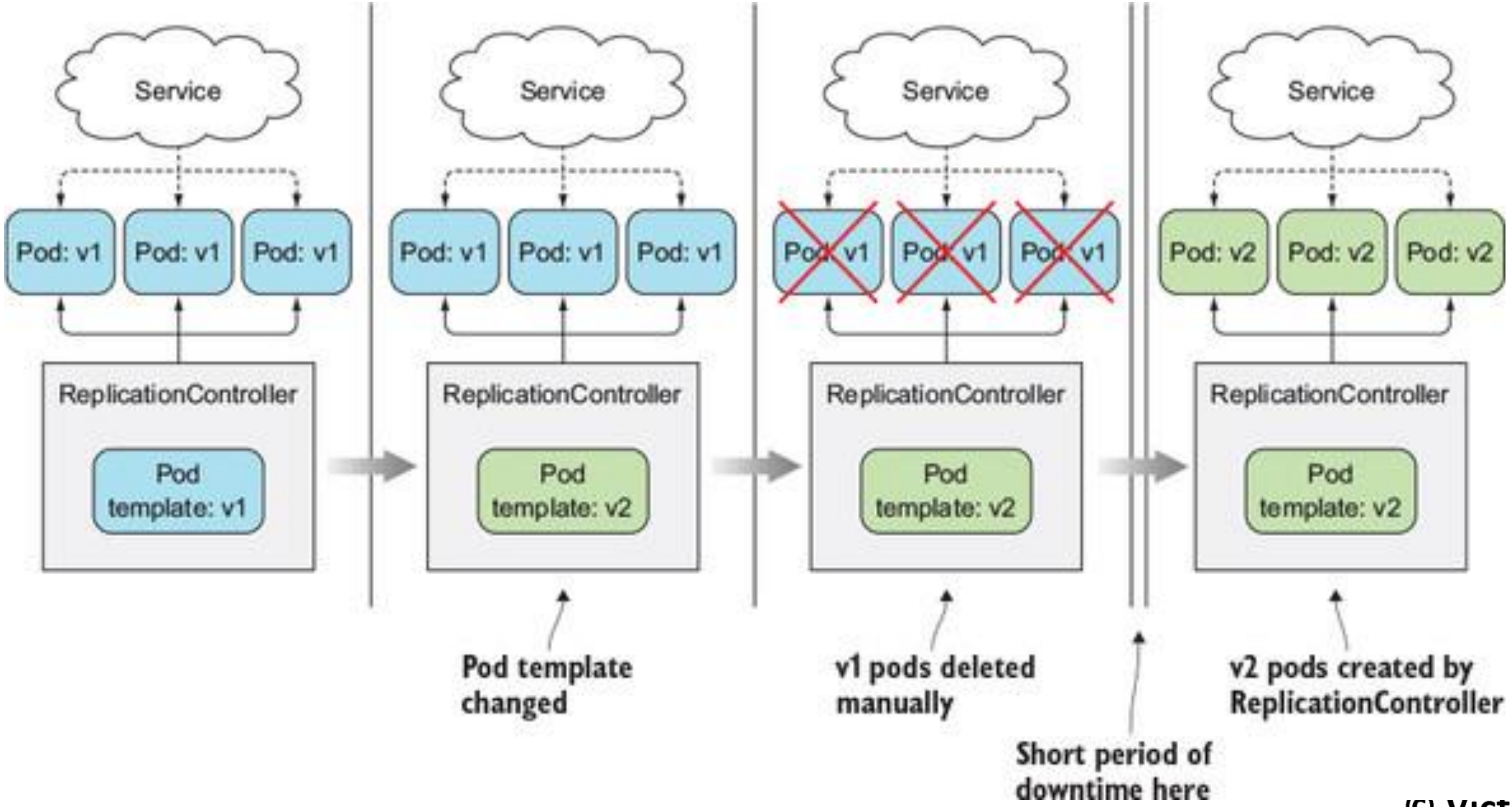
Working with Pods

- When a Pod gets created, it is scheduled to run on a worker node in the cluster
  - Pod remains on that node until the Pod finishes execution, the Pod object is deleted, the Pod is evicted for lack of resources, or the node fails
- Workload resource controllers handles replication and rollout and automatic healing in case of Pod failure

Pod templates

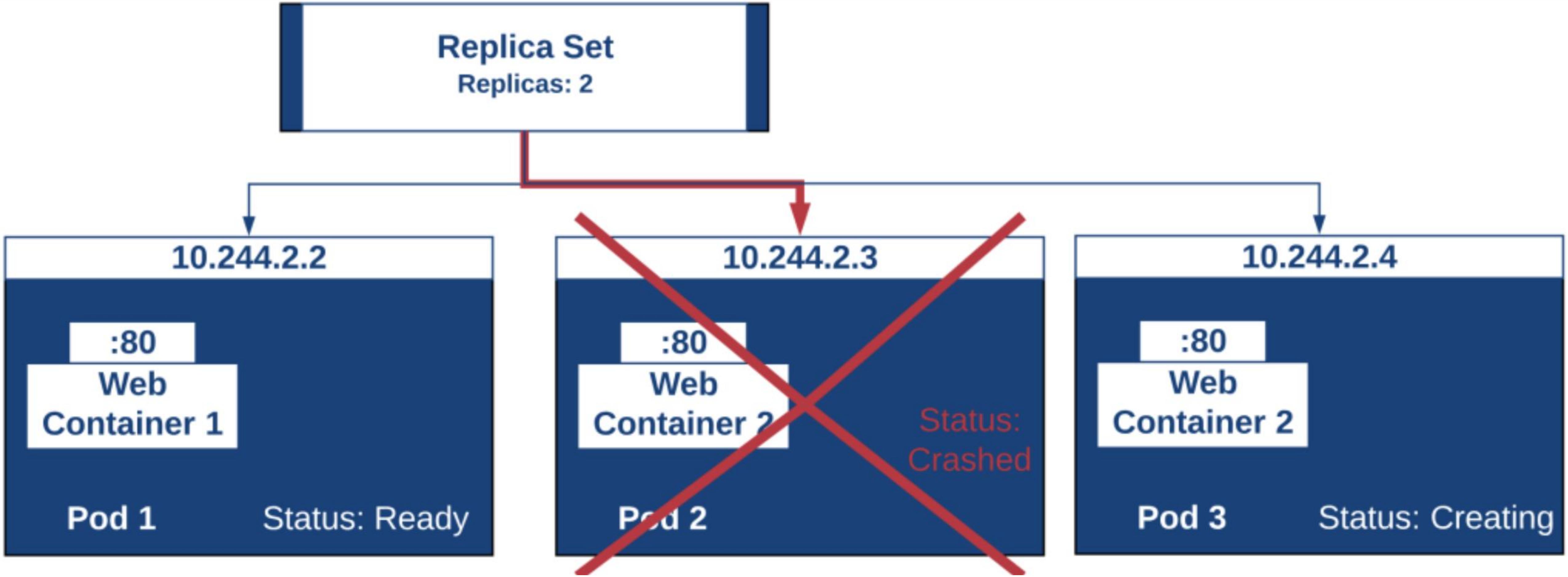
- Controllers for workload resources create Pods from a pod template
  - Each controller for a workload resource uses the PodTemplate inside the workload object to make actual Pods
- The PodTemplate is part of the desired state of whatever resource object is used to run your app.
  - When the Pod template for a workload resource is changed, the controller creates new Pods based on the updated template instead of updating or patching the existing Pods.

Pod templates



ReplicaSet

- ensures that the desired number of pod replicas are available at all times
  - by either creating or deleting Pod replicas dynamically as pod numbers change
- Use indirectly by a deployment object
- includes a selector that specifies
  - how to identify Pods it can acquire
  - the number of replicas for that Pod
  - a pod template specifying the configuration of new Pods it should create

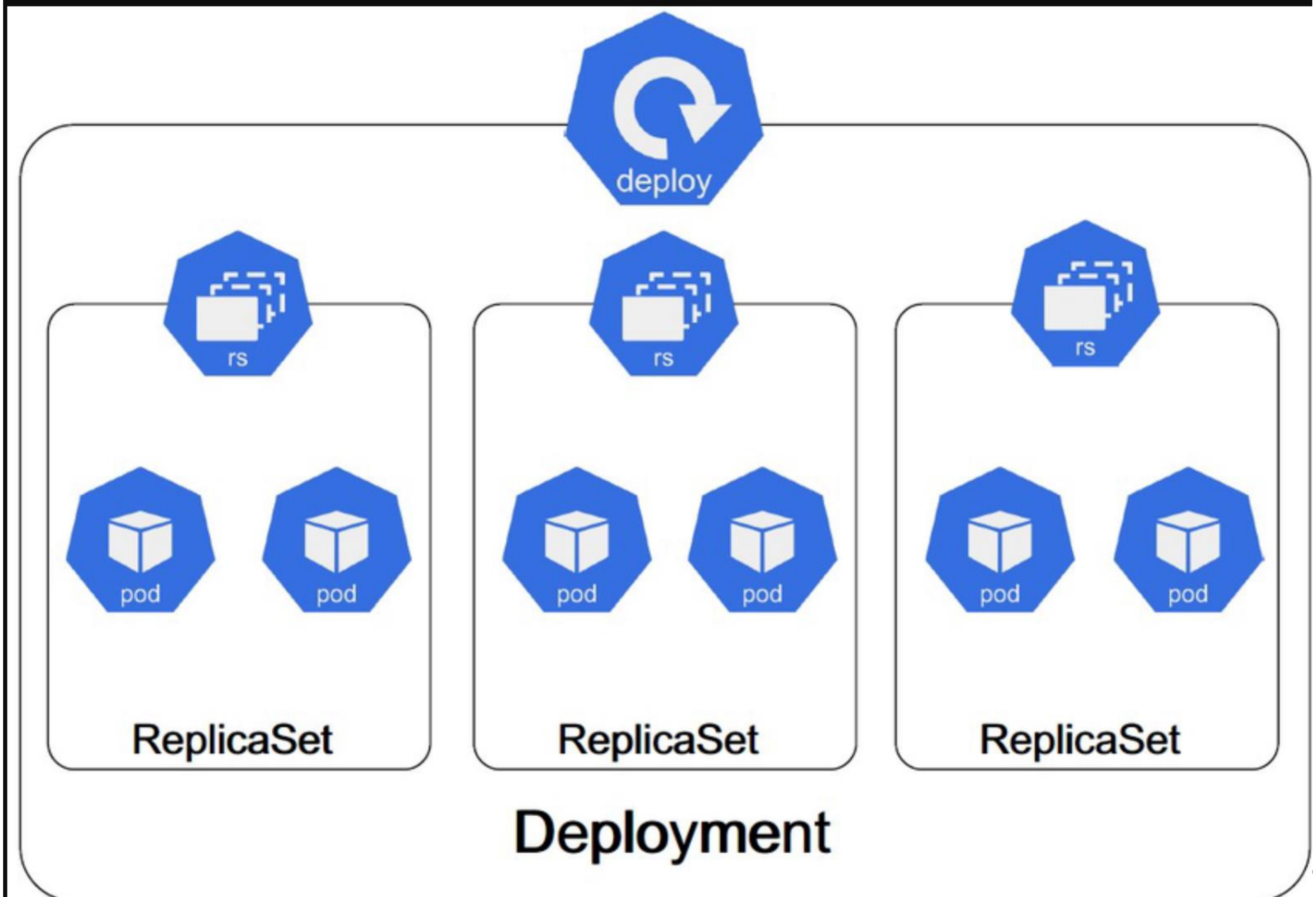

Replica Set
Replicas: 2
10.244.2.2
:80
Web Container 1
Pod 1
Status: Ready
10.244.2.3
:80
Web Container 2
Status: Crashed
Pod 2
10.244.2.4
:80
Web Container 2
Pod 3
Status: Creating

Deployments

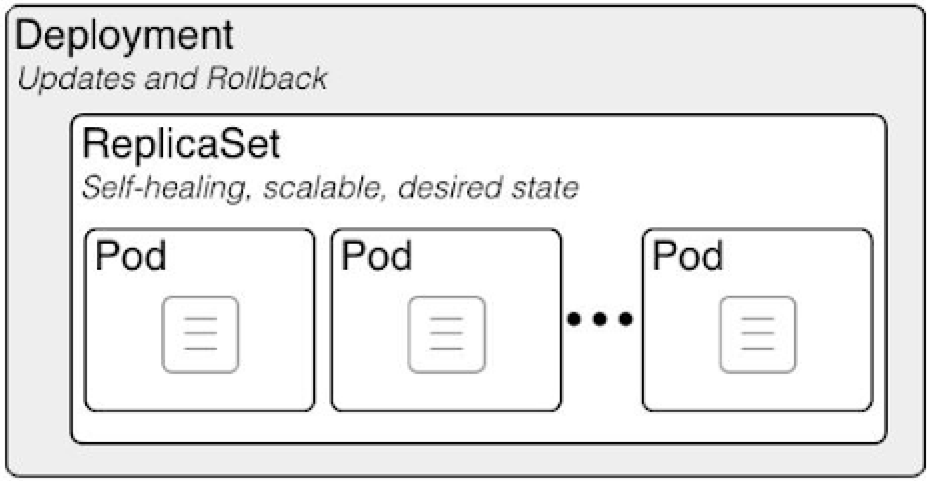
- used to manage the lifecycle of one or more pod replicas
  - allows declarative specification of the desired state of your application
  - e.g. number of replicas, image for containers in pods, resources required such as volumes
- usually done through a pod template in the manifest for the deployment
  - uses a ReplicaSet to manage pod replicas that match this template

Deployment typical use cases

- Create a Deployment to rollout a ReplicaSet
- Declare the new state of the Pods by updating the Pod Template of the Deployment
- Rollback to an earlier Deployment revision
- Scale up the Deployment to facilitate more load


deploy
rs
rs
rs
pod
pod
pod
pod
pod
pod
ReplicaSet
ReplicaSet
ReplicaSet
Deployment

Tan 2024



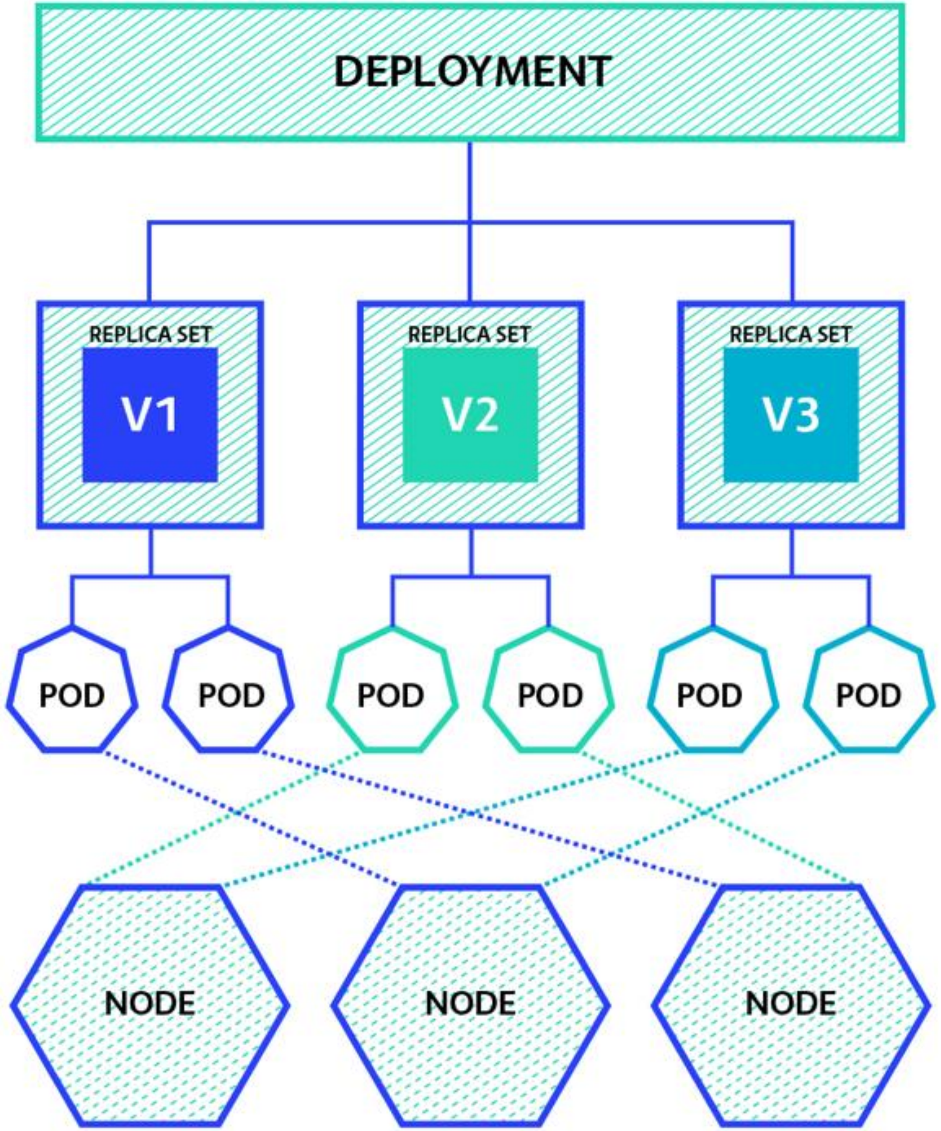
Deployment
Updates and Rollback

ReplicaSet
Self-healing, scalable, desired state

Pod

Pod

Pod



DEPLOYMENT

REPLICA SET
V1

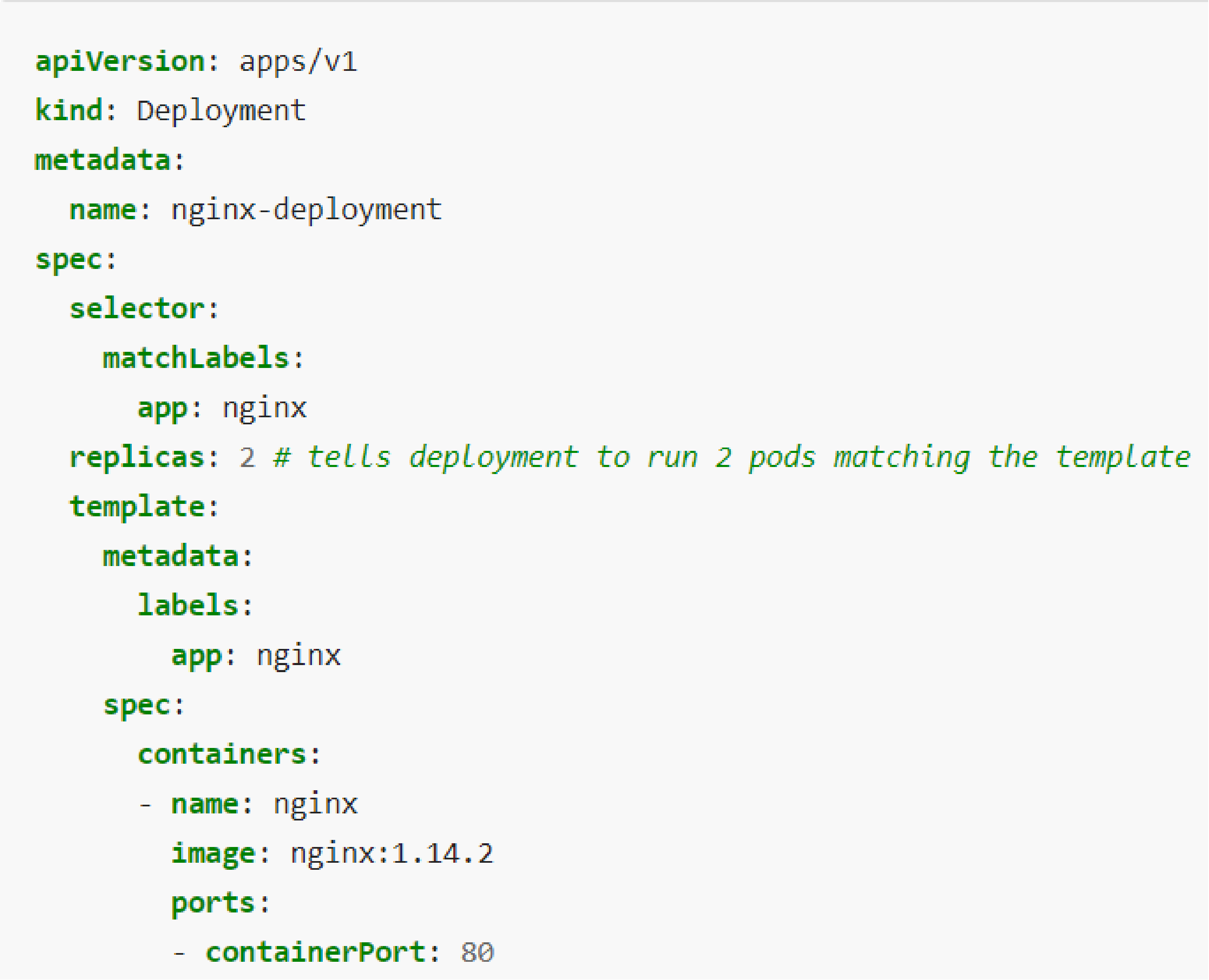
REPLICA SET
V2

REPLICA SET
V3

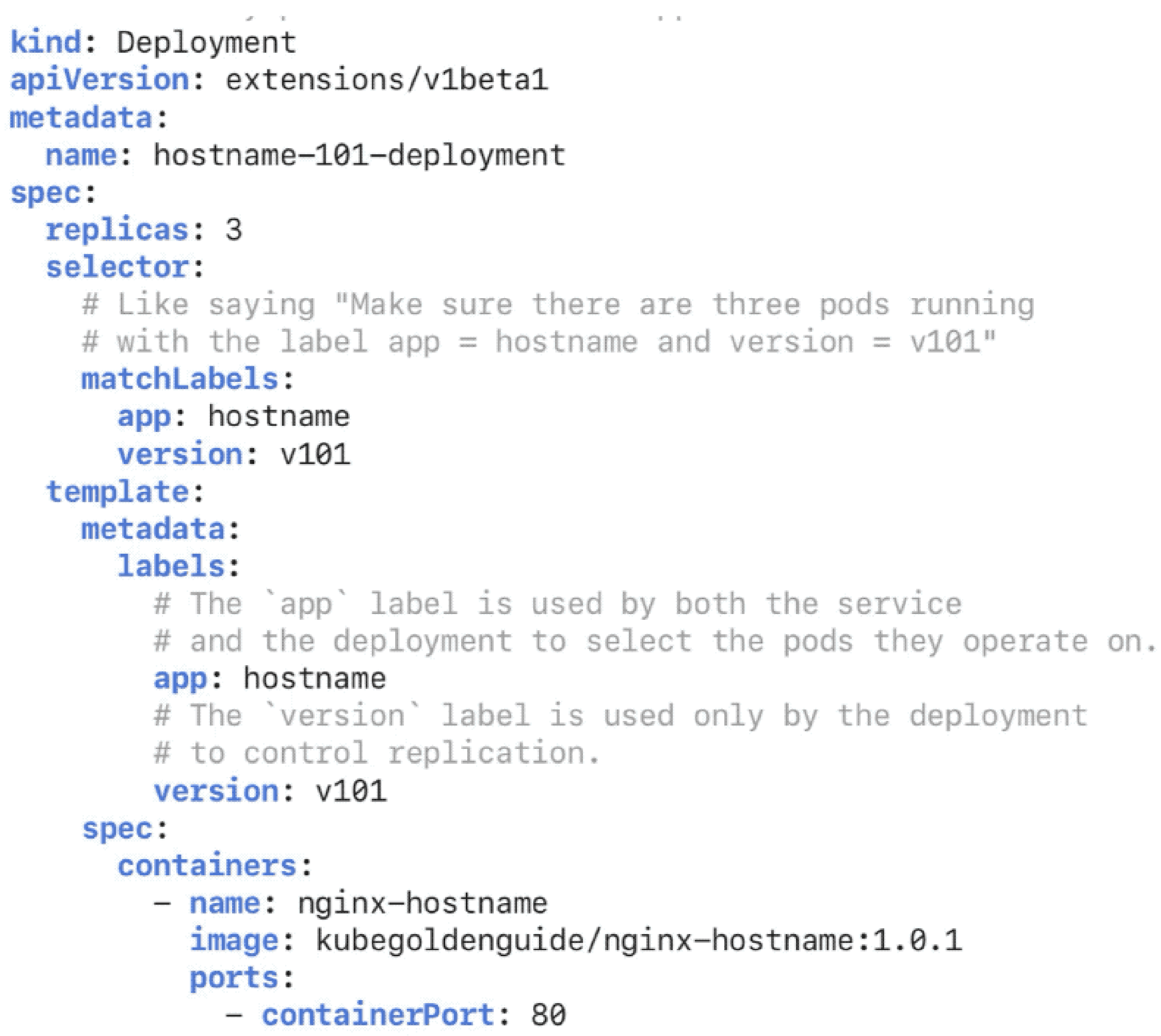
POD POD POD POD POD POD

NODE NODE NODE

Sample YAML manifest for deployment



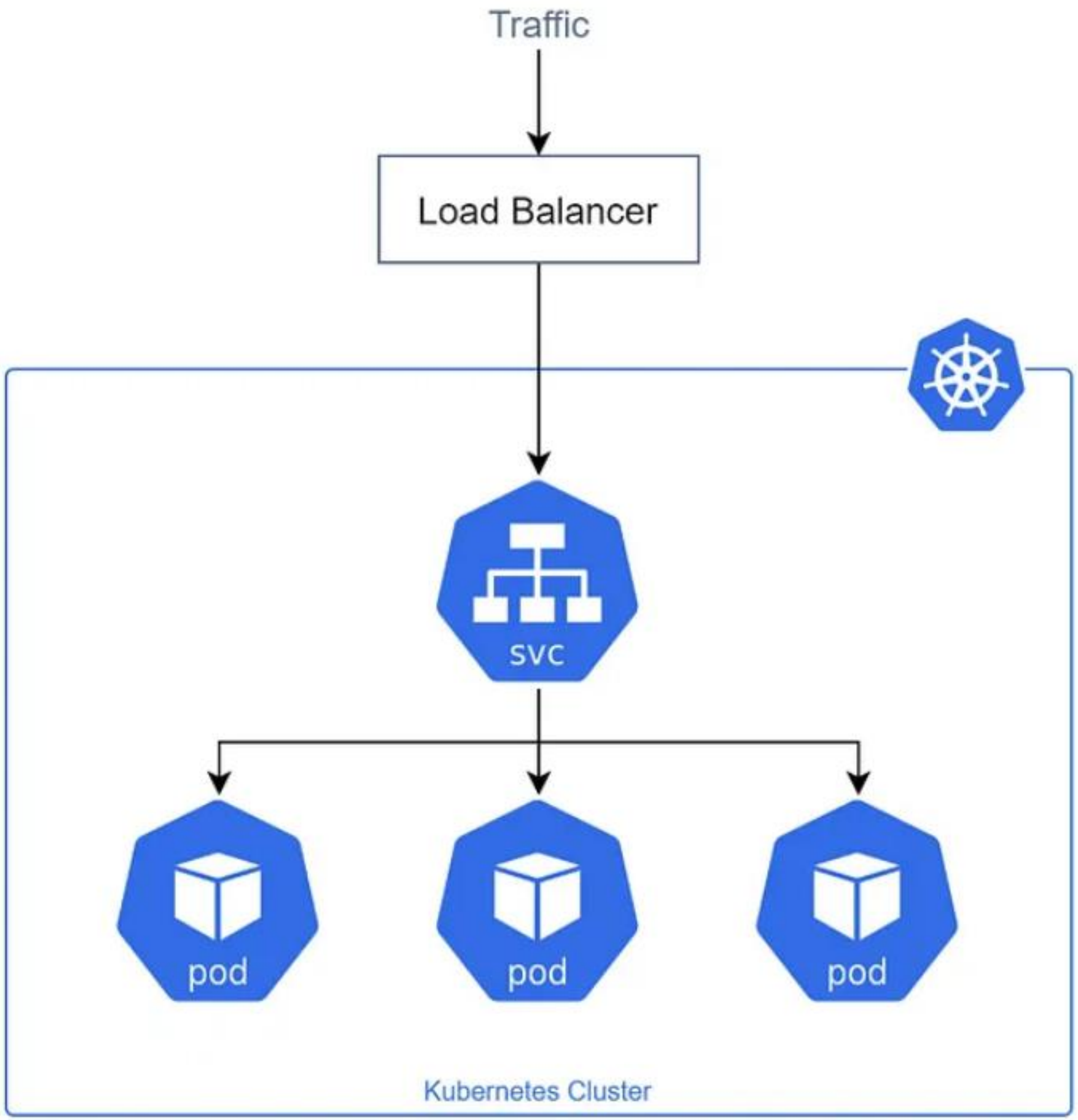
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: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



```yaml
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: hostname-101-deployment
spec:
  replicas: 3
  selector:
    # Like saying "Make sure there are three pods running
    # with the label app = hostname and version = v101"
    matchLabels:
      app: hostname
      version: v101
  template:
    metadata:
      labels:
        # The `app` label is used by both the service
        # and the deployment to select the pods they operate on.
        app: hostname
        # The `version` label is used only by the deployment
        # to control replication.
        version: v101
    spec:
      containers:
        - name: nginx-hostname
          image: kubegoldenguide/nginx-hostname:1.0.1
          ports:
            - containerPort: 80
```

Service

- provides an easy way to access a group of Pod replicas that have identical functionality
  - exposes a single, fixed point of network access for external clients via kube-proxy
  - requests from clients are then routed to the Pods (which may have dynamic IP addresses when they are created and replaced)
- also can perform load balancing by distributing incoming traffic evenly to all pod replicas



Traffic

Load Balancer

svc

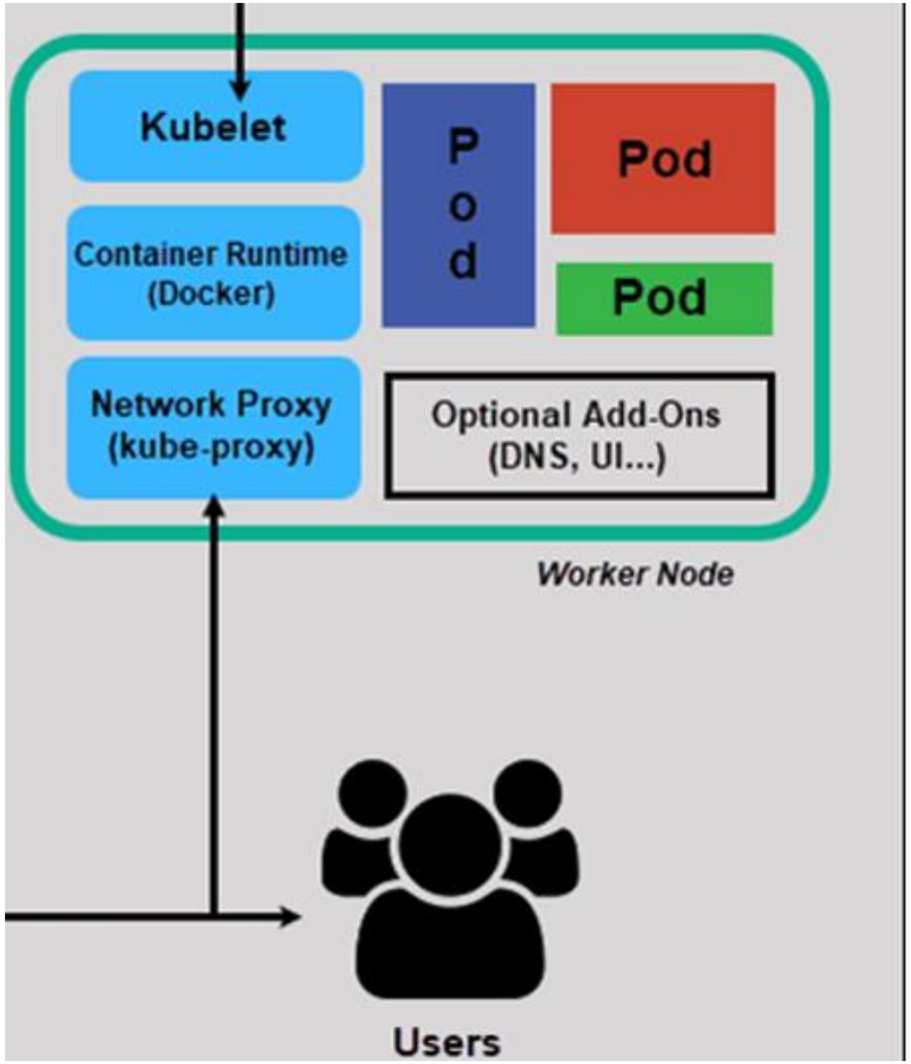
pod

pod

pod

Kubernetes Cluster

LoadBalancer



Kubelet

Container Runtime (Docker)

Network Proxy (kube-proxy)

Pod

Pod

Pod

Optional Add-Ons (DNS, UI...)

Worker Node

Users

© Victor Tan 2024