

The background of the slide is a solid blue color. On the left side, there are several overlapping geometric shapes: a dark blue parallelogram, a medium blue parallelogram, and a light blue parallelogram, all slanted at an angle. In the center-right background, there is a large, faint, light blue hexagon containing a ship's wheel, which is the Kubernetes logo.

KUBERNETES

Agenda


- Introduction
 - About me
 - What is Kubernetes?
 - What does Kubernetes do?
- Architecture
 - Master Components
 - Node Components
 - Additional Services
 - Networking
- Concepts
 - Core
 - Workloads
 - Network
 - Storage
 - Configuration
 - Auth and Identity
- Behind the Scenes
 - Deployment from Beginning to End



Introduction



About me

- Senior Staff Software Engineer @ AMD AI Group
 - Master's in Computer Science
NYU Tandon School of Engineering (2023)
 - Teaching Assistant
CS:9223 Cloud Computing & Big Data, NYU Tandon (4 semesters)
 - **LinkedIn:** [linkedin.com/in/prateek1709](https://www.linkedin.com/in/prateek1709)
- 



Intro - What is Kubernetes?

- Kubernetes (K8s) is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.
 - Originally developed by Google, Kubernetes is now maintained by the Cloud Native Computing Foundation (CNCF) and has gained widespread adoption in the industry.
 - At its core, Kubernetes provides a platform for automating the deployment, scaling, and management of containerized applications
- 

Intro - What Does Kubernetes do?

- Container Orchestration: Kubernetes schedules and manages containers, ensuring they run in a reliable and scalable manner.
- Automated Scaling: It can automatically scale applications up or down based on demand.
- Load Balancing: Kubernetes can distribute traffic across multiple containers for high availability.
- Self-Healing: It detects and replaces failed containers to maintain application health.
- Rollouts and Rollbacks: Kubernetes facilitates controlled updates and rollbacks of application versions.
- Resource Management: It efficiently allocates computing resources to containers.



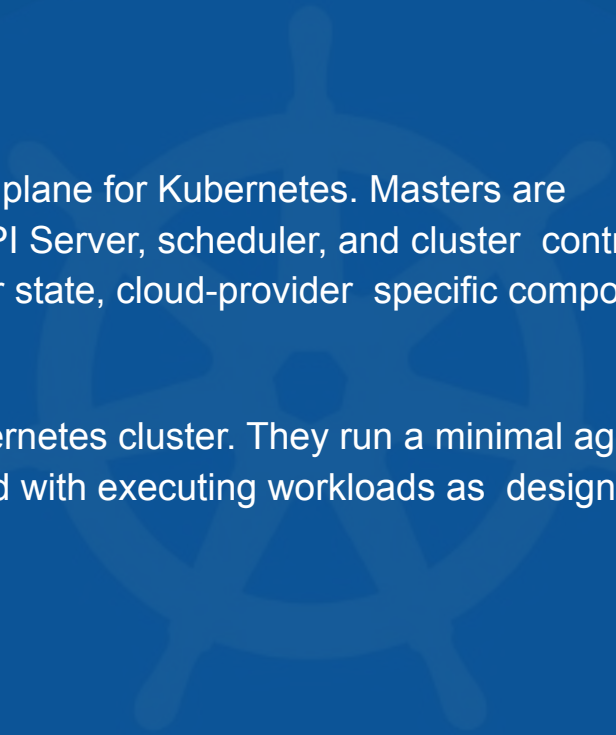
Architecture



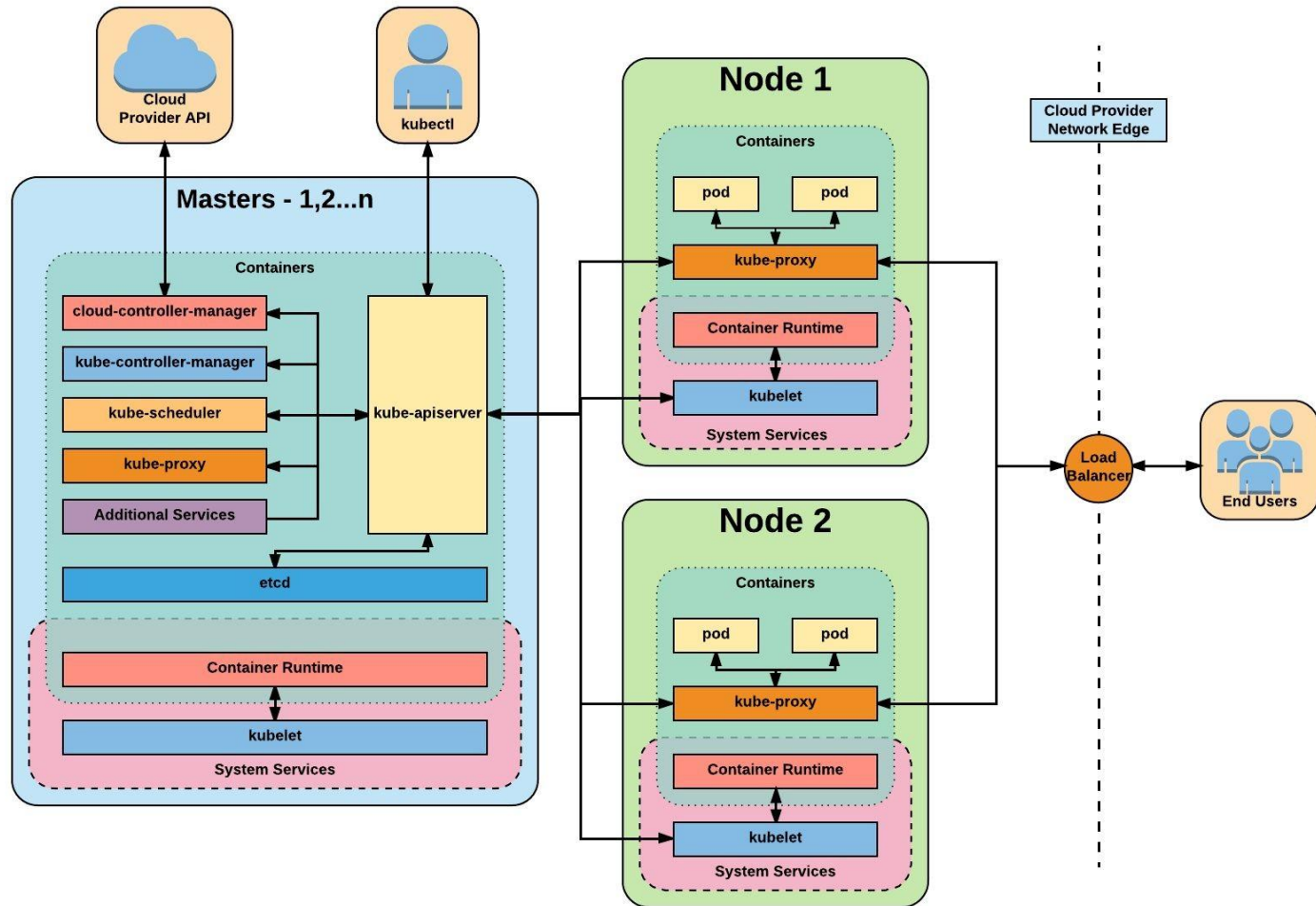
Architecture Overview

Master Nodes - Acts as the primary control plane for Kubernetes. Masters are responsible at a minimum for running the API Server, scheduler, and cluster controller. They commonly also manage storing cluster state, cloud-provider specific components and other cluster essential services.

Worker Nodes - Are the 'workers' of a Kubernetes cluster. They run a minimal agent that manages the node itself, and are tasked with executing workloads as designated by the master.



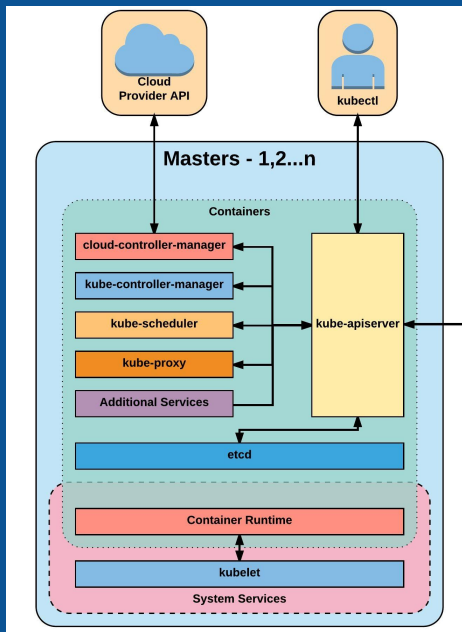
Architecture Overview





Master Components

Master Components



- Kube-apiserver
- Etcd
- Kube-controller-manager
- Cloud-controller-manager
- Kube-scheduler



kube-apiserver

Serves as the central control point for the entire Kubernetes cluster.

Exposes the Kubernetes API, which allows users, administrators, and other components to interact with the cluster.

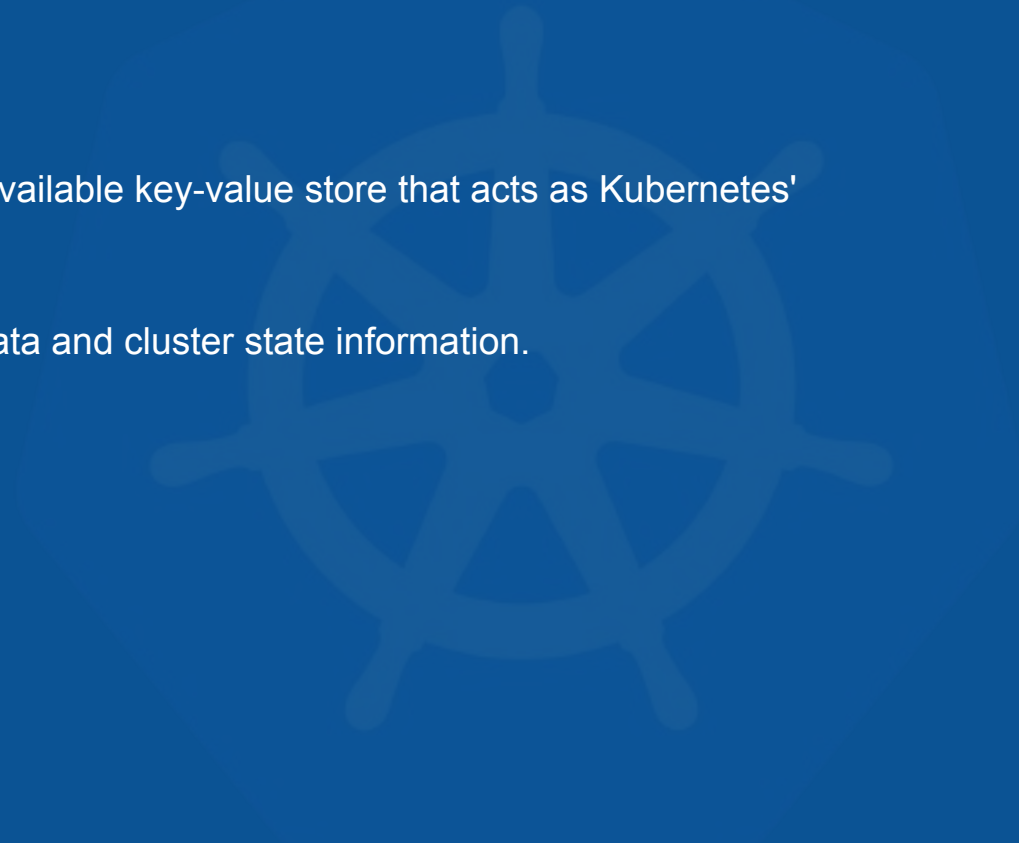
Accepts and processes RESTful API requests, which can include creating, updating, and deleting resources like Pods, Services, ConfigMaps, and more.

Implements authentication, authorization, and admission control for security.



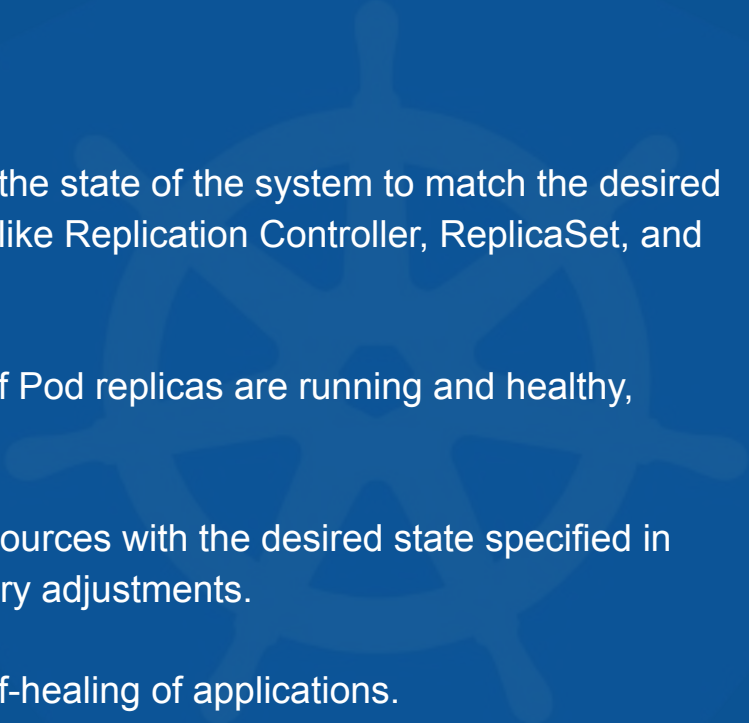


etcd

- A distributed and highly-available key-value store that acts as Kubernetes' primary database.
 - Stores all configuration data and cluster state information.
- 



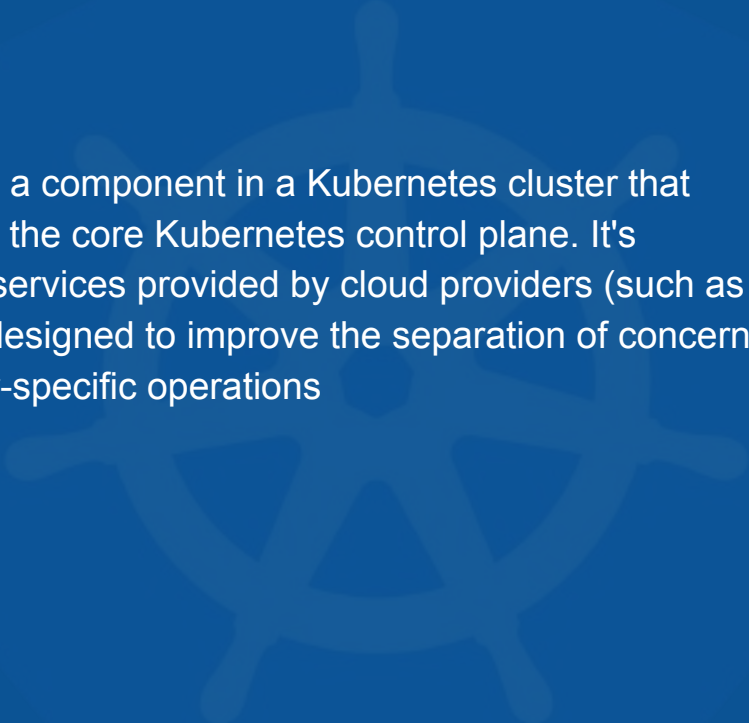
kube-controller-manager

- Manages controllers that regulate the state of the system to match the desired state. Includes various controllers like Replication Controller, ReplicaSet, and StatefulSet.
 - Ensures that the correct number of Pod replicas are running and healthy, helping in fault tolerance.
 - Reconciles the current state of resources with the desired state specified in the configuration, making necessary adjustments.
 - Helps in automatic scaling and self-healing of applications.
- 



cloud-controller-manager

The Cloud Controller Manager (CCM) is a component in a Kubernetes cluster that offloads cloud-specific functionality from the core Kubernetes control plane. It's primarily used in managed Kubernetes services provided by cloud providers (such as AWS EKS, GKE, or Azure AKS) and is designed to improve the separation of concerns between Kubernetes and cloud provider-specific operations





kube-scheduler

Monitors the API Server for new Pods without assigned nodes.

Selects an appropriate node for each Pod based on resource requirements, constraints, and policies.

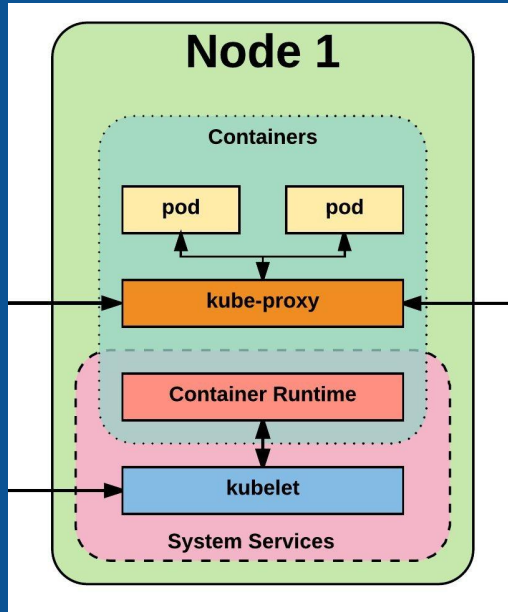
Takes into account factors like CPU and memory availability, anti-affinity rules, and node capacity.





Node Components

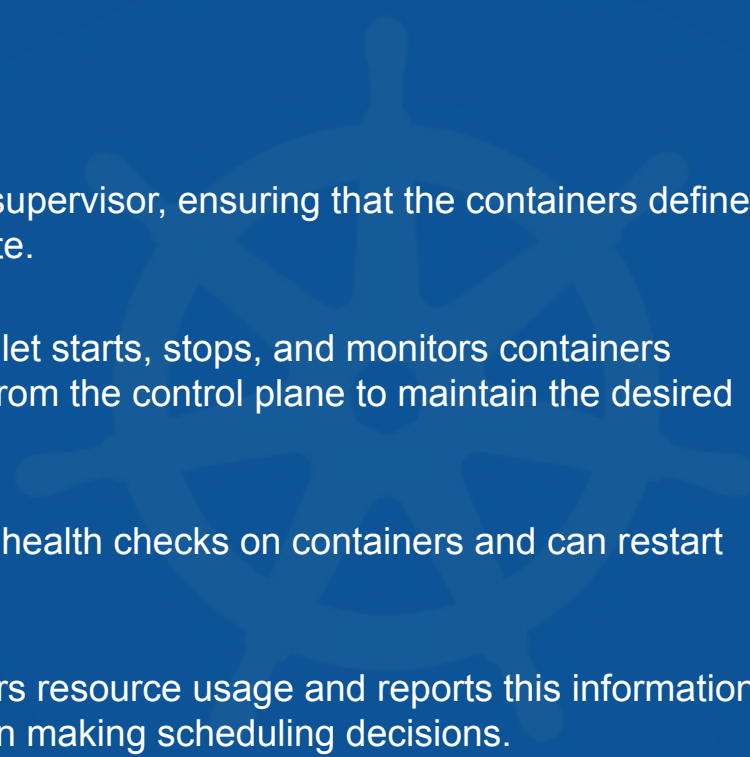
Node Components



- Kubelet
- Kube-proxy
- Container runtime engine

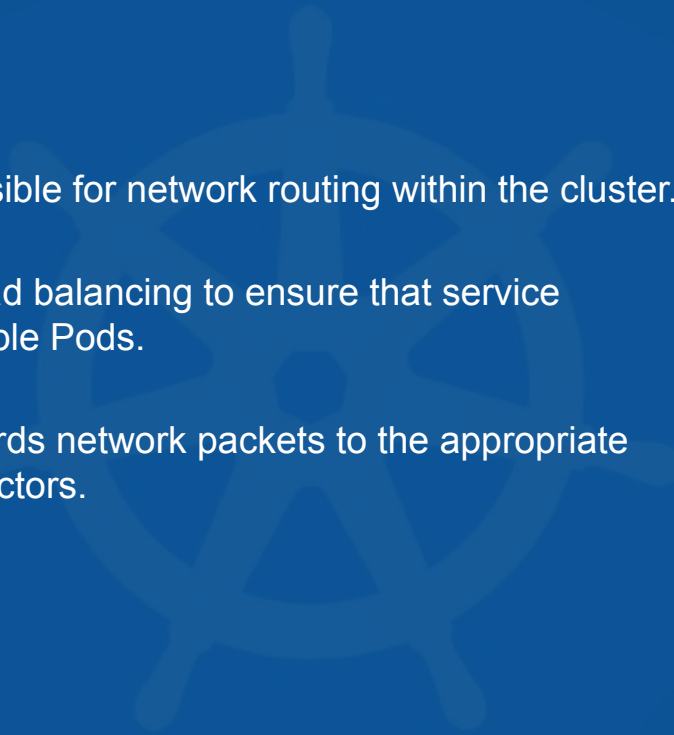


kubelet

- Pod Supervisor: It acts as a Pod supervisor, ensuring that the containers defined within Pods are in the desired state.
 - Pod Lifecycle Management: Kubelet starts, stops, and monitors containers within Pods. It takes instructions from the control plane to maintain the desired number of Pod replicas.
 - Health Checks: Kubelet performs health checks on containers and can restart them if they fail.
 - Resource Management: It monitors resource usage and reports this information to the control plane, which helps in making scheduling decisions.
- 



kube-proxy

- Network Proxy: Kube Proxy is responsible for network routing within the cluster.
 - Service Load Balancing: It enables load balancing to ensure that service requests are distributed among available Pods.
 - Packet Forwarding: Kube Proxy forwards network packets to the appropriate destination Pod based on service selectors.
- 

Container Runtime

With respect to Kubernetes, A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.

- Containerd (docker)
- Cri-o
- Rkt
- Kata (formerly clear and hyper)
- Virtlet (VM CRI compatible runtime)

The background of the slide is a solid blue color. On the left side, there are several overlapping geometric shapes: a dark blue parallelogram, a medium blue parallelogram, and a light blue parallelogram, all slanted at an angle. In the center-right background, there is a large, faint, light blue ship's wheel, which is the logo for Kubernetes.

Kubernetes Concepts

Concepts - Core (cont.)

Label - Key-value pairs that are used to **identify**, describe and group together related sets of objects. Labels have a strict syntax and available character set. *

Annotation - Key-value pairs that contain **non-identifying** information or metadata. Annotations do not have the the syntax limitations as labels and can contain structured or unstructured data.

Selector - Selectors use labels to filter or select objects. Both equality-based (=, ==, !=) or simple key-value matching selectors are supported.

* <https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set>

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx
  annotations:
    description: "nginx frontend"
  labels:
    app: nginx
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
      tier: frontend
  template:
    metadata:
      labels:
        app: nginx
        tier: frontend
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

Labels, Annotations and Selectors

Labels:
app: nginx
tier: frontend

Annotations
description: "nginx
frontend"

Selector:
app:
nginx
tier: frontend

Concepts - Workloads

Pod - A pod is the smallest unit of work or management resource within Kubernetes. It is comprised of one or more containers that share their storage, network, and context (namespace, cgroups etc).

ReplicationController - Method of managing pod replicas and their lifecycle. Their scheduling, scaling, and deletion.

ReplicaSet - Next Generation ReplicationController. Supports set-based selectors.

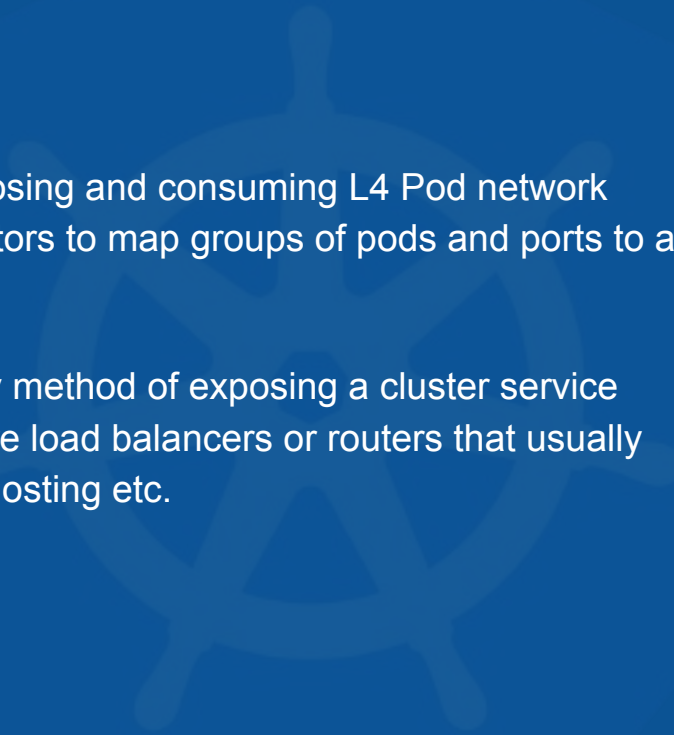
Deployment - A declarative method of managing stateless Pods and ReplicaSets. Provides rollback functionality in addition to more granular update control mechanisms.



Concepts - Network

Service - Services provide a method of exposing and consuming L4 Pod network accessible resources. They use label selectors to map groups of pods and ports to a cluster-unique virtual IP.

Ingress - An ingress controller is the primary method of exposing a cluster service (usually http) to the outside world. These are load balancers or routers that usually offer SSL termination, name-based virtual hosting etc.



Service

- Acts as the unified method of accessing replicated pods.
- Four major Service Types:
 - ClusterIP - Exposes service on a strictly cluster-internal IP (default)
 - NodePort - Service is exposed on each node's IP on a statically defined port.
 - LoadBalancer - Works in combination with a cloud provider to expose a service outside the cluster on a static external IP.
 - ExternalName - used to reference endpoints **OUTSIDE** the cluster by providing a static internally referenced DNS name.

```
kind: Service
apiVersion: v1
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Ingress Controller

- Deployed as a pod to one or more hosts
- Ingress controllers are an external controller with multiple options.
 - Nginx
 - HAproxy
 - Contour
 - Traefik
- Specific features and controller specific configuration is passed through annotations.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: "nginx"
  name: nginx-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /nginx
        backend:
          service: nginx
          servicePort: 80
```

Concepts - Storage

Volume - Storage that is tied to the Pod Lifecycle, consumable by one or more containers within the pod.

PersistentVolume - A PersistentVolume (PV) represents a storage resource. PVs are commonly linked to a backing storage resource, NFS, GCEPersistentDisk, RBD etc. and are provisioned ahead of time. Their lifecycle is handled independently from a pod.

PersistentVolumeClaim - A PersistentVolumeClaim (PVC) is a request for storage that satisfies a set of requirements instead of mapping to a storage resource directly. Commonly used with dynamically provisioned storage.

Volumes

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: www
  volumes:
  - name: www
    emptyDir: {}
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:latest
    name: nginx
    volumeMounts:
    - mountPath: /usr/share/nginx/html
      name: www
  volumes:
  - name: www
    awsElasticBlockStore:
      volumeID: <volume-id>
      fsType: ext4
```

Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs
spec:
  capacity:
    storage: 500Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /data
    server: 10.255.100.10
```

- PVs are a cluster-wide resource
- Not directly consumable by a Pod
- PV Parameters:
 - Capacity
 - accessModes
 - ReadOnlyMany (ROX)
 - ReadWriteOnce (RWO)
 - ReadWriteMany (RWX)
 - persistentVolumeReclaimPolicy
 - Retain
 - Recycle
 - Delete
 - StorageClass

Persistent Volume Claims

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 50Gi
  storageClass: slow
```

- PVCs are scoped to namespaces
- Supports accessModes like PVs
- Uses resource request model similar to Pods
- Claims will consume storage from matching PVs or StorageClasses based on *storageClass* and selectors.



Concepts - Configuration

ConfigMap - Externalized data stored within kubernetes that can be referenced as a command line argument, environment variable, or injected as a file into a volume mount. Ideal for separating containerized application from configuration.

Secret - Functionally identical to ConfigMaps, but stored encoded as base64, and encrypted at rest (if configured).





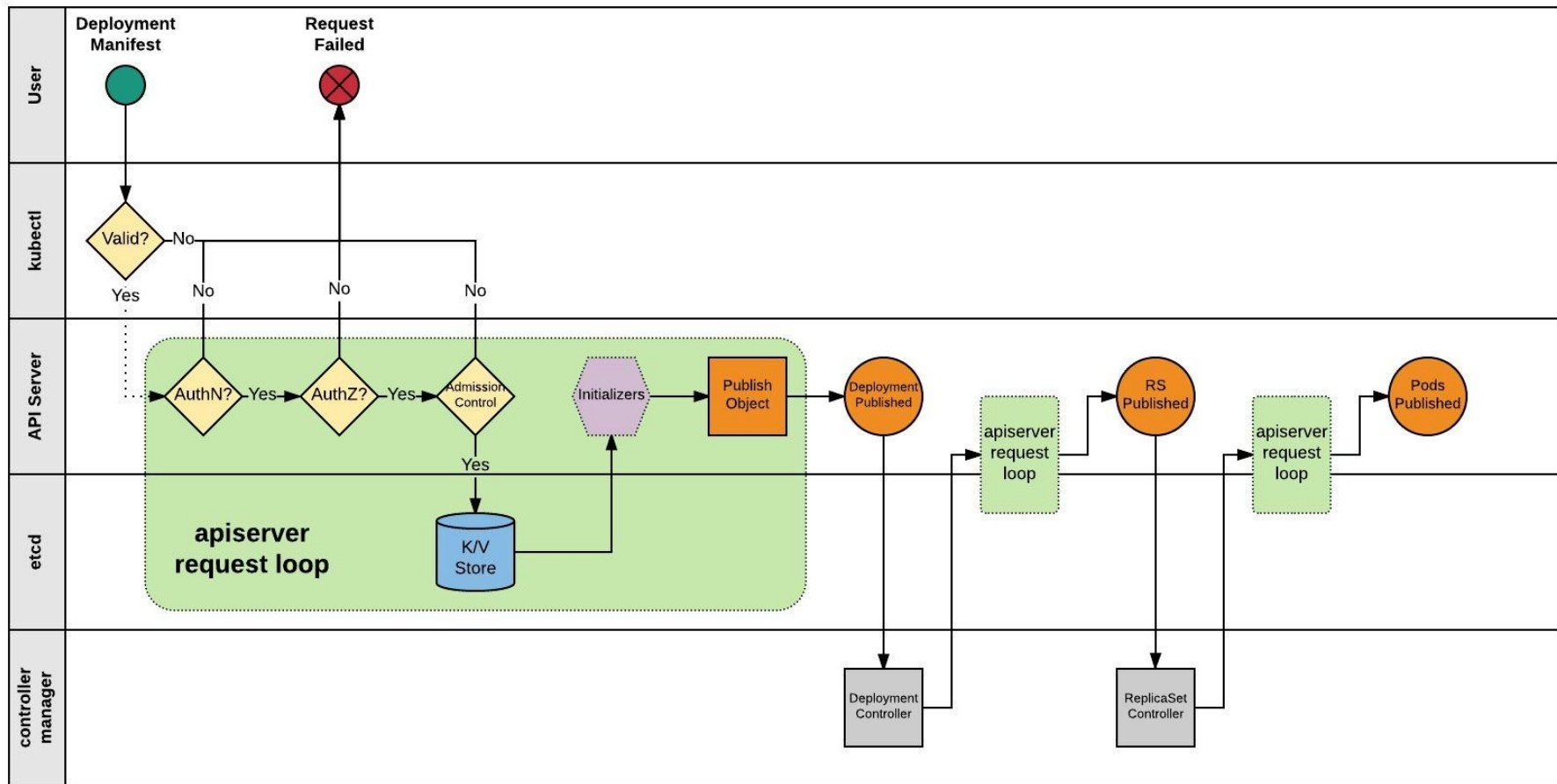
Behind The Scenes



Behind
The Scenes



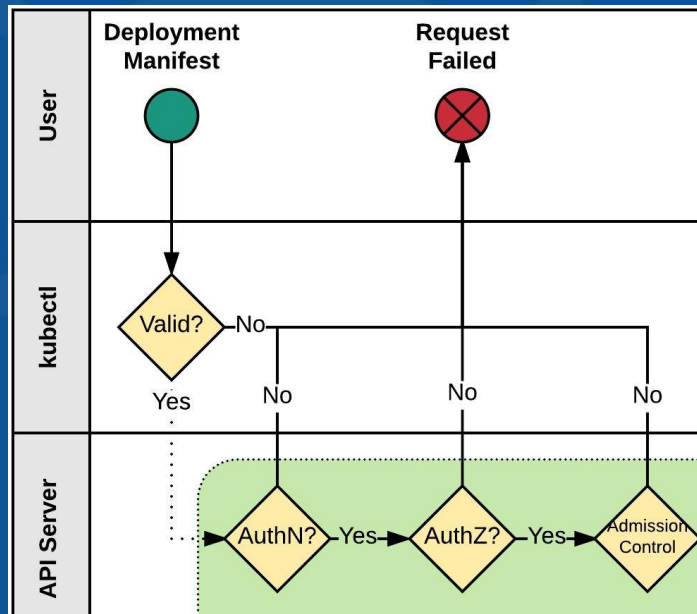
Deployment From Beginning to End



Kubectl

Kubectl performs client side validation on manifest (linting).

Manifest is prepared and serialized creating a JSON payload.



APIServer Request Loop

Kubectl authenticates to apiserver via x509, jwt, http auth proxy, other plugins, or http-basic auth.

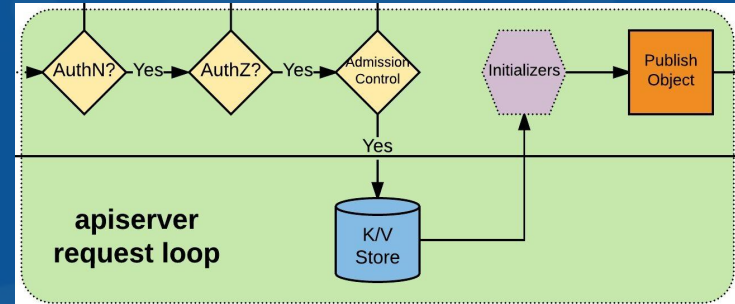
Authorization iterates over available AuthZ sources: Node, ABAC, RBAC, or webhook.

AdmissionControl checks resource quotas, other security related checks etc.

Request is stored in etcd.

Initializers are given opportunity to mutate request before the object is published.

Request is published on apiserver.



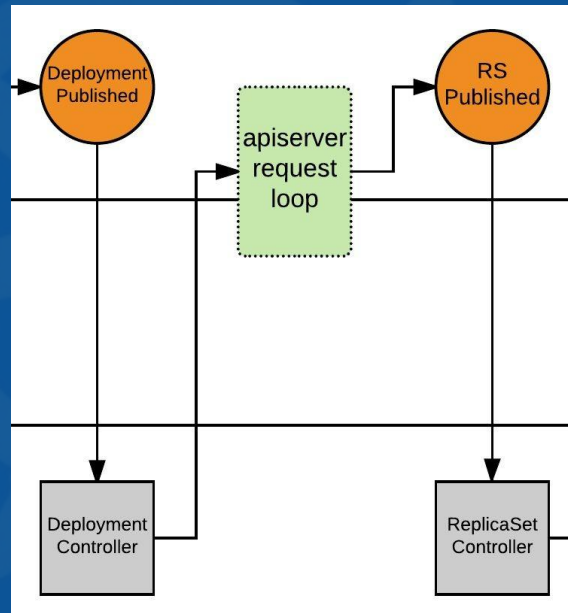
Deployment Controller

Deployment Controller is notified of the new Deployment via callback.

Deployment Controller evaluates cluster state and reconciles the desired vs current state and forms a request for the new ReplicaSet.

apiserver request loop evaluates Deployment Controller request.

ReplicaSet is published.



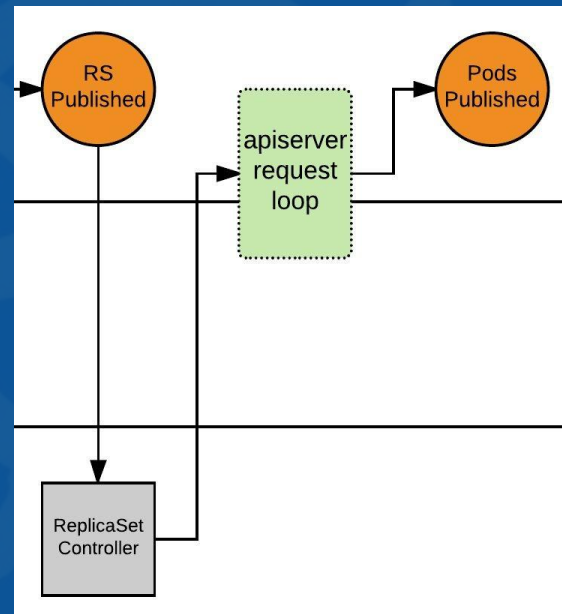
ReplicaSet Controller

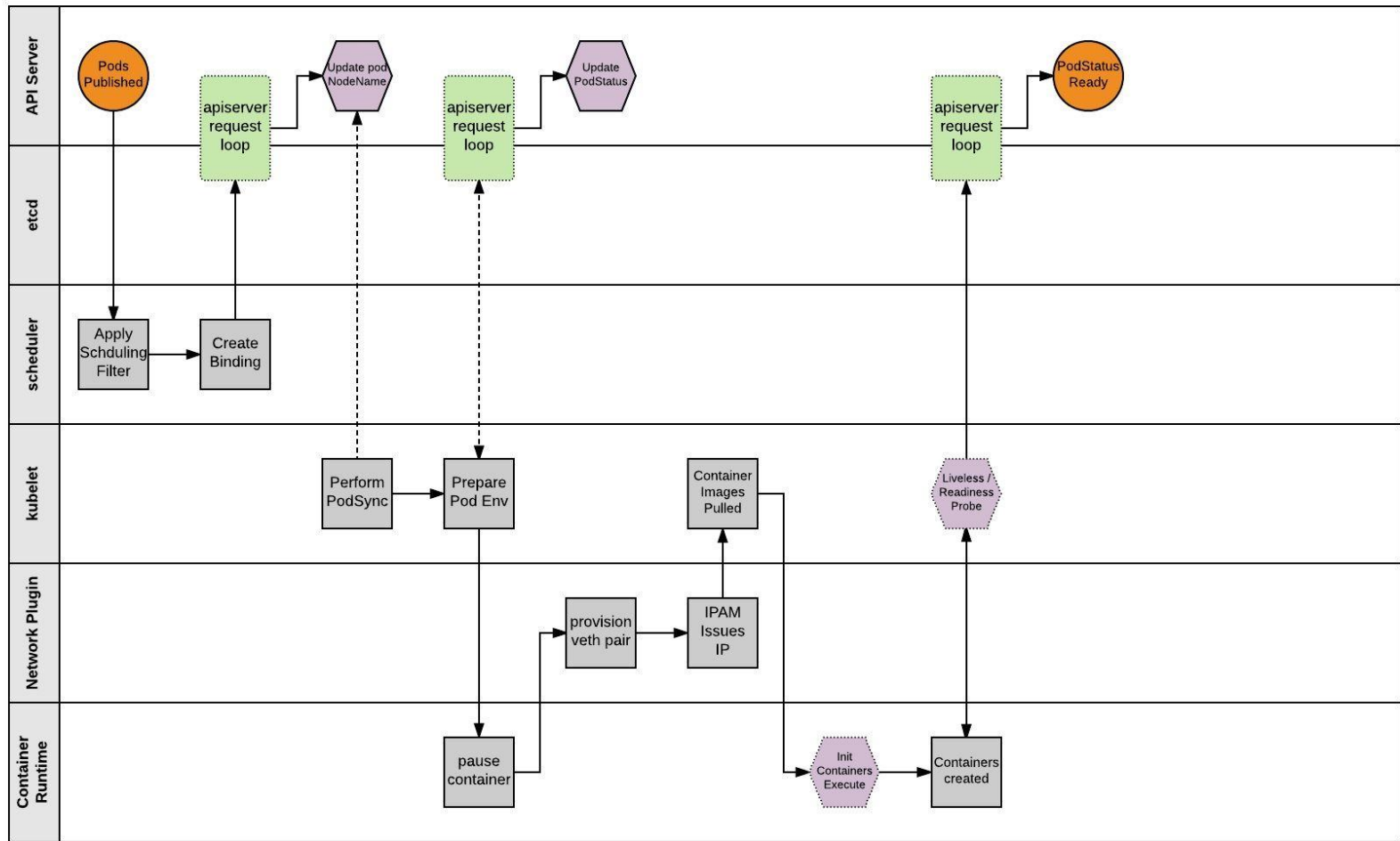
ReplicaSet Controller is notified of the new ReplicaSet via callback.

ReplicaSet Controller evaluates cluster state and reconciles the desired vs current state and forms a request for the desired amount of pods.

apiserver request loop evaluates ReplicaSet Controller request.

Pods published, and enter 'Pending' phase.





Scheduler

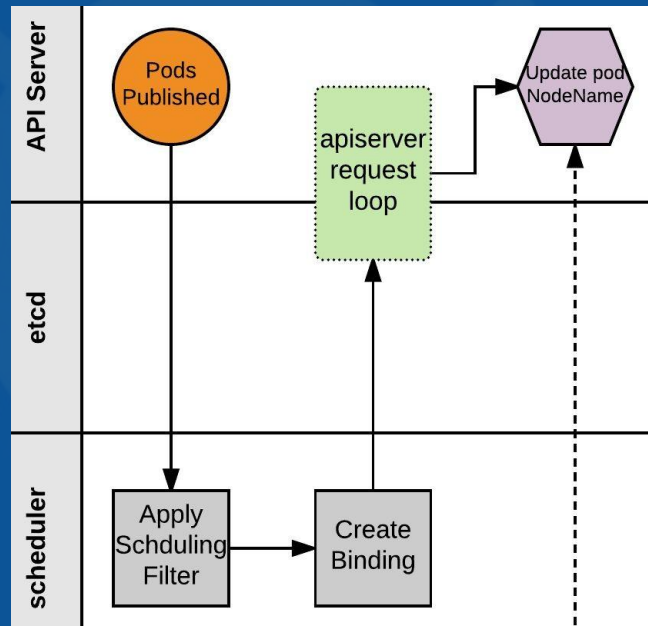
Scheduler monitors published pods with no 'nodeName' assigned.

Applies scheduling rules and filters to find a suitable node to host the Pod.

Scheduler creates a binding of Pod to Node and POSTs to apiserver.

apiserver request loop evaluates POST request.

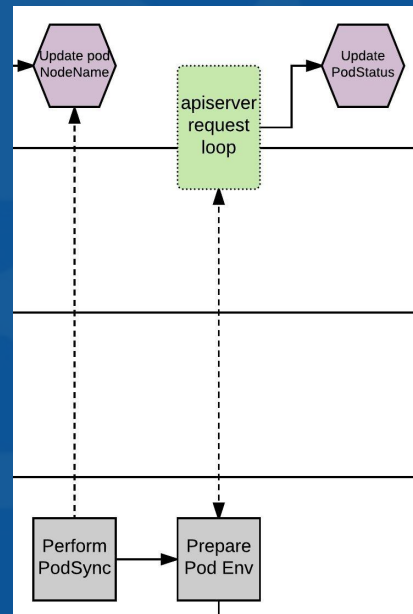
Pod status is updated with node binding and sets status to 'PodScheduled'.



Kubelet - PodSync

The kubelet daemon on every node polls the apiserver filtering for pods matching its own 'nodeName'; checking its current state with the desired state published through the apiserver.

Kubelet will then move through a series of internal processes to prepare the pod environment. This includes pulling secrets, provisioning storage, applying AppArmor profiles and other various scaffolding. During this period, it will asynchronously be POST'ing the 'PodStatus' to the apiserver through the standard apiserver request loop.

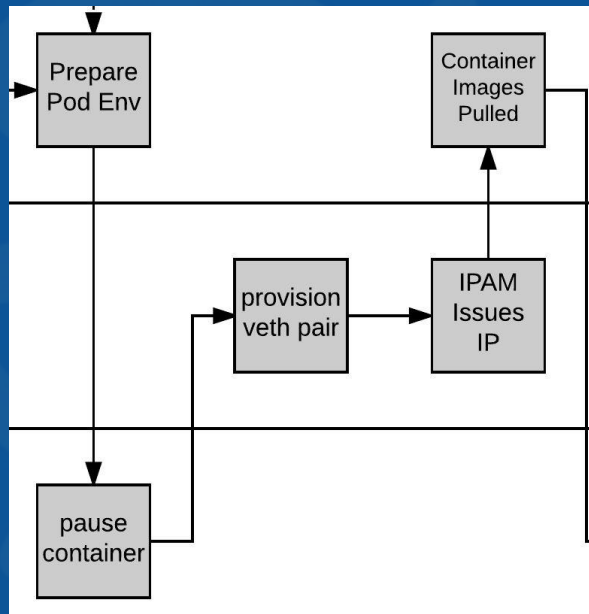


Pause and Plumbing

Kubelet then provisions a 'pause' container via the CRI (Container Runtime Interface). The pause container acts as the parent container for the Pod.

The network is plumbed to the Pod via the CNI (Container Network Interface), creating a veth pair attached to the pause container and to a container bridge (cbr0).

IPAM handled by the CNI plugin assigns an IP to the pause container.

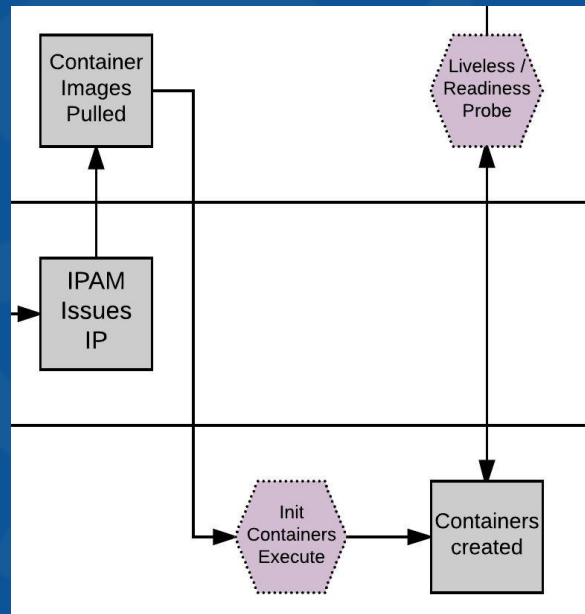


Kubelet - Create Containers

Kubelet pulls the container Images.

Kubelet first creates and starts any init containers.

Once the optional init containers complete, the primary pod containers are started.

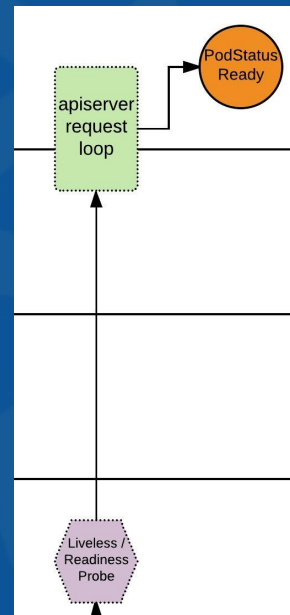


Pod Status

If there are any liveless/readiness probes, these are executed before the PodStatus is updated.

If all complete successfully, PodStatus is set to ready and the container has started successfully.

The Pod is Deployed!





Demo

