Chapter 3
Container Orchestration with Kubernetes

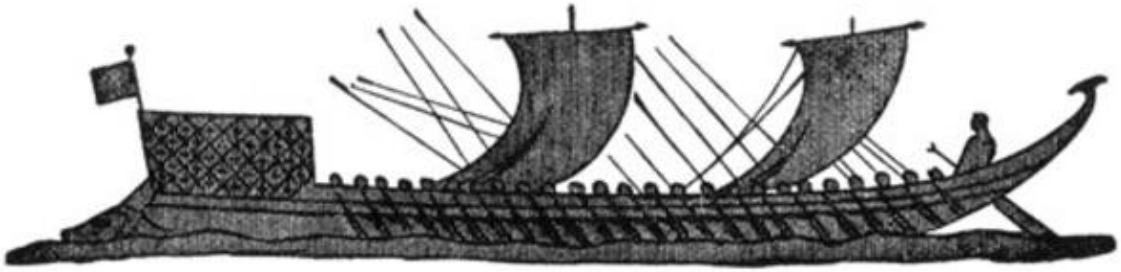
Learning Topics

- Overview of Kubernetes
- Kubernetes Architecture
- Key Components
- Key Terminologies
- Deployment & Load Balancing
- High Availability
- Horizontal Pod AutoScaler

Kubernetes Meaning

Greek for “pilot” or

“Helmsman of a ship”



K8s History

- Project that was spun out of Google as an open source container orchestration platform.
- Built from the lessons learned in the experiences of developing and running Google's Borg and Omega.
- Designed from the ground-up as a **loosely coupled** collection of components centered around deploying, maintaining and scaling workloads.

K8s History

- Contributors include Google, CodeOS, Redhat, Mesosphere, Microsoft, HP, IBM, VMWare, Pivotal, SaltStack etc.

- Kubernetes is loosely coupled, meaning that all the components have little knowledge of each other and function independently.
  - This makes them easy to replace and integrate with a wide variety of systems

- Written in Go Language

Who Manages Kubernetes

- Sub-Foundation of Linux Foundation
- A Vendor Neutral Entity to Manage “Cloud Native” Projects
- Focused on
  - Containers
  - Dynamic Orchestration
  - Many More Services

Container Issues

Scheduling: Where should my containers run?

Lifecycle and health: Keep my containers running despite failures

Discovery: Where are my containers now?

Monitoring: What's happening with my containers?

Auth{n,z}: Control who can do things to my containers

Aggregates: Compose sets of containers into jobs

Scaling: Making jobs bigger or smaller

What Does Kubernetes Do?

- Groups containers that make up an application into logical units for easy management and discovery
- It acts as an engine for resolving state by converging actual and the **desired state** of the system
- It is declarative, you tell it what you want it to be, and it figures it out
  - e.g. ‘I want 3 instances of x’ and it just does it, if something dies, it brings it back to get to 3

Most Popular Use Cases

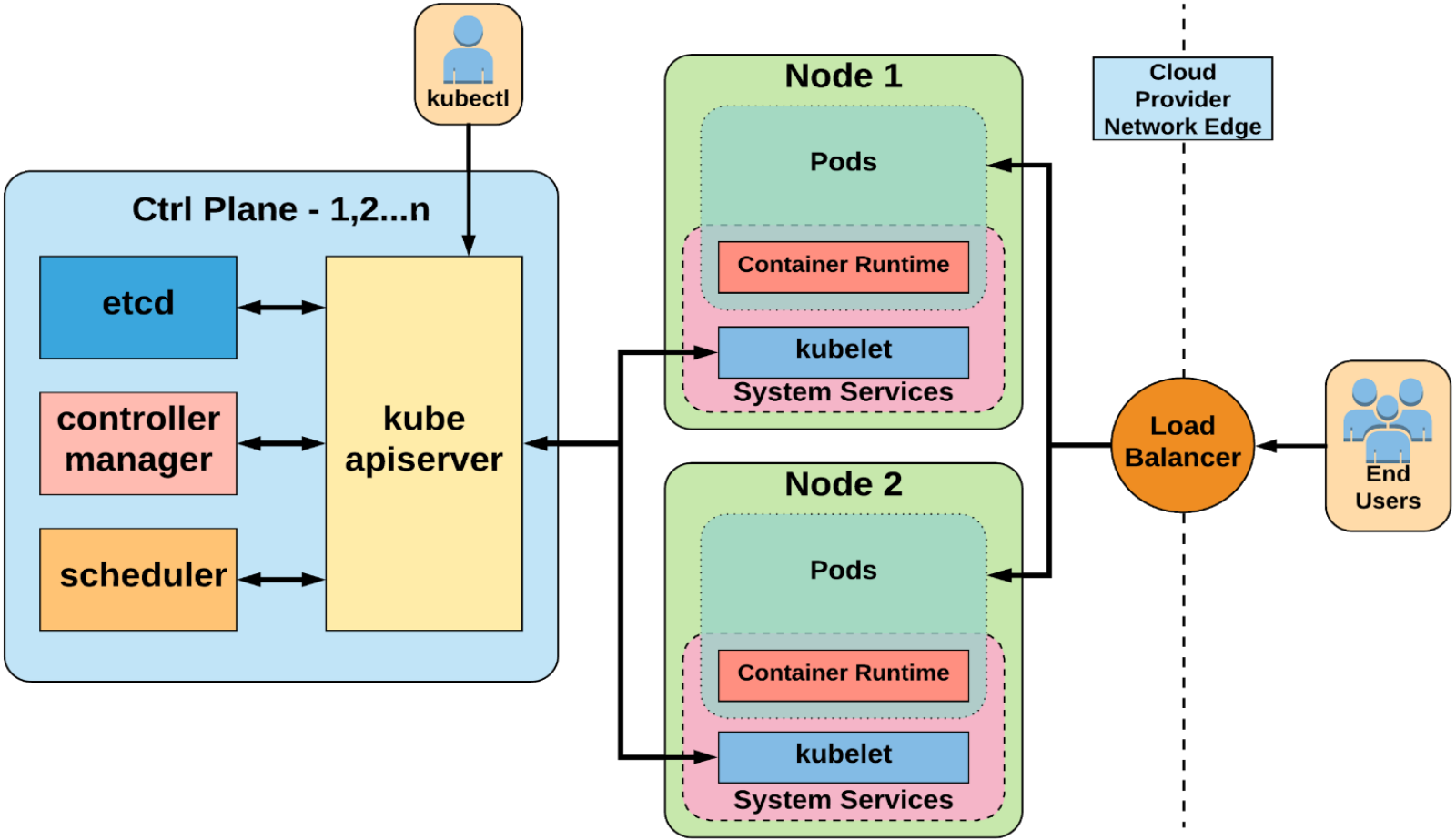
- Autoscale Workloads
- Blue/Green Deployments
- Scheduled Cronjobs
- Manage Stateless and Stateful Applications
- Easily Integrate and Support 3rd Party Apps
- Provide Native Methods of Service Discovery

Kubernetes Features

- Kubernetes is an open-source system for
  - Automating Deployment
  - Scaling
  - Management

  of containerized applications
- Kubernetes can scale without increasing your ops team

Kubernetes Architecture



Key Terminologies

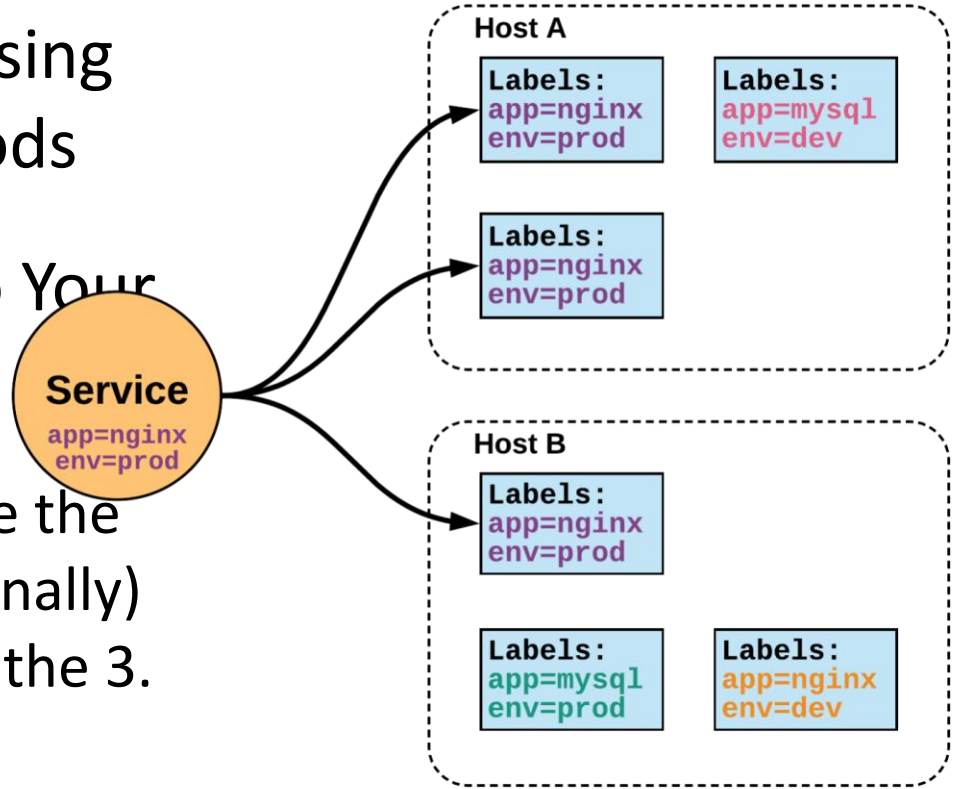
- **Pod** - A group of Containers
- **Labels** - Labels for identifying pods
- **Kubelet** - Container Agent
- **Proxy** - A load balancer for Pods
- **etcd** - A metadata service
- **cAdvisor** - Container Advisor provides resource usage/performance statistics
- **Replication Controller** - Manages replication of pods
- **Scheduler** - Schedules pods in worker nodes
- **API Server** - Kubernetes API server

Pods

- Smallest Unit of Work
- Collection of One or More Containers
- Share Volumes, Network Namespace
- Part of Single Context, Managed Together
- Ephemeral in Nature

Services (Proxy)

- Unified Method of Accessing Exposed Workloads of Pods
- Internal Load Balancer to Your Pod(s)
  - Create a service, reference the pods, for e.g. 3, and (internally) it will load balance across the 3.
- Durable Resource

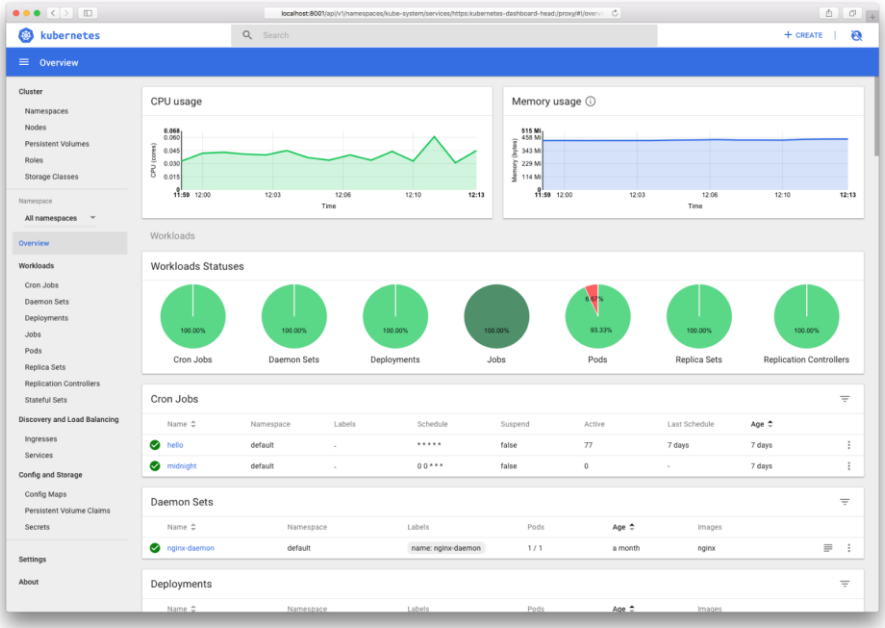


Cluster Overview for Labs

- Elastic Load Balancer (Cloud Provided)
- 3 Masters (Facilitates HA)
- 50 Nodes

Lab 1 – Deploy Stateless App on K8 Cluster

- Use Kubernetes Dashboard



Control Plane Components

- **Kube-apiserver**
  - Gate keeper for everything in kubernetes
  - EVERYTHING interacts with kubernetes through the apiserver
- **Etcd**
  - Distributed storage back end for kubernetes
  - The apiserver is the only thing that talks to it
- **Kube-controller-manager**
  - The home of the core controllers
- **Kube-scheduler**
  - Handles placement

Kube-apiserver

- Provides a forward facing REST interface into the kubernetes control plane and datastore.
- All clients and other applications interact with kubernetes **strictly** through the API Server.
- Handles authn, authz, request validation, mutation and admission control and serves as a generic front end to the backing datastore

etcd

- etcd acts as the cluster datastore.
- Purpose in relation to Kubernetes is to provide a strong, consistent, highly durable and highly available key-value store for persisting cluster state.
- Stores objects and config information.

Kube-controller-manager

- Its the director behind the scenes
- Serves as the primary daemon that manages all core component control loops.
- Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**.
- Does NOT handle scheduling, just decides what the desired state of the cluster should look like
  - e.g. receives request for a deployment, produces replicaset, then produces pods

Kube-scheduler

- Scheduler decides which nodes should run which pods
- Serves as the primary daemon that manages all core component control loops.
- Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**.
- Does NOT handle scheduling, just decides what the desired state of the cluster should look like
  - e.g. receives request for a deployment, produces replicaset, then produces pods

Lab 2 – Scale Running Pods

- Use Kubernetes Dashboard

Node Components

- **Kubelet**
  - Agent running on every node, including the control plane
- **Kube-proxy**
  - The network ‘plumber’ for Kubernetes services
  - Enables in-cluster load-balancing and service discovery
- **Container Runtime Engine**
  - The containerizer itself - typically docker

Kubelet

- Acts as the node agent responsible for managing the lifecycle of every pod on its host.
- Kubelet understands YAML container manifests that it can read from several sources:
  - file path
  - HTTP Endpoint
  - etcd watch acting on any changes
  - HTTP Server mode accepting container manifests over a simple API.
  - The single host daemon required for a being a part of a kubernetes cluster

Kube-proxy

- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.
- Creates the rules on the host to map and expose services
- Available Proxy Modes:
  - Userspace
  - iptables
  - ipvs (default if supported)

Container Runtime Engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)

- Kubernetes functions with multiple different containerizers

- Interacts with them through the CRI - container runtime interface

- CRI creates a ‘shim’ to talk between kubelet and the container runtime

Lab 3 – Deploy & Expose App

- Use Command Line

Kubernetes Lab – Deploy & Expose App

- Use nginx as Base Image
- Modfiy Index.html (Add Your Name to Text)
- Use Dockerfile to Create New Image
- Save Image to Docker Hub
- Deploy Container Using This Image on K8 Cluster
  - **Use Dashboard or CLI for Deployment**

Object Model

- Objects are a "*record of intent*" or a persistent entity that represent the desired state of the object within the cluster.
- All objects **MUST** have apiVersion, kind, and poses the nested fields metadata.name, metadata.namespace, and metadata.uid.
- This establishes the type of object, the version of the object and a unique identity for the object itself

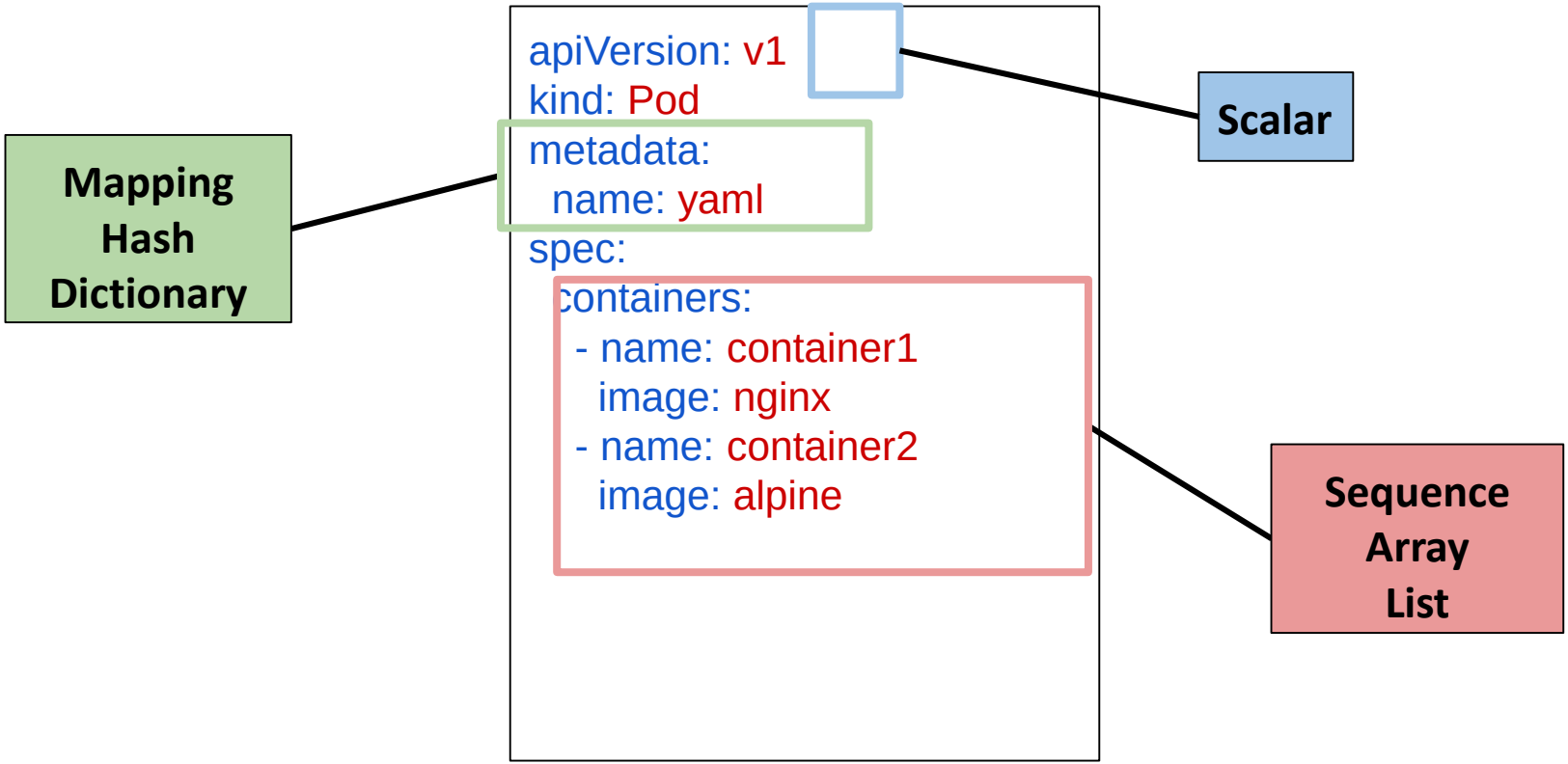
Object Model Requirements

- apiVersion: Kubernetes API version of the Object
- kind: Type of Kubernetes Object
- metadata.name: Unique name of the Object
- metadata.namespace: Scoped environment name that the object belongs to (will default to current).
- metadata.uid: The (generated) uid for an object.

Object Model YAML

- Files or other representations of Kubernetes Objects are generally represented in YAML.
- A “*Human Friendly*” data serialization standard.
- Uses white space (specifically spaces) alignment to denote ownership.
- Three basic data types:
    - **mappings** - hash or dictionary,
    - **sequences** - array or list
    - **scalars** - string, number, boolean etc

Object Model YAML



Object Model Workloads

- Workload related objects within Kubernetes have an additional two nested fields spec and status.
  - spec - Describes the **desired state** or **configuration** of the object to be created.
  - status - Is managed by Kubernetes and describes the **actual state** of the object and its history.

Lab 4 – Deploy Stateless Application with Deployment Objects

- Create Kubernetes Deployment Object
- Describe Deployment in YAML file
  - Define Replicas
  - Define Images
  - Define Containers
- Run Apps Using Deployment Objects

Lab 5 – Rolling Updates

- Deployment Stays Same
- YAML Changes
- Replace One Pod At a Time
- Update Image

Kubernetes Networking

- **Pod Network**
  - Cluster-wide network used for pod-to-pod communication managed by a CNI (***Container Network Interface***) plugin.
- **Service Network**
  - Cluster-wide range of **Virtual IPs** managed by **kube-proxy** for service discovery.

Kubernetes Networking

- Unlike Docker, every pod gets its own cluster wide unique IP, and makes use of the CNI plugin

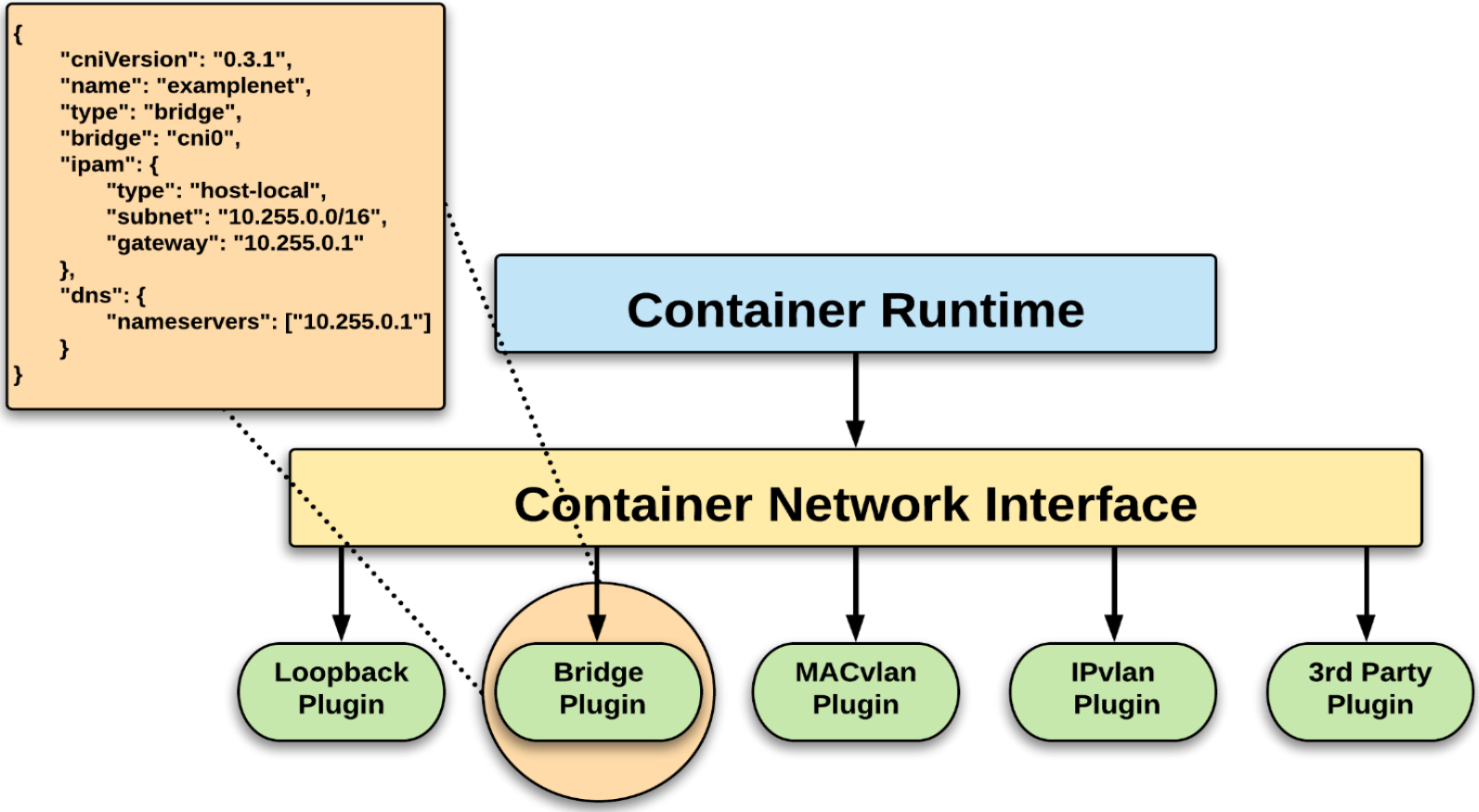
- Services are a separate range of static non-routable virtual IPs that are used like an internal LB or static IP

- Service IPs are special and can’t be treated like a normal IP, they are a ‘mapping’ stored and managed by kube-proxy
  - Pod IPs are pingable
  - Service IPs are not

Container Network Interface (CNI)

- Pod networking within Kubernetes is plumbed via the Container Network Interface (CNI).
- Functions as an interface between the container runtime and a **network implementation plugin**.
- CNCF Project
- Uses a simple JSON Schema.
- CNI runtime focuses solely on container lifecycle connectivity

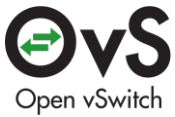
CNI Overview



CNI Plugins

- Amazon ECS
- Calico
- Cillium
- Contiv
- Contrail
- Flannel

- GCE
- kube-router
- Multus
- OpenVSwitch
- Romana
- Weave



Fundamental Networking Rules

- All containers within a pod can communicate with each other unimpeded.
- All Pods can communicate with all other Pods without NAT.
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as is the same IP that others see it as.
- Pods are given a cluster unique IP for the duration of its lifecycle.

Fundamental Networking Rules - II

- **Container-to-Container**
  - Containers within a pod exist within the **same network namespace** and share an IP.
  - Enables intrapod communication over *localhost*.
- **Pod-to-Pod**
  - Allocated **cluster unique IP** for the duration of its life cycle.
  - Pods themselves are fundamentally ephemeral.

Fundamental Networking Rules - III

- **Pod-to-Service**
  - managed by **kube-proxy** and given a **persistent cluster unique IP**
  - exists beyond a Pod's lifecycle
  - kubernetes creates a cluster-wide IP that can map to n-number of pods
- **External-to-Service**
  - Handled by **kube-proxy.**
  - Works in cooperation with a cloud provider or other external entity (load balancer).

This concludes Chapter 3

Container Orchestration with Kubernetes