3.

Orchestration and Management

Recaps

Definitions

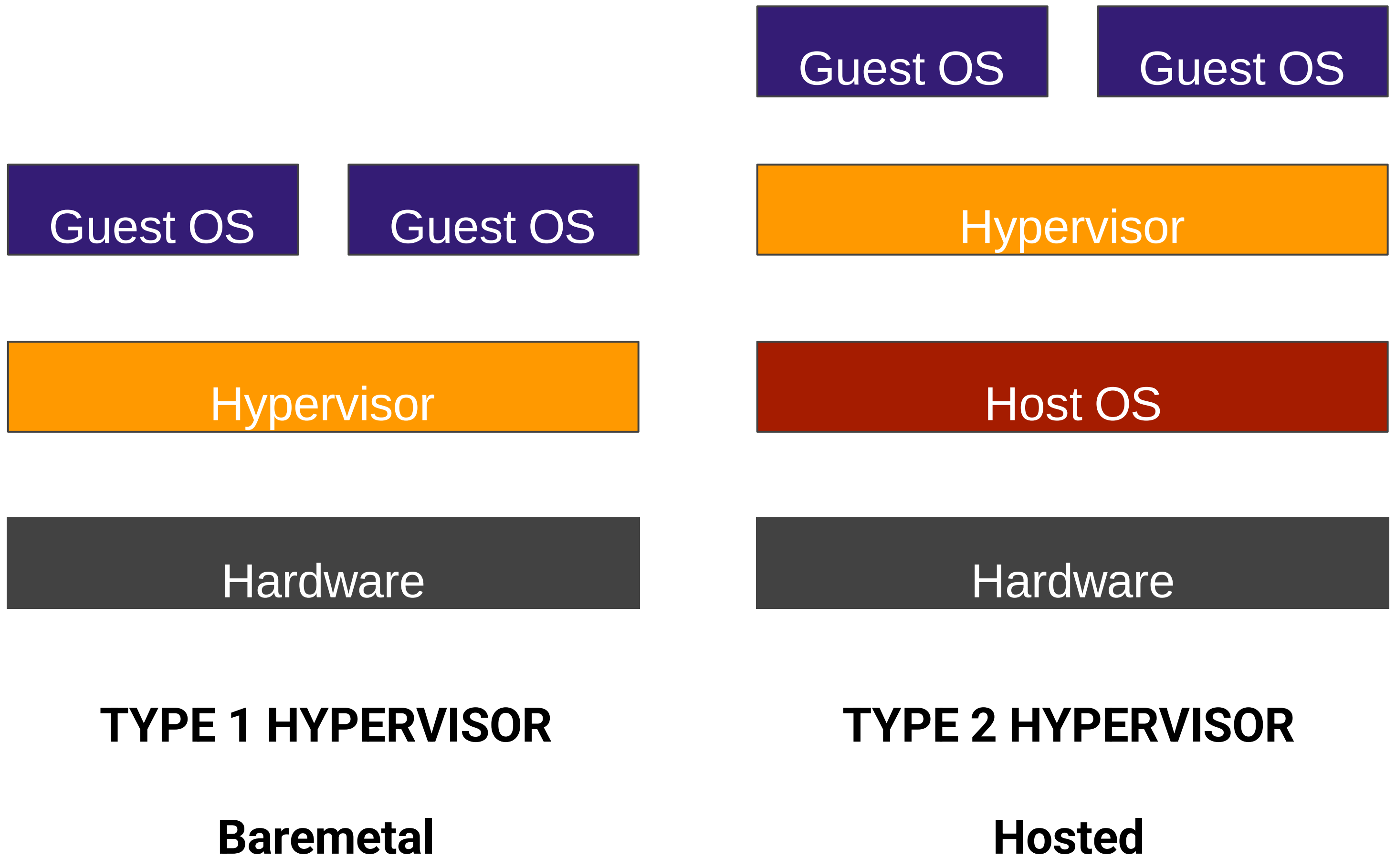
Virtualization

- Virtualization is the abstraction of the true underlying hardware or software from VM Operating System (OS), application, storage or network

- Since we cannot run two operating systems on the same hardware at the same time (OSs like to have the full control !), virtualization makes an OS believe it’s running on its own physical machine and has the full control of it

  VM OS is called **Guest OS**
  Physical machine OS is called **Host OS**

How it works ?



Why call it a “Hypervisor”?

Initially, it was about resource allocation, and the goal was to try to utilize areas of memory that were not normally accessible to programmers. The code produced was successful and was dubbed a hypervisor because, at the time, operating systems were called supervisors, and this code could supersede them.

What are containers?

- Containers offer a **logical packaging mechanism** in which **applications can be abstracted from the environment in which they actually run**. This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is a private data center, the public cloud, or even a developer's personal laptop

- Containerization **provides a clean separation of concerns**, as **developers focus on their application logic and dependencies**, while **IT operations teams can focus on deployment and management** without bothering with application details such as specific software versions and configurations specific to the app

Monolithic vs Microservice Architecture

Monolithic Architecture:

A single, unified application where all components are interconnected and run as a single service.

Benefits:

•Easier to develop and deploy initially.

Disadvantages:

•All components share the same codebase.

•Scaling requires scaling the entire application.

•Changes in one part can affect the whole system.

Monolithic vs Microservice Architecture

Microservices Architecture:

An architectural style that structures an application as a collection of loosely coupled services, each responsible for a specific function.

Benefits:

- Each service can be developed, deployed, and scaled independently.
- Technologies and languages can vary between services.
- Better fault isolation; issues in one service do not necessarily affect others.

Disadvantages:

- More complex to manage due to inter-service communication

What is orchestration?

Network Orchestration refers to the automated management and coordination of network resources and services. It involves:

- **Integration**: Combining various network components and services into a cohesive system.
- **Automation**: Streamlining processes to reduce manual intervention and improve efficiency.
- **Policy Management**: Implementing rules and policies to optimize network performance and security.
- **Resource Allocation**: Dynamically assigning resources based on demand and performance metrics.

Why do we need Orchestration ?

- Automation
- Networking
- Management
- Re/Configuration

Docker compose

Docker Compose is a tool for defining and running multi-container Docker applications using a simple YAML file.

Key Features:

- Multi-Container Management: Define multiple services, networks, and volumes in a single file.
- Simplified Configuration: Use a docker-compose.yml file to configure your application's services.
- Single Command Deployment: Start all services with a single command (docker-compose up).

Docker Compose: For defining and running multi-container applications locally.

Docker swarm

Docker Swarm is a native clustering and orchestration tool for Docker. It allows you to manage a cluster of Docker engines (nodes) as a single virtual system.

Key Features:

- Clustering: Combines multiple Docker hosts into a single virtual host.
- Load Balancing: Distributes incoming requests across multiple containers.
- Scaling: Easily scale services up or down.
- High Availability: Ensures that services are running and can recover from failures.

Docker Swarm: For orchestrating and managing clusters of Docker containers in production.

3.
IaaS for Telcos: A focus on OpenStack

What is OpenStack ?

- OpenStack is a **cloud operating system** that controls pools of compute, storage, and networking resources throughout a datacenter, all **managed and provisioned through APIs** with common authentication mechanisms
- First release to the open source in **2010** after NASA and Rackspace co-development
- **OpenStack Foundation** was launched in **2012** to promote the software and its community
- Beyond standard **Infrastructure-as-a-service** functionality, additional components provide **orchestration**, **fault management** and **service management** amongst other services to ensure high availability of user applications
- Written mostly in the **Python** programming language
- OpenStack release code names are cities or counties near where the corresponding OpenStack design summit took place

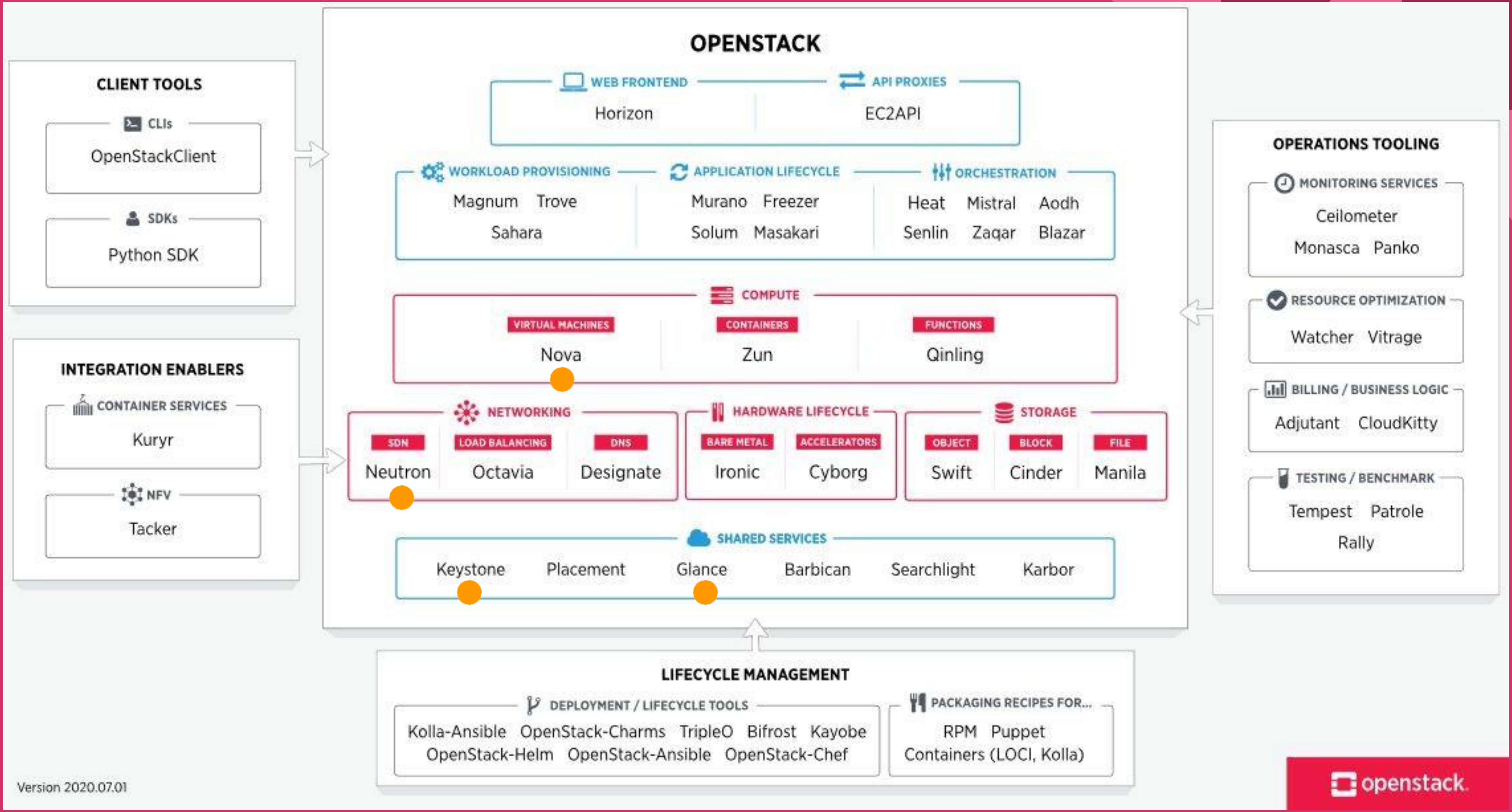
What is OpenStack ?

- **Xena**: Named after **Xena**, a fictional character, but often associated with the city of **Xena** in Greece.
  Release Date: October 2021
- **Wallaby**: Named after the **Wallaby**, a marsupial native to Australia.
  Release Date: April 2021
- **Victoria**: Named after **Victoria**, the capital city of British Columbia, Canada.
  Release Date: October 2020
- **Ussuri**: Named after the **Ussuri River** in Russia.
  Release Date: April 2020

What is OpenStack ?

- OpenStack is developed and released around 6-month cycles
- Current release is zed (named after the city of **Zed** in the fictional universe of "The Matrix")

2. IaaS for Telcos: A focus on OpenStack



OpenStack Core Services [2]

Keystone: Identity service

- Provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. It supports LDAP, OAuth, OpenID Connect, SAML and SQL

- First appeared in OpenStack 'Essex' release

OpenStack Core Services [2]

Glance: Image service

- Allows discovering, registering, and retrieving virtual machine images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project
- Supported disk formats: raw, vhd, vhdx, vmdk, vdi, iso, qcow2, etc.

- Depends on: Keystone
- First appeared in OpenStack 'Bexar' release

OpenStack Core Services [2]

Nova: Compute Service

- To implement services and associated libraries to provide massively scalable, on demand, self service access to compute resources, including bare metal, virtual machines, and containers

- Depends on: Keystone, Neutron, Glance
- First appeared in OpenStack 'Austin' release

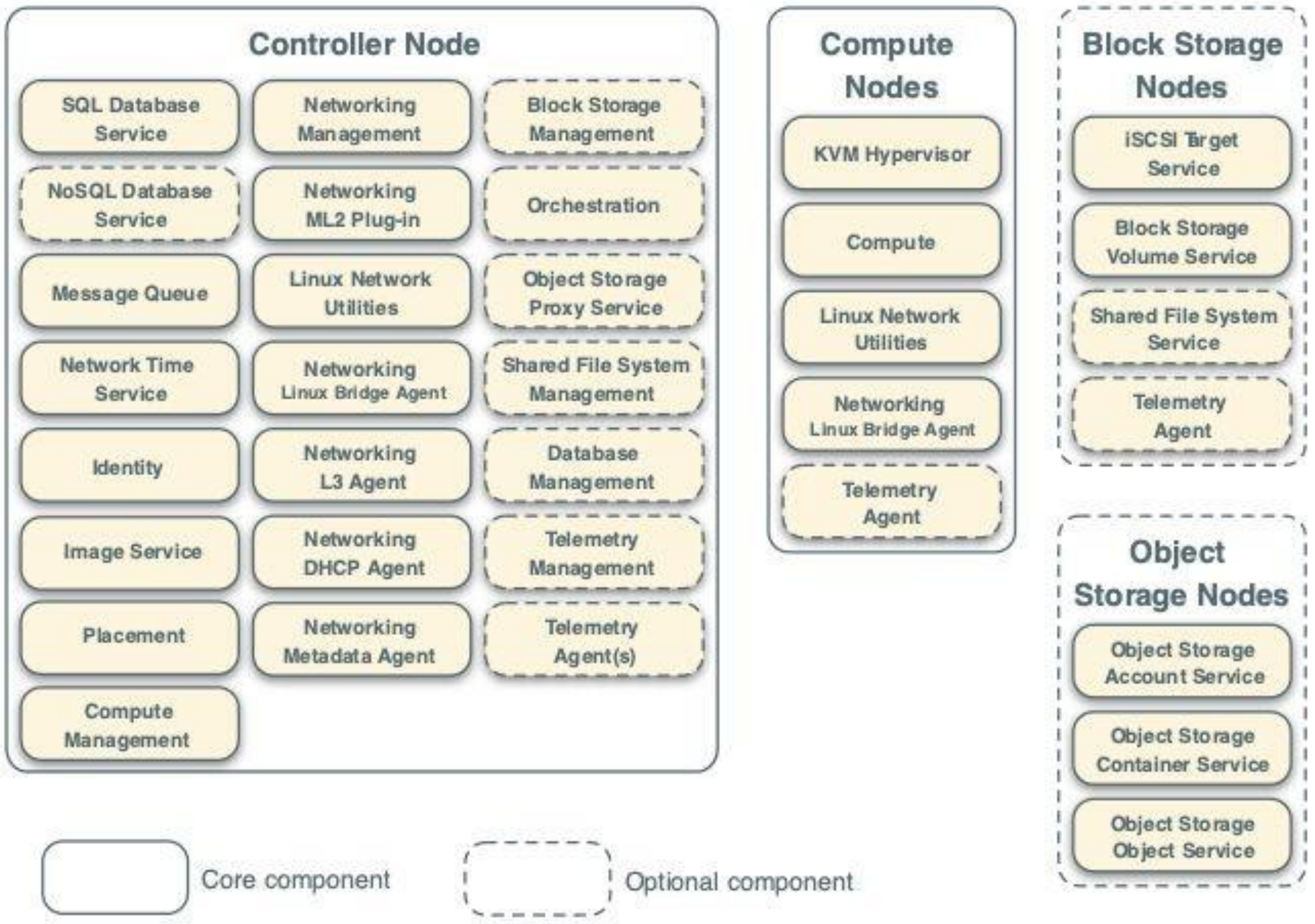
OpenStack Core Services [2]

Neutron: Networking Service

- Neutron is an SDN networking project focused on delivering Networking-as-a-Service (NaaS) in virtual compute environments

- Depends on: Keystone
- First appeared in OpenStack 'Folsom' release

OpenStack Deployment Options [2]



1. **Control Nodes**
   - **Function**: Manage and orchestrate the OpenStack services.
   - **Components**: Typically run services like **Keystone** (identity), **Glance** (image service), **Nova** (compute), **Neutron** (networking), and **Horizon** (dashboard).
   - **Role**: Centralized management and coordination of the cloud environment.
2. **Compute Nodes**
   - **Function**: Provide the actual compute resources for running virtual machines (VMs).
   - **Components**: Run the **Nova** compute service, which manages the lifecycle of VMs.
   - **Role**: Host and execute workloads.
3. **Block Storage Nodes**
   - **Function**: Provide persistent block storage to instances.
   - **Components**: Run the **Cinder** service, which manages block storage volumes.
   - **Role**: Allow VMs to attach and use block storage devices.
4. **Object Storage Nodes**
   - **Function**: Provide scalable object storage for unstructured data.
   - **Components**: Run the Swift service, which manages object storage.
   - **Role**: Store and retrieve large amounts of data, such as backups and media files.

→ **Block Storage**

•**Type of Data**: Structured data, typically used for applications that require low-latency access.

•**Use Cases**:

- Virtual machine disks (e.g., boot volumes)
- Databases
- File systems

•**Characteristics**:

- Data is stored in fixed-size blocks.
- Allows for random access, making it suitable for applications that require high performance.

→ **Object Storage**

•**Type of Data**: Unstructured data, such as files, images, videos, and backups.

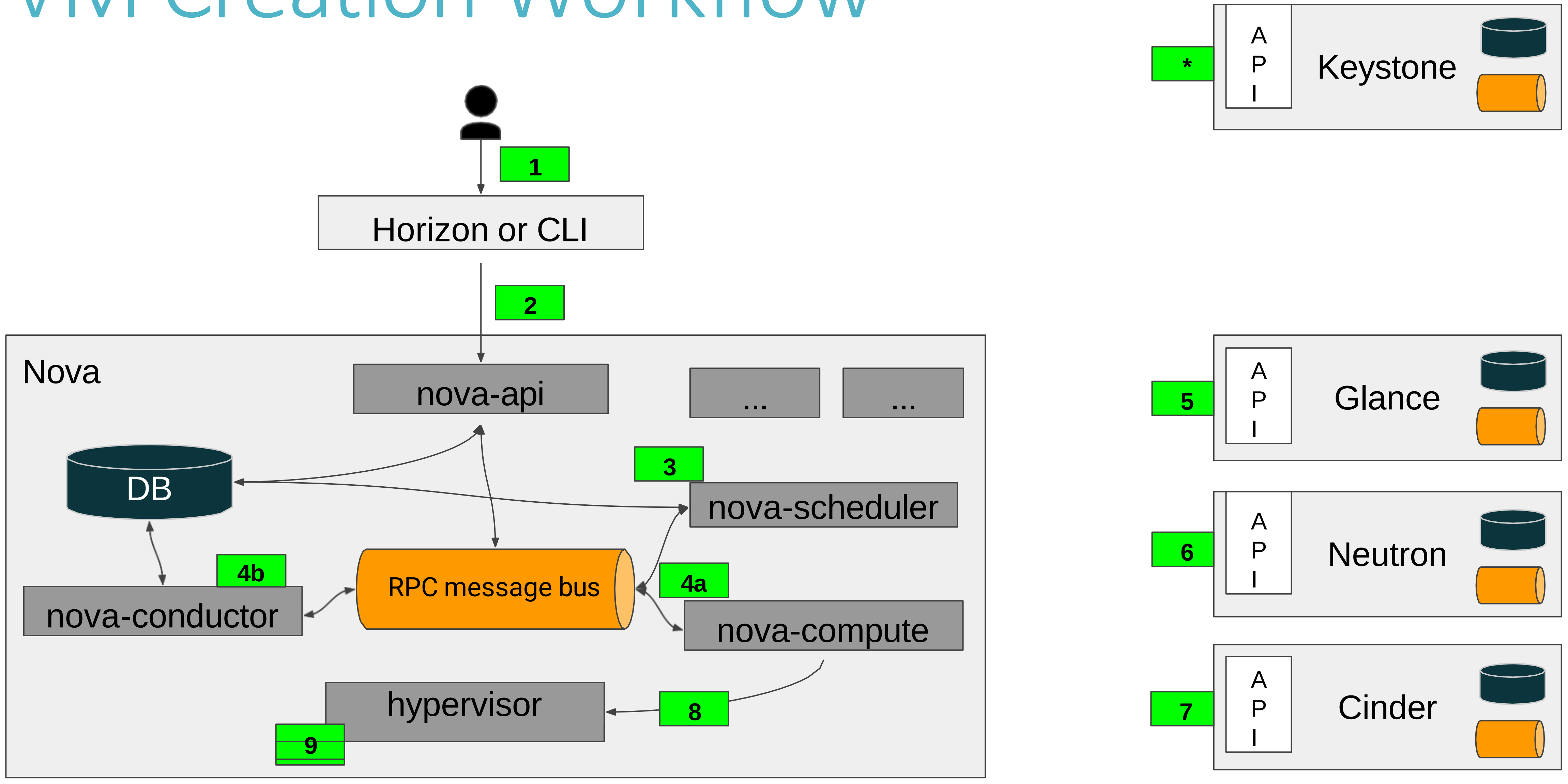
•**Use Cases**:

- Media storage
- Backup and archival
- Big data analytics

•**Characteristics**:

- Data is stored as objects with metadata.
- Designed for scalability and durability, suitable for large amounts of data.

VM Creation Workflow



VM Creation Workflow

- **Step 1**: The user makes his/her request through Horizon or CLI, which in turns, translate it into a REST API call to OpenStack Nova (nova-api application) Service API
- **Step 2**: nova-api will request the nova-scheduler through the communication bus to schedule the VM to a given host.

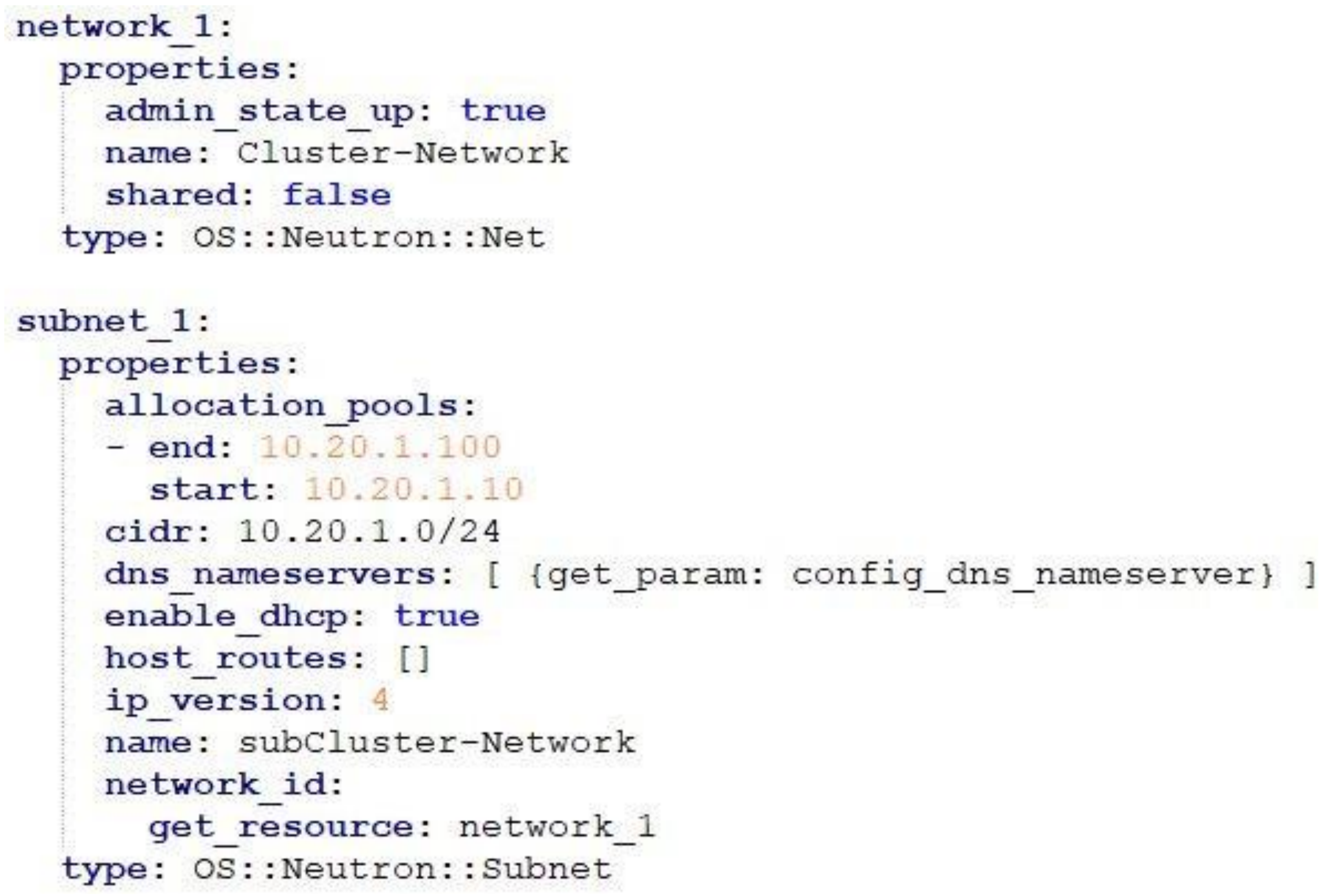
  The result of this step is a host ID (or an error in case the VM can't be deployed for some reason)
- **Step 3**: nova-scheduler look into the database for an available host, taking into account VM's characteristics.

  If found, it requests nova-compute to create and boot the VM
- **Step 4a/b**: nova-compute requests VMs information from nova-conductor (acting as a DB interface)
- **Step 5**: nova-compute requests the VM image URI from the glance-api and loads it from storage
- **Step 6**: nova-compute requests networking configuration (port, IP@) from neutron-server
- **Step 7**: nova-compute requests the creation a VM's block storage volume to cinder-api
- **Step 8**: nova-compute requests creating and booting the VM to the Compute Node hypervisor
- **Step 9**: VM is up and running

* each service validates auth-token and access permission using Keystone for each API request

OpenStack Orchestration Service: Heat [2]

- Heat **orchestrates the infrastructure resources** for a cloud application based on templates in the form of **YAML files** that can be treated like code (i.e. versioned) a.k.a. Heat Orchestration Templates (HOT)
- It provides both an OpenStack-native REST API and a CloudFormation-compatible (AWS) Query API
- It also provides an **auto-scaling** service that integrates with the OpenStack Telemetry services

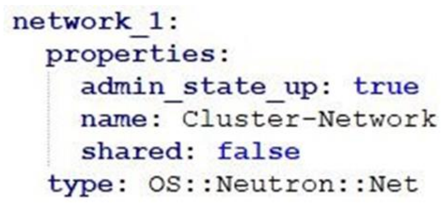


```
network_1:
  properties:
    admin_state_up: true
    name: Cluster-Network
    shared: false
  type: OS::Neutron::Net

subnet_1:
  properties:
    allocation_pools:
    - end: 10.20.1.100
      start: 10.20.1.10
    cidr: 10.20.1.0/24
    dns_nameservers: [ {get_param: config_dns_nameserver} ]
    enable_dhcp: true
    host_routes: []
    ip_version: 4
    name: subCluster-Network
    network_id:
      get_resource: network_1
  type: OS::Neutron::Subnet
```

OpenStack Orchestration Service: Heat [2]

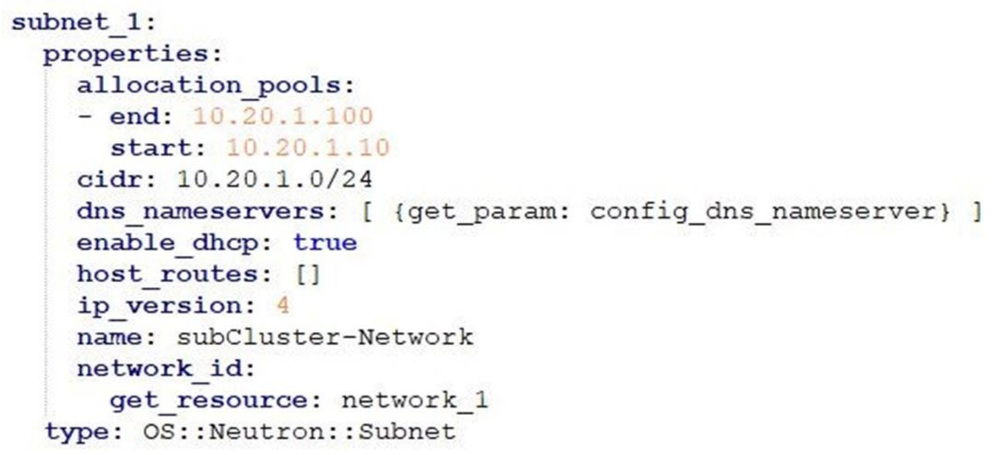
- **network_1**: This is the logical name of the network resource being defined.
- **properties**: Contains the configuration settings for the network.
    - **admin_state_up**: Set to true, indicating that the network is active and operational.
    - **name**: The name of the network, here it is set to "Cluster-Network".
    - **shared**: Set to false, meaning this network is not shared across different projects (tenants).
    - **type**: Specifies the resource type as OS::Neutron::Net, indicating that this resource is a Neutron network.



```
network_1:
  properties:
    admin_state_up: true
    name: Cluster-Network
    shared: false
  type: OS::Neutron::Net
```

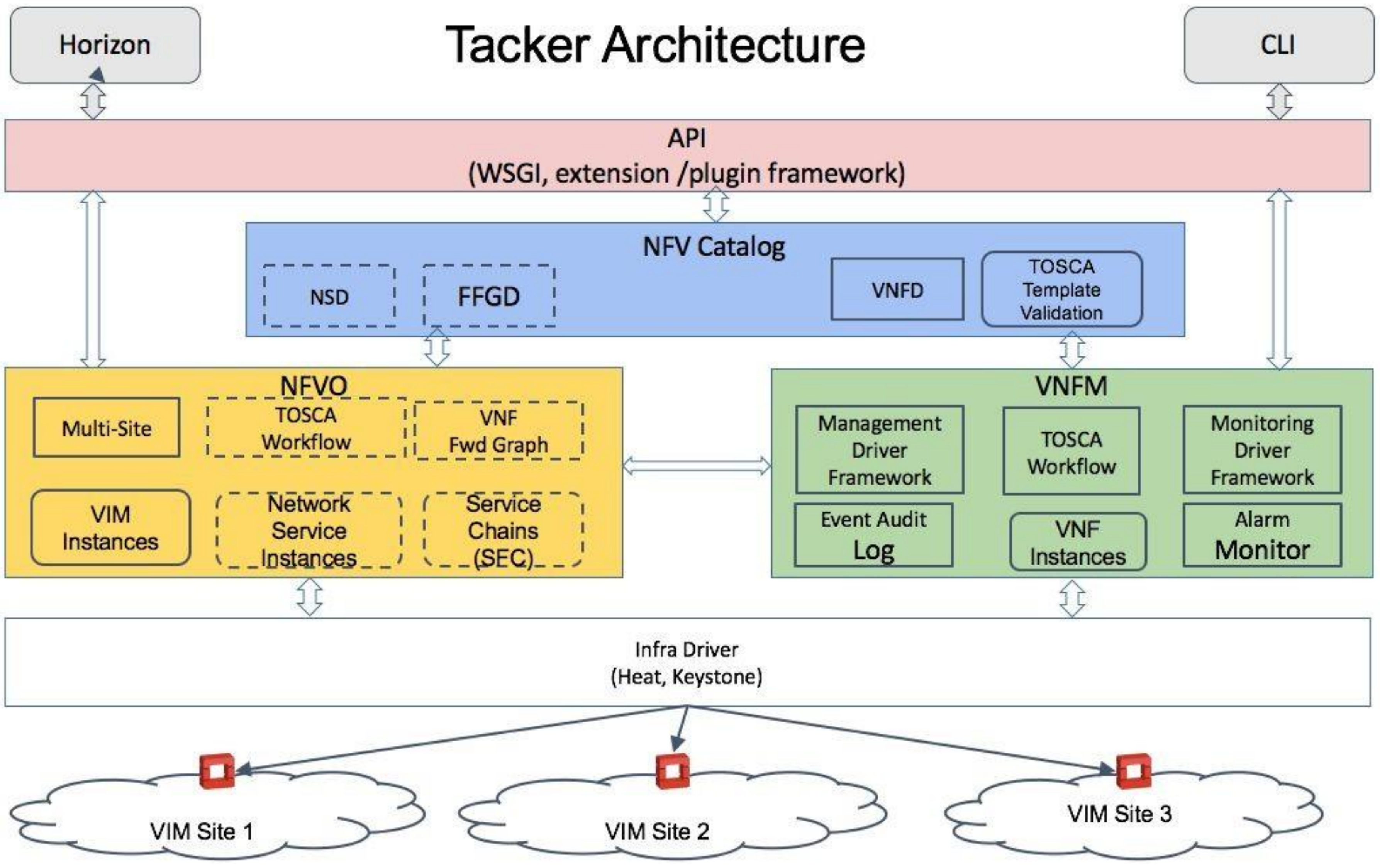
OpenStack Orchestration Service: Heat [2]

- **subnet_1**: This is the logical name of the subnet resource being defined.
- **properties**: Contains the configuration settings for the subnet.
  - **allocation_pools**: Defines the range of IP addresses that can be allocated to instances within this subnet.
    - **end**: The last IP address in the allocation pool (10.20.1.100).
    - **start**: The first IP address in the allocation pool (10.20.1.10).
  - **cidr**: The Classless Inter-Domain Routing (CIDR) notation for the subnet (10.20.1.0/24), indicating a subnet mask of 255.255.255.0.
  - **dns_nameservers**: Specifies the DNS servers to be used. It retrieves the value from a parameter called config_dns_nameserver.
  - **enable_dhcp**: Set to true, enabling DHCP for the subnet, allowing automatic IP address assignment to instances.
  - **host_routes**: An empty list, indicating no additional routing rules are defined for this subnet.
  - **ip_version**: Set to 4, indicating that this subnet uses IPv4 addressing.
  - **name**: The name of the subnet, here it is set to "subCluster-Network".
  - **network_id**: Retrieves the ID of the previously defined network (network_1), linking the subnet to that network.
  - **type**: Specifies the resource type as OS::Neutron::Subnet, indicating that this resource is a Neutron subnet.



```
subnet_1:
  properties:
    allocation_pools:
    - end: 10.20.1.100
      start: 10.20.1.10
    cidr: 10.20.1.0/24
    dns_nameservers: [ {get_param: config_dns_nameserver} ]
    enable_dhcp: true
    host_routes: []
    ip_version: 4
    name: subCluster-Network
    network_id:
      get_resource: network_1
  type: OS::Neutron::Subnet
```

Tacker - OpenStack NFV Orchestration



3.

CaaS for Telcos: A focus on Kubernetes

Why do we need a container orchestrator ?

How to

- manage faulty containers ?
- scale applications ?
- cope with the increasing number of deployed apps ?
- manage container resources ?
- enable containers monitoring ?
- manage service continuity upon host faults ?
- enable cluster deployment ?
- manage hosts maintenance ?
- ...

What is Kubernetes ?

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services
- It facilitates declarative configuration and automation
- It provides a framework to run distributed systems resiliently
- It takes care of scaling and failover
- It provides deployment patterns such as Canary deployments

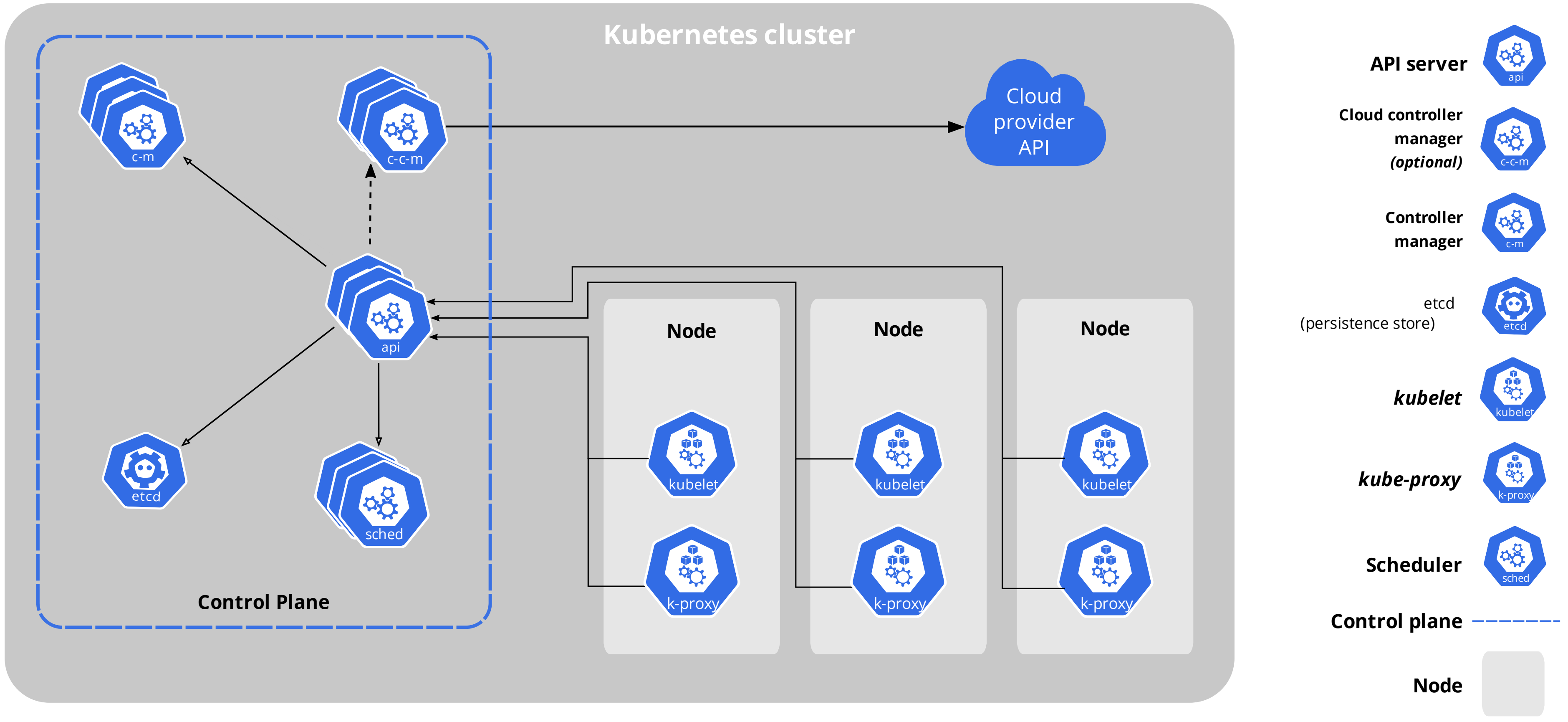
- Kubernetes features include: **Service discovery and load balancing**, Storage orchestration, **automated rollouts and rollbacks**, automatic bin packing, **self-healing**, secret and configuration management, etc.

https://landscape.cncf.io/

Kubernetes Resources [3]

- **Pods** are the **smallest deployable units** of computing that you can create and manage in Kubernetes. In concrete terms, a pod is a **group of one or more containers**, with **shared storage/network resources**, and a specification for how to run the containers
- **Controllers** are workload resources to create and manage multiple Pods
  - ReplicaSet: maintains a stable set of replica Pods running at any given time
  - Deployments: provides declarative updates for Pods and ReplicaSets
  - ...
- Namespaces, Services, ConfigMaps, Volumes, Custom Resources, ...

Kubernetes Components [3]

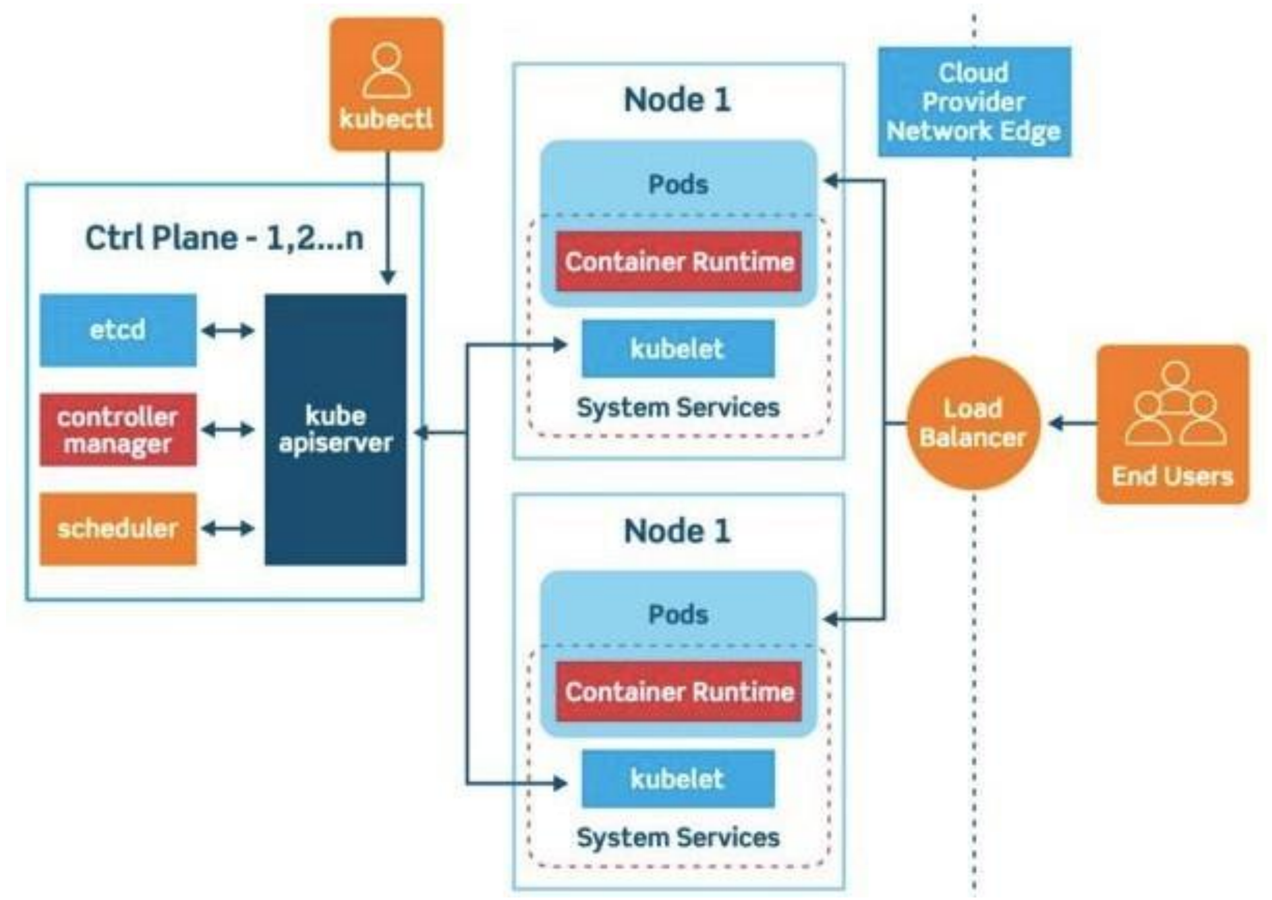


https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Components [3]

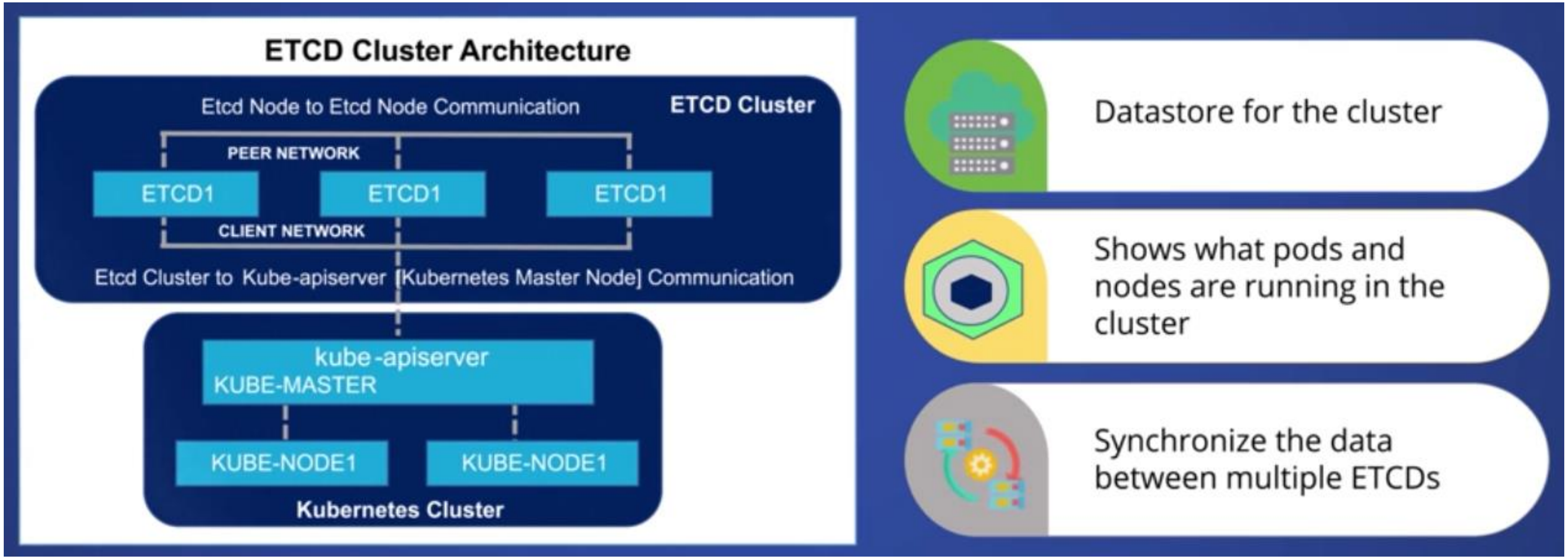
Control Plane (master node) Components

- **Kube-apiserver**: The API server exposes the Kubernetes API. It is the front end for the Kubernetes control plane
- **etcd**: Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data



platform9.com

Kubernetes Components [3]


ETCD Cluster Architecture
Etcd Node to Etcd Node Communication
ETCD Cluster
PEER NETWORK
ETCD1
ETCD1
ETCD1
CLIENT NETWORK
Etcd Cluster to Kube-apiserver [Kubernetes Master Node] Communication
kube-apiserver
KUBE-MASTER
KUBE-NODE1
KUBE-NODE1
Kubernetes Cluster

Datastore for the cluster

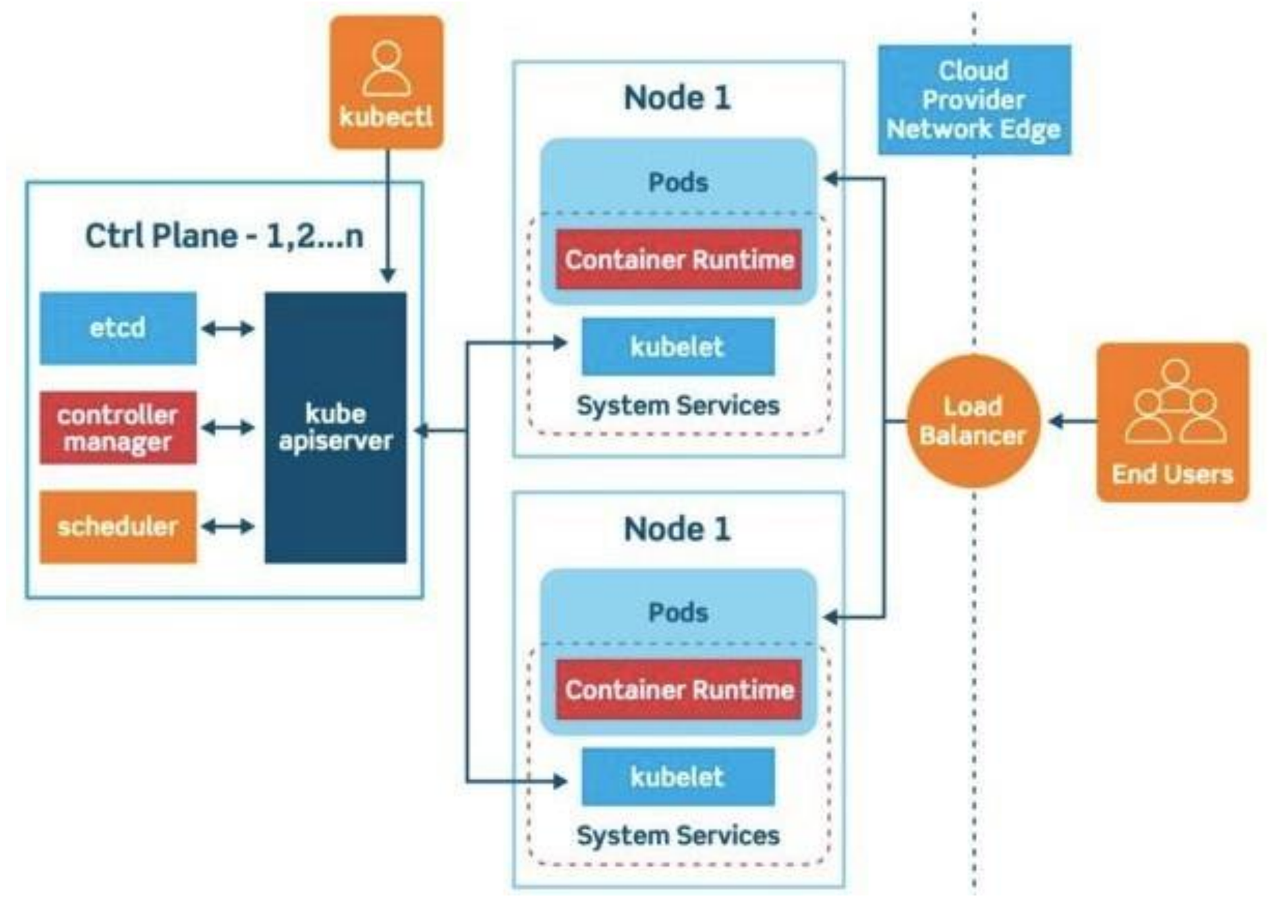
Shows what pods and nodes are running in the cluster

Synchronize the data between multiple ETCDs

Kubernetes Components [3]

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 include: individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines

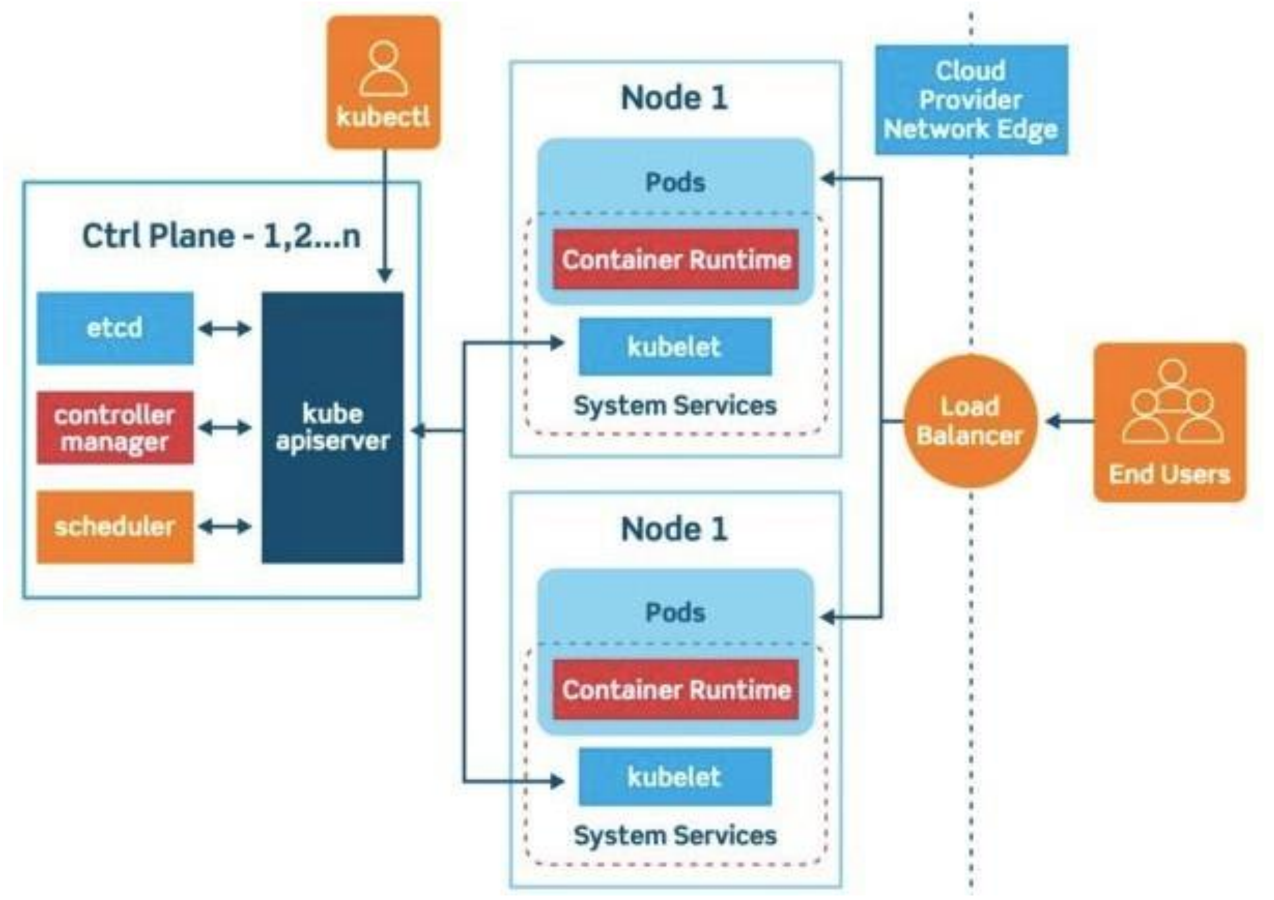


platform9.com

Kubernetes Components [3]

Control Plane Components

- **Kube-controller-manager**
  - *Node Ctrl*: notices and responds when nodes go down
  - *Replication Ctrl*: Responsible for maintaining the correct number of pods for every replication controller object in the system
  - *Endpoints Ctrl*: Populates the Endpoints object
  - *Service Account & Token Ctrl*: Create default accounts and API access tokens for new namespaces
- **Cloud-controller-manager**: embeds cloud-specific control logic to allow the interaction with cloud provider's API

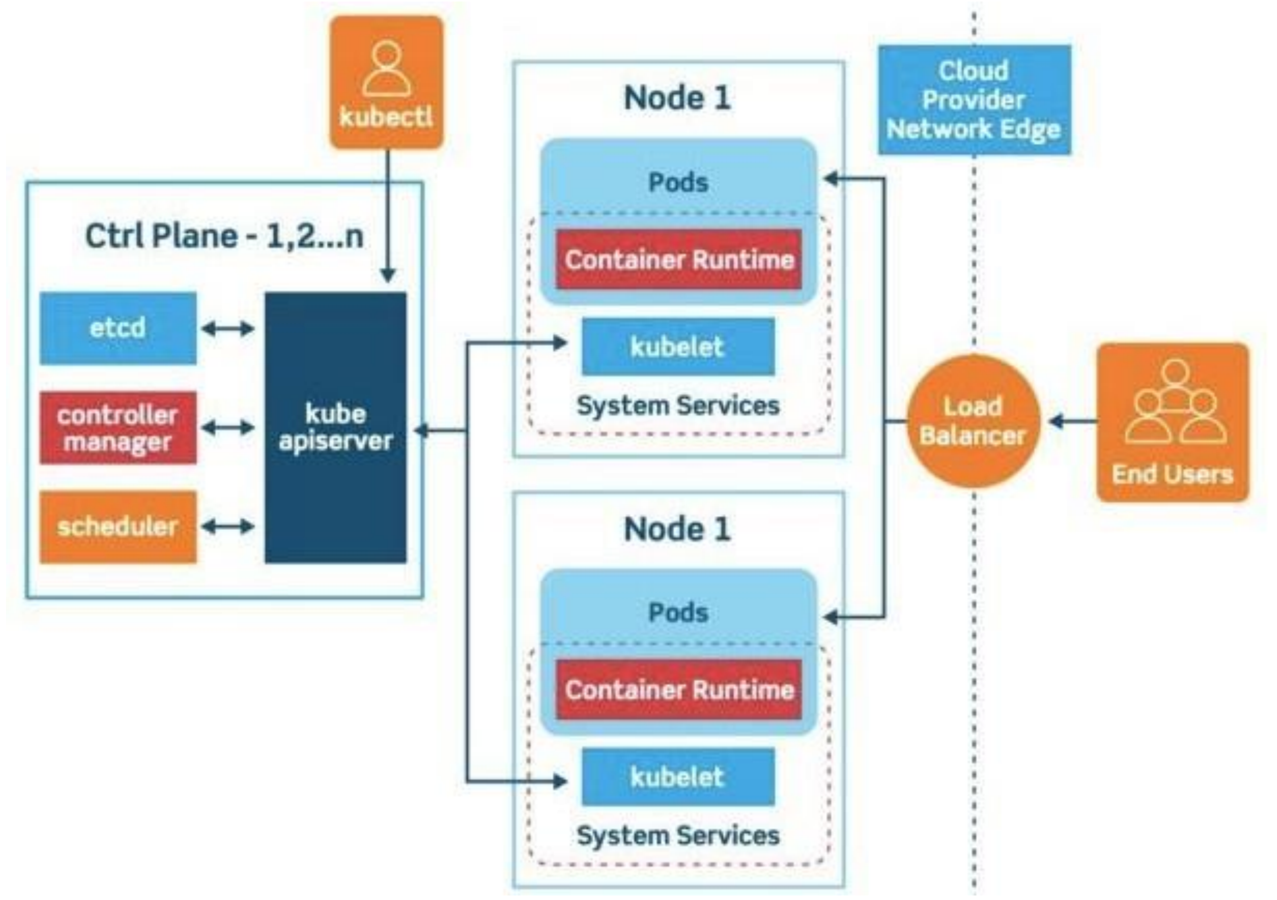


platform9.com

Kubernetes Components [3]

Worker 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

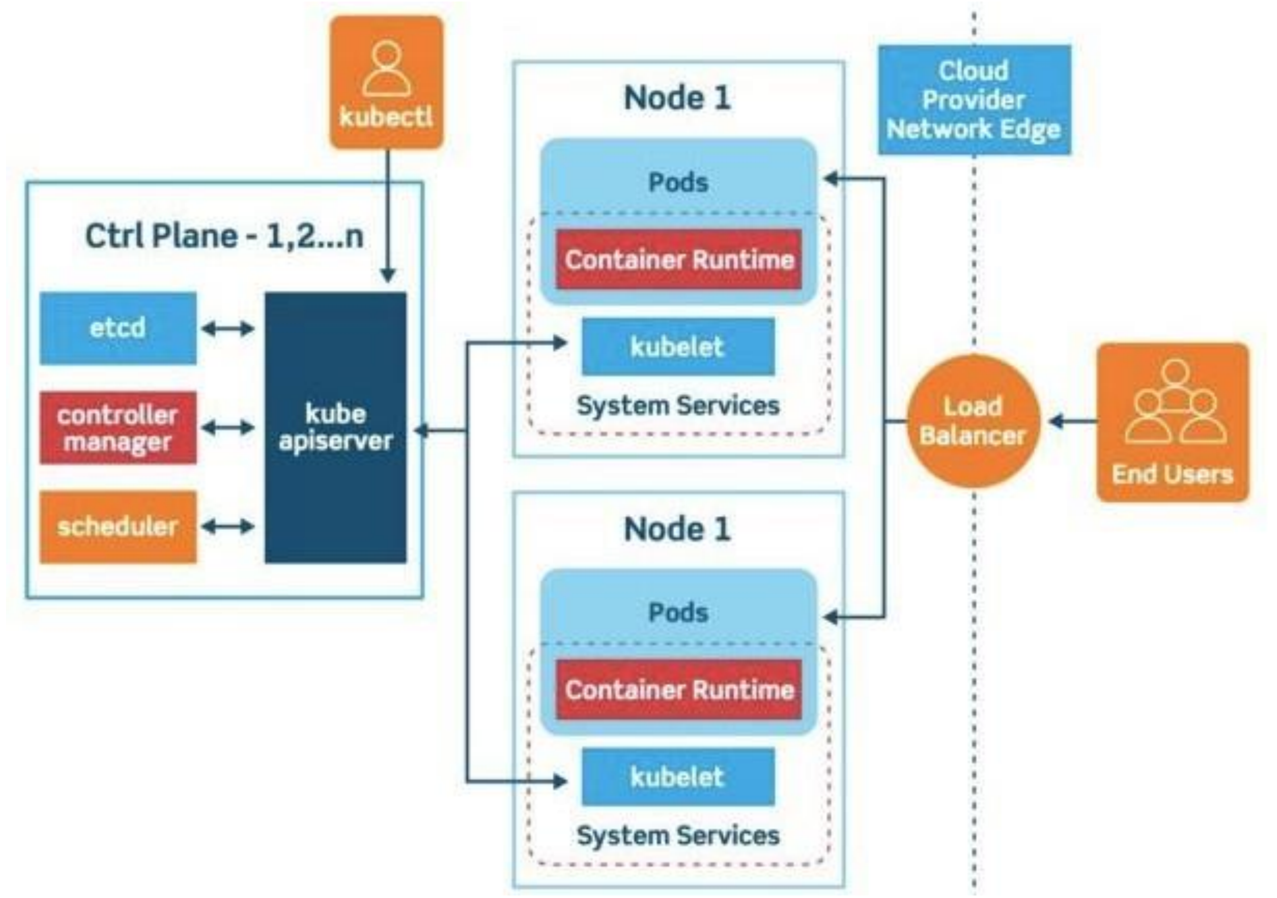


platform9.com

Kubernetes Components [3]

Worker Node Components

- **kube-proxy**: kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.



platform9.com

Container Runtime Interface (CRI) & Container Runtimes

- A CRI is the lowest layer of a Kubernetes worker node that, among other things, starts and stops containers
- The most widely known runtime is Docker, but it is not alone

docker	containerd		rkt	cri-o
The startup that made OS Containers accessible to everyone	Developed by Docker and donated to the CNCF in 2017	Developed by Docker and donated to the Open Container Initiative (OCI) living under the CNCF	Developed by the former CoreOS (now acquired by RedHat)	Developed and maintained by the OCI/CNCF

Container Network Interface (CNI) & Network Plugins

- Container Network Interface (CNI) is a specification and a set of libraries for configuring network interfaces in Linux containers. In Kubernetes, CNI is used to manage networking for pods.

How CNI Works in Kubernetes?

1. **Pod Creation**:
    1. When a pod is created, the kubelet calls the CNI plugin to set up networking.
2. **IP Address Assignment**:
    1. The CNI plugin assigns an IP address to the pod and configures the network interface.
3. **Routing**:
    1. The CNI plugin manages routing rules to ensure that traffic can flow between pods and external networks.

Container Network Interface (CNI) & Network Plugins

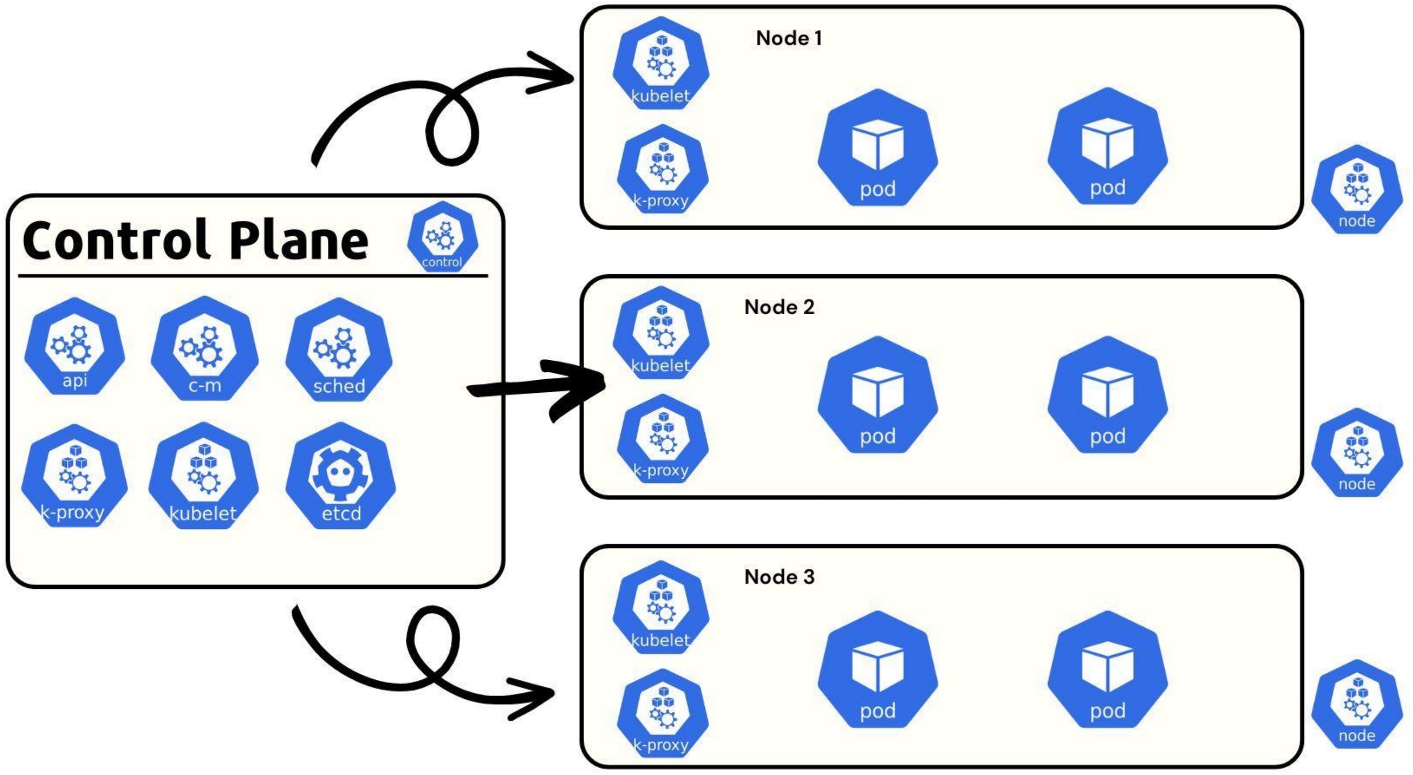
- CNI is a set of specification and libraries for writing plugins to configure network interfaces in Linux containers, along with a number of supported plugins

- CNI handles network connectivity of containers and removes allocated resources when the container is deleted

Full list: https://github.com/containernetworking/cni

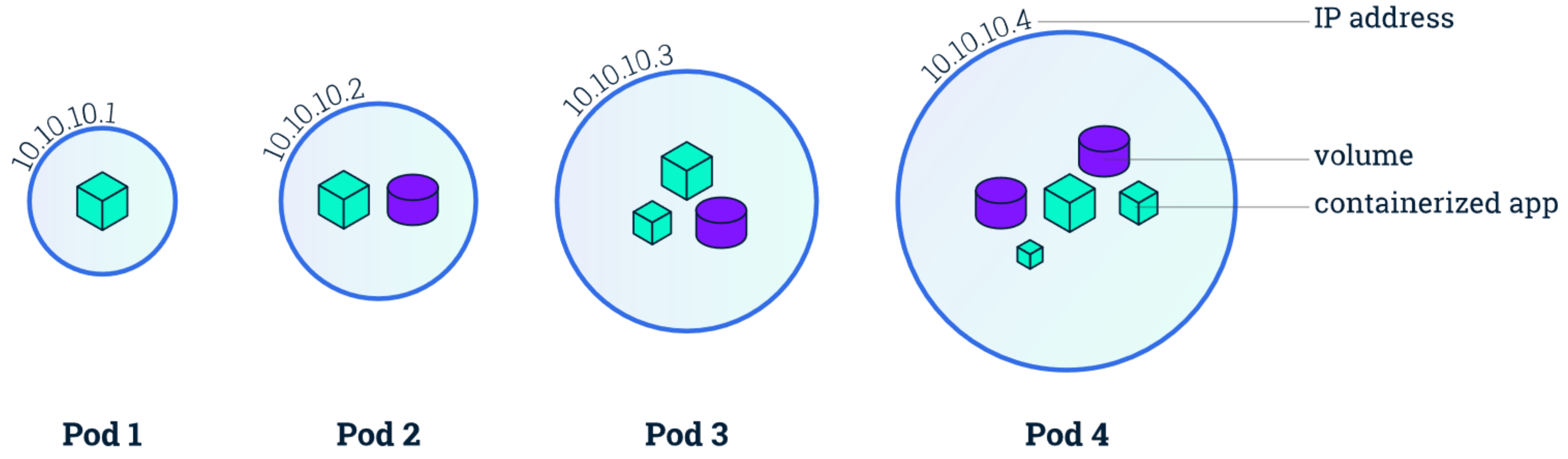
Kubernetes Cluster

A Kubernetes cluster consists of a control plane plus a set of worker machines



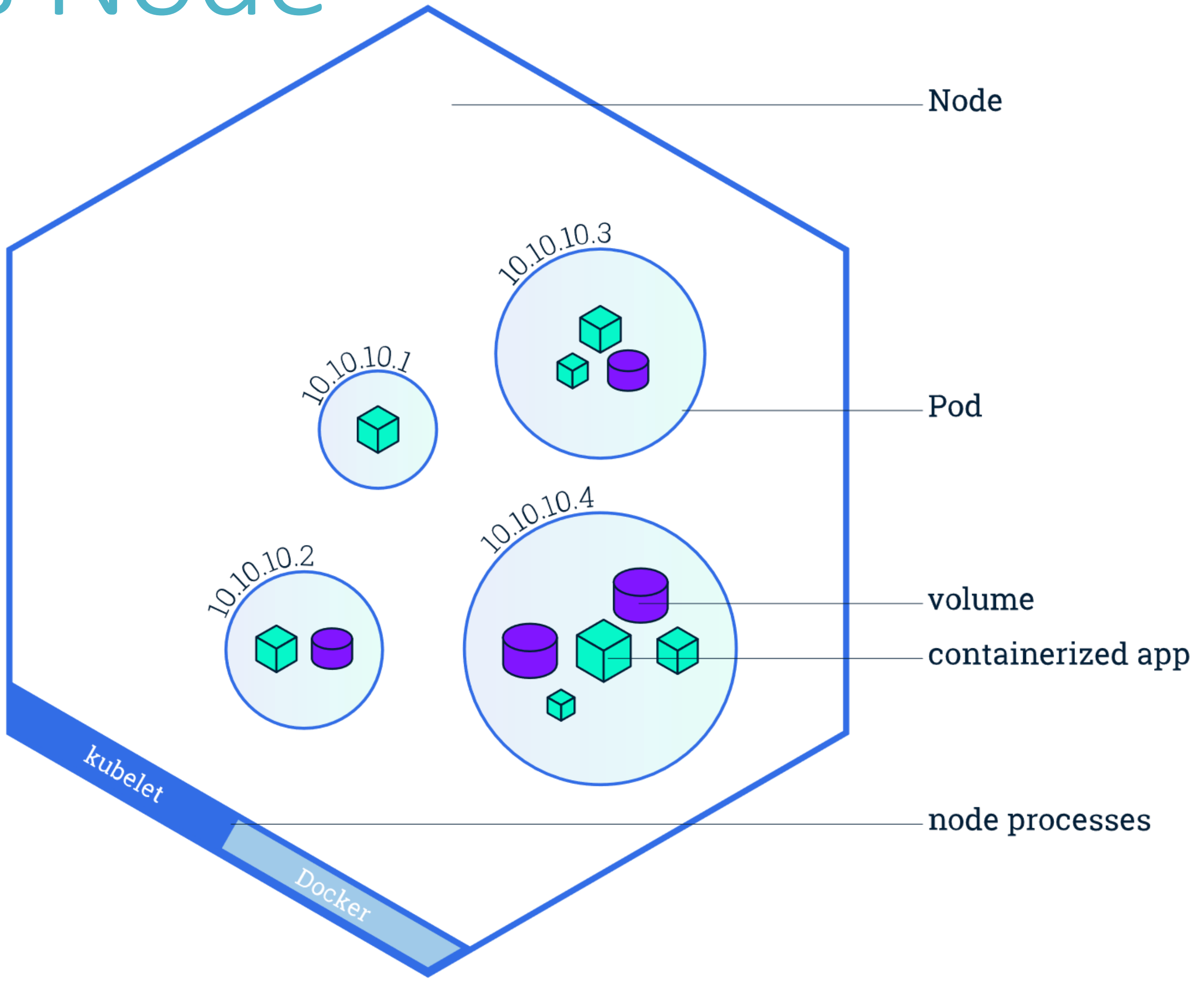
Kubernetes Pods

- **Definition:** The smallest deployable unit in Kubernetes, encapsulating one or more containers.
- **Lifecycle:** Pods can be created, destroyed, and managed independently.



https://kubernetes.io/fr/docs/tutorials/kubernetes-basics/explore/explore-intro/

Kubernetes Node



https://kubernetes.io/fr/docs/tutorials/kubernetes-basics/explore/explore-intro/

Kubernetes services

- **Definition:** An abstraction that defines a logical set of pods and a policy for accessing them.
- **Types of Services:**
  - **ClusterIP:** Exposes the service on a cluster-internal IP.
  - **NodePort:** Exposes the service on each node's IP at a static port.
  - **LoadBalancer:** Exposes the service externally using a cloud provider's load balancer.

Kubernetes ReplicatSets

- **Purpose:** Ensures that a specified number of pod replicas are running at any given time.
- **Functionality:** Monitors the state of pods and replaces any that fail or are deleted.
- **Example:** If a ReplicaSet is configured for three replicas, it will maintain exactly three pods.

Kubernetes Deployments

- **Definition:** A higher-level abstraction that manages ReplicaSets and provides declarative updates.
- **Features:** Supports rolling updates, rollbacks, and scaling.

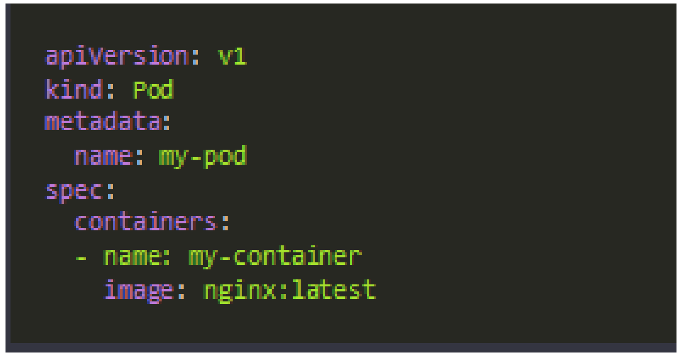
Kubernetes config maps and secrets

- **ConfigMaps:**
  - Store non-sensitive configuration data in key-value pairs.
  - Can be referenced in pods to configure applications.
- **Secrets:**
  - Store sensitive information (e.g., passwords, tokens) securely.
  - Base64-encoded and can be mounted as files or environment variables.

Kubernetes Persistent Storage

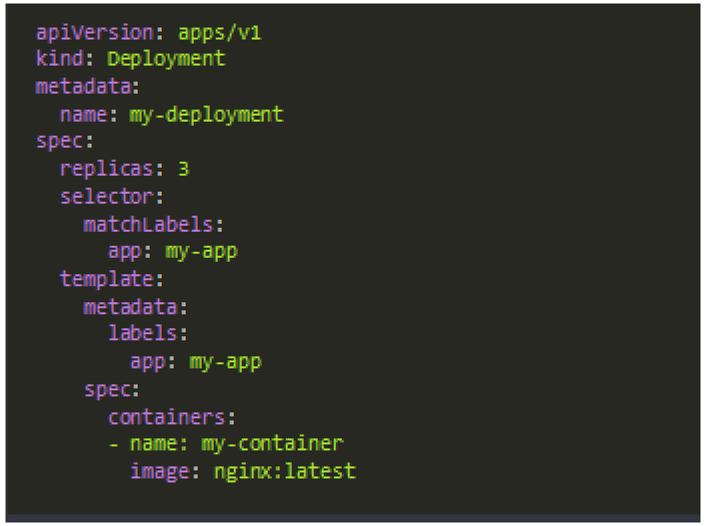
- **Definition:** Manages storage resources for stateful applications.
- **Types:**
  - **Persistent Volumes (PV):** Storage resources in the cluster.
  - **Persistent Volume Claims (PVC):** Requests for storage by users.
- **Use Cases:** Essential for databases and applications requiring data persistence..

How to deploy a pod ?



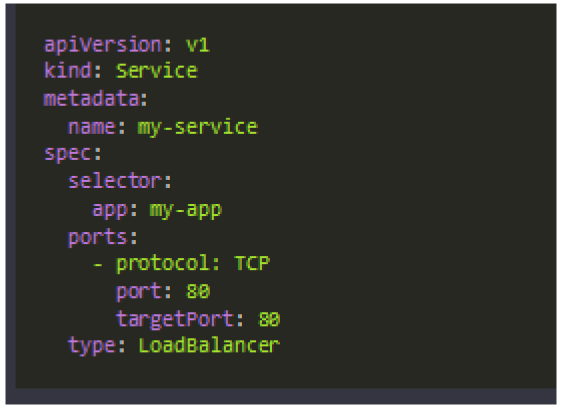
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx:latest
```

Pod



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-container
        image: nginx:latest
```

Deployment



```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Service

Kubectl

- **Definition:** Command-line tool for interacting with Kubernetes clusters.
- **Purpose:** Allows users to manage cluster resources and applications.
- **Installation:** Typically installed via package managers or binaries.

Basic kubectl Commands

- **`kubectl get pods`:** Lists all pods in the current namespace.
- **`kubectl apply -f <file.yaml>`:** Creates resources defined in a YAML file.
- **`kubectl delete pod <pod-name>`:** Deletes a specified pod.
- **`kubectl describe <resource> <name>`:** Provides detailed information about a resource.

Managing Deployments with kubectl

- **`kubectl apply -f <file.yaml>`:** Applies configuration changes from a file.
- **`kubectl rollout status deployment/<name>`:** Monitors the status of a deployment rollout.
- **`kubectl scale deployment/<name> --replicas=<number>`:** Adjusts the number of replicas for a deployment.

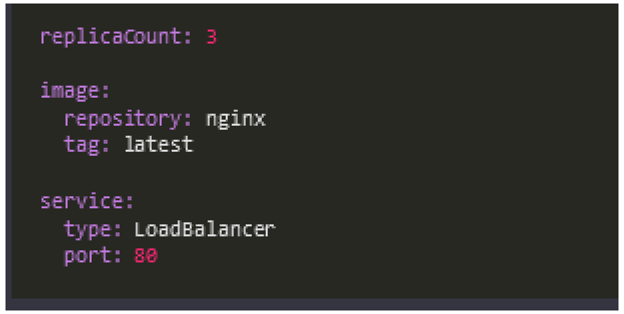
Viewing Logs and Debugging

- **`kubectl logs <pod-name>`:** Retrieves logs from a specified pod.
- **`kubectl exec -it <pod-name> -- /bin/bash`:** Opens a shell in a running pod for debugging.
- **`kubectl port-forward <pod-name> <local-port>:<pod-port>`:** Forwards a port from a pod to the local machine.

Helm

- **Definition:** A package manager for Kubernetes, simplifying application deployment.
- **Features:** Allows for templating and versioning of Kubernetes resources.
- **Command Example:** helm install <chart-name> to deploy an application.

Helm Example

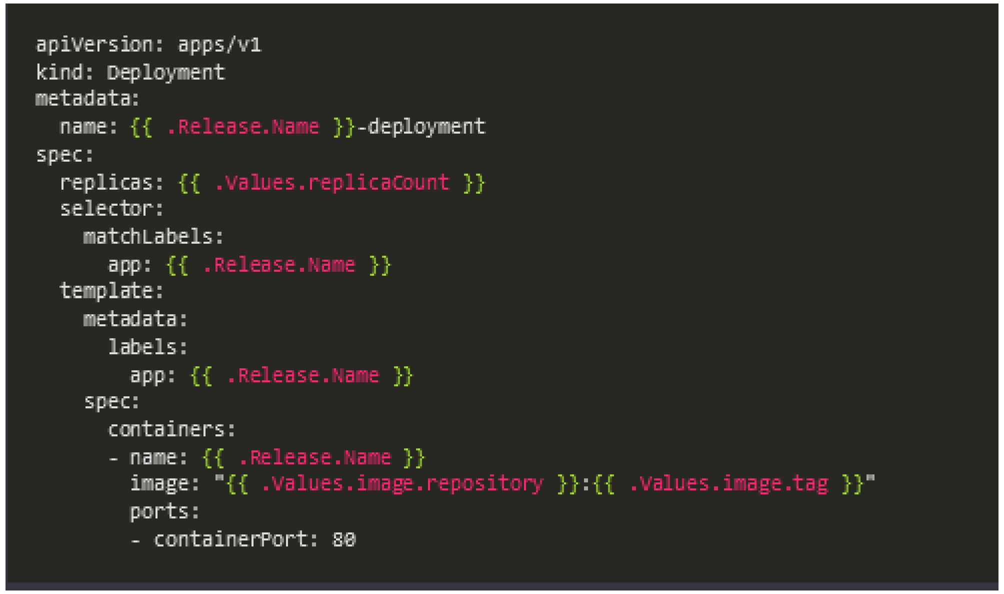


```yaml
replicaCount: 3

image:
  repository: nginx
  tag: latest

service:
  type: LoadBalancer
  port: 80
```

Values,yaml



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Release.Name }}-deployment
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ .Release.Name }}
    spec:
      containers:
      - name: {{ .Release.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - containerPort: 80
```

Helm chart

Quiz

https://play.kahoot.it/v2/?quizId=960a0cd6-a041-47c5-8e29-9d98fa568782&hostId=df5bf548-6a73-45dd-b0d9-7e8c8973798a

4.
Key Takeaways

Key Takeaways

- OpenStack IaaS provides a fully featured cloud operating system with multiple services to virtualize and abstract compute, network and storage resources across multiple locations
- OpenStack provides basic automation through the Heat service
- Kubernetes CaaS & container orchestration system provides out-of-the box automation, auto-healing and scaling mechanisms that are essential for the 5G