kubernetes

Lev Epshtein
lev@opsguru.io

NAYA College

Kubernetes

Basic

- What is K8S
- K8S Concepts & Components
- K8S OBJECTS
  - POD
  - Namespace
  - Deployment
  - Labels/Annotations
  - Services
    - ClusterIP
    - NodePort
    - LoadBalancer
  - Ingress

NAYA College

- K8S -

WHAT IT’S ALL ABOUT

NAYA College

K8S - WHAT IT’S ALL ABOUT

From K8S WEBSITE:

“Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.”

K8S - WHAT IT’S ALL ABOUT

K8S design principles

- Kubernetes implements composable control processes
- The processes continuously drive the current state of an object towards the provided desired state
- Facilitates declarative configuration (manifest)
- Kubernetes extends its feature using pluggable modules (Plugins / Operators / CRD)

K8S - WHAT IT'S ALL ABOUT

In plain english k8s is:

- **Container orchestration platform**
- Environment agnostic solution (can run on any infrastructure)
- Originated by Google Borg
- K8S Allows developers / system operators to cut to the cord and truly run a container-centric dev / microservice environment

NAYA College

K8S - WHAT IT'S ALL ABOUT

Kubernetes:

- Does not deploy source code and does not build your application - CI/CD Does
- Does not provide application-level services, such as middleware (e.g., message buses), data-processing frameworks (for example, Spark), databases (e.g., mysql), caches, nor cluster storage systems (e.g., Ceph) as built-in services
- Does not dictate logging, monitoring, or alerting solutions (at least not as good as Prometheus for monitoring and ECK Elastic Cloud K8s)
- Does not provide nor mandate a configuration language/system (e.g., jsonnet). It provides a declarative API that may be targeted by arbitrary forms of declarative specifications.
- Does not provide nor adopt any comprehensive machine configuration, maintenance, management, or self-healing systems.

- K8S -

Core Concepts & Components

NAYA College

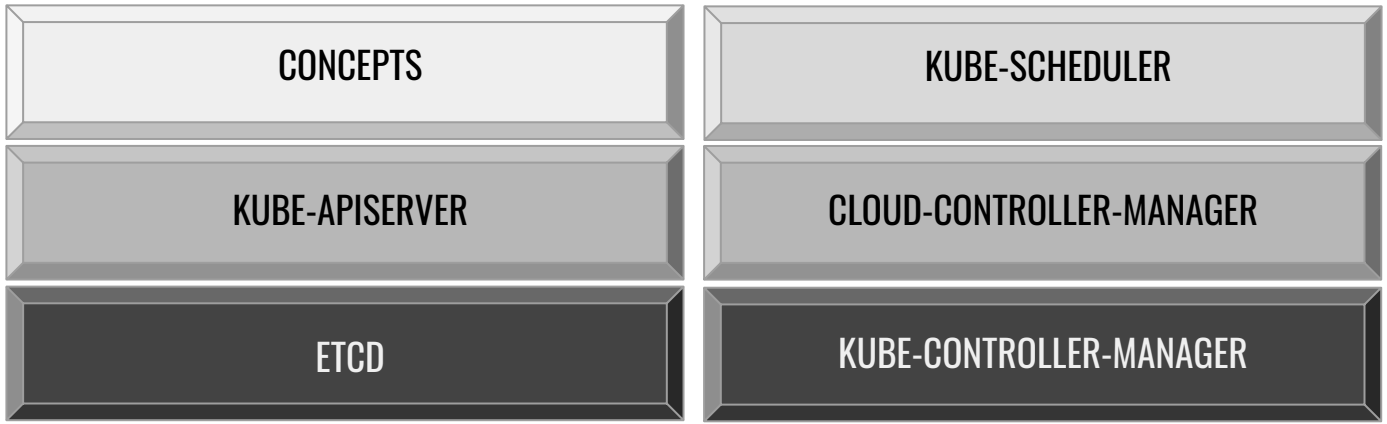
K8S - CORE CONCEPTS

To master K8S like any other framework or system we work on,

We are required to have a deep understanding of what makes it tick.

In the next slides we will review and discuss K8S Architecture in details.

K8S CORE CONCEPTS - Control Plane (Master nodes)



NAYA College

K8S - CORE CONCEPTS

Every K8S cluster is made of two types of roles - Master & Node.

While the Master controls and manage our K8S cluster by maintaining our cluster state.

Our Node Role maintaining running pods and providing the Kubernetes runtime environment

K8S - CORE CONCEPTS

Understanding K8S system and abstraction

To work with Kubernetes, we use Kubernetes API objects to describe our cluster's **desired state**:

- What applications or other workloads we want to run
- What container images they use, the number of replicas, what network and disk resources we want to make available.
- Setting our desired state by creating objects using the Kubernetes API (typically via the command-line interface - "kubectl")

Once we've set our desired state,

Kubernetes Control Plane works to make the cluster's current state match the desired state

- K8S Master -

Core Concepts & Components

NAYA College

K8S - CORE CONCEPTS

The Kubernetes Master role is a collection of three processes that run on a single node in our cluster (in production we will deploy it as quorum).

The process that designated a node to become a master node are:

1. **Kube-apiserver** who Validates and configures data for the api objects which include containers, services, replicationcontrollers, and others.
2. **Kube-controller-manager** - is a an application control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired state.
3. **Kube-scheduler** has the job to take pods [TBD:For now we think about pods as containers] that aren't bound to a node, and assign them one along with hardware/software/policy constraints

K8S - CORE CONCEPTS

K8S **Master components** are the cluster's **control plane**. The Master components make global decisions about the cluster state, and they detect and respond to cluster events (for example, running multiple pods[containers] of our application when required and scale in the number of running pods).

K8S - CORE CONCEPTS

Master node Deep Dive:

The Kubernetes **Control Plane** consists of a collection of processes running on our cluster

- **Kube-Apiserver** - 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** - highly-available key value FILESYSTEM store used as Kubernetes' backing store for all cluster data
- **Kube-Scheduler** - Watches newly created pods that have no node assigned, and selects a node for them to run on. Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines.

K8S - CORE CONCEPTS

Master node Deep Dive:

The **Kubernetes Control Plane** consists of a collection of processes running on our cluster

- **Kube-controller-manager** which includes separate process but for complexity reduction they are running as one binary
  - **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

K8S - CORE CONCEPTS

Master node Deep Dive:

Cloud-controller-manager [Used when running on cloud) - runs controllers that interact with the underlying cloud providers. cloud-controller-manager allows cloud vendors code and the Kubernetes core to evolve independent of each other and develops functionality (by the cloud providers) that will be linked to K8S cloud-controller-manger.

The following controllers have cloud provider dependencies

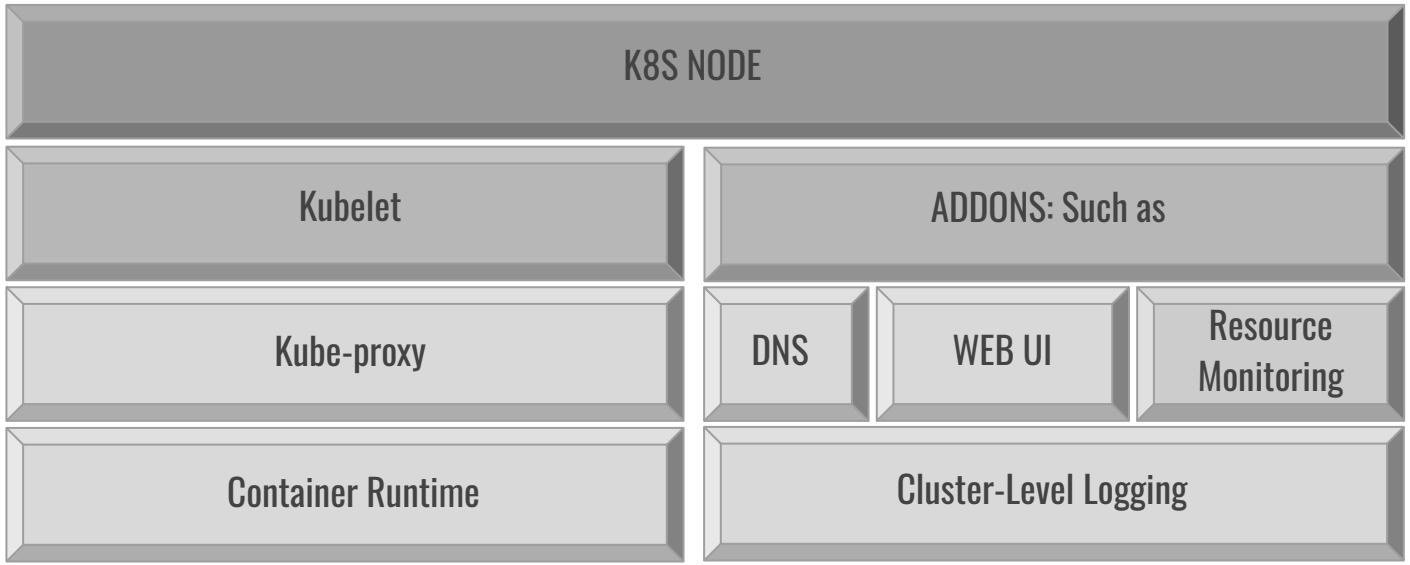
- **Node Controller**: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route Controller**: For setting up routes in the underlying cloud infrastructure
- **Service Controller**: For creating, updating and deleting cloud provider load balancers
- **Volume Controller**: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

- K8S Nodes -

Core Concepts & Components

NAYA College

K8S CORE CONCEPTS - Worker NODE



NAYA College

K8S - CORE CONCEPTS

Node

Node - may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include Docker, kubelet and kube-proxy. See The Kubernetes Node section in the architecture design doc for more details.

K8S - CORE CONCEPTS

Node

Each individual non-master node in our cluster runs two K8S processes:

- **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 [TBD: yaml files with declaration on how to deploy our application] that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.
- **Kube-proxy** - is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept [TBD:]. kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster. kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

K8S - CORE CONCEPTS

Node

Each individual **non-master** node in our cluster also runs:

Container Runtime - The container runtime is the software that is responsible for running containers. Kubernetes runtimes: Docker , rkt (coreOS) and CRI-O

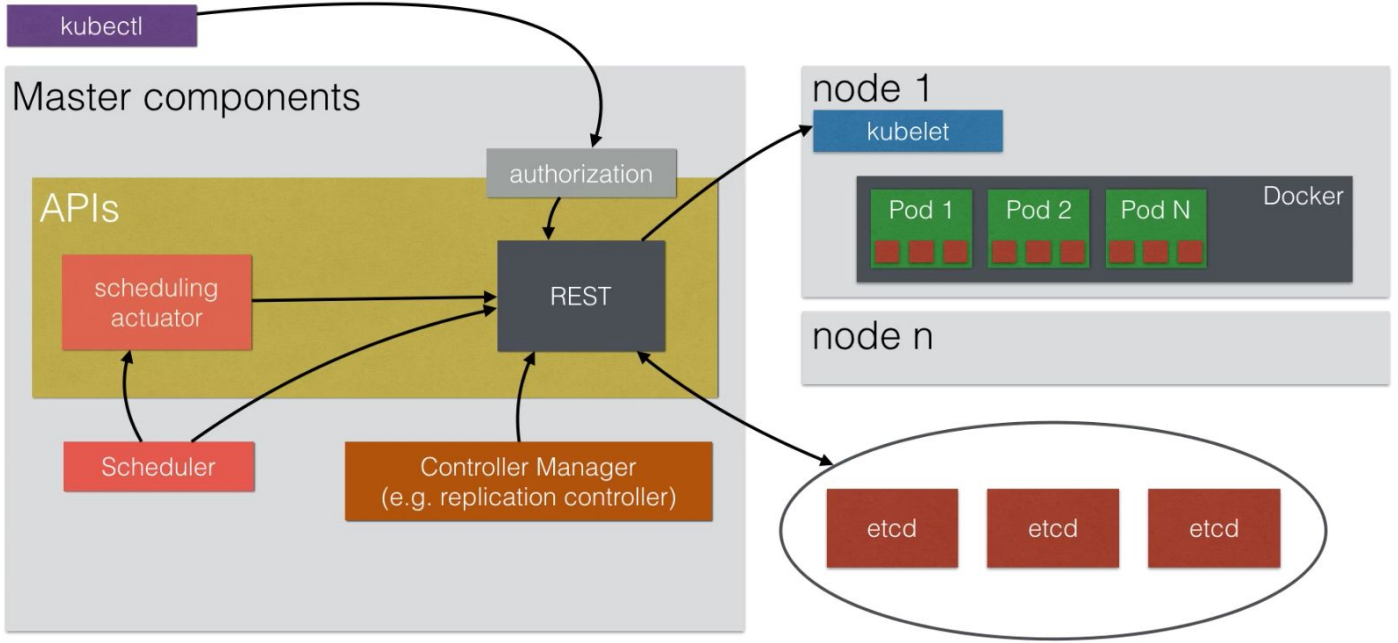
K8S - CORE CONCEPTS

Addons

- **DNS** - While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it. Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services. Containers started by Kubernetes automatically include this DNS server in their DNS searches.
- **WEB UI** - General purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself
- **Container Resource monitoring** - Records generic time-series metrics about containers in a central database, and provides a UI for browsing that data
- **Cluster-level Logging** - A Cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

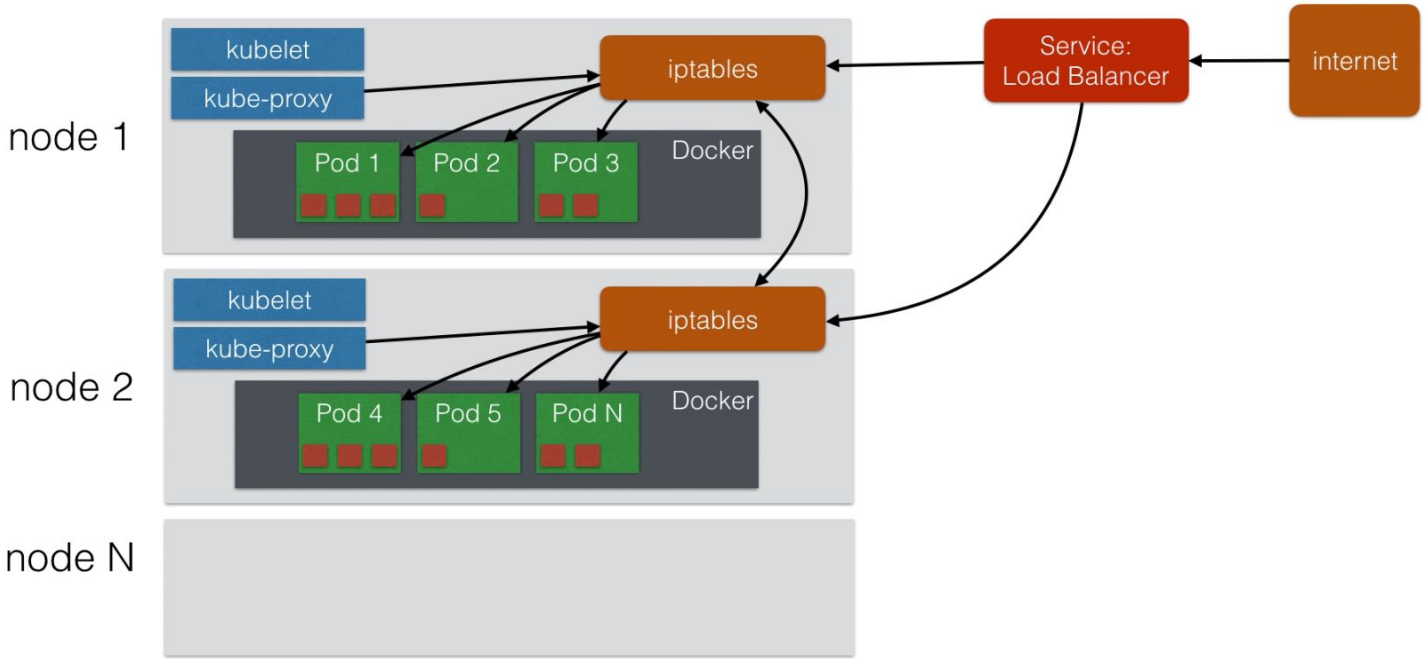
K8S - CORE CONCEPTS

SUMMARY NODE & MASTER



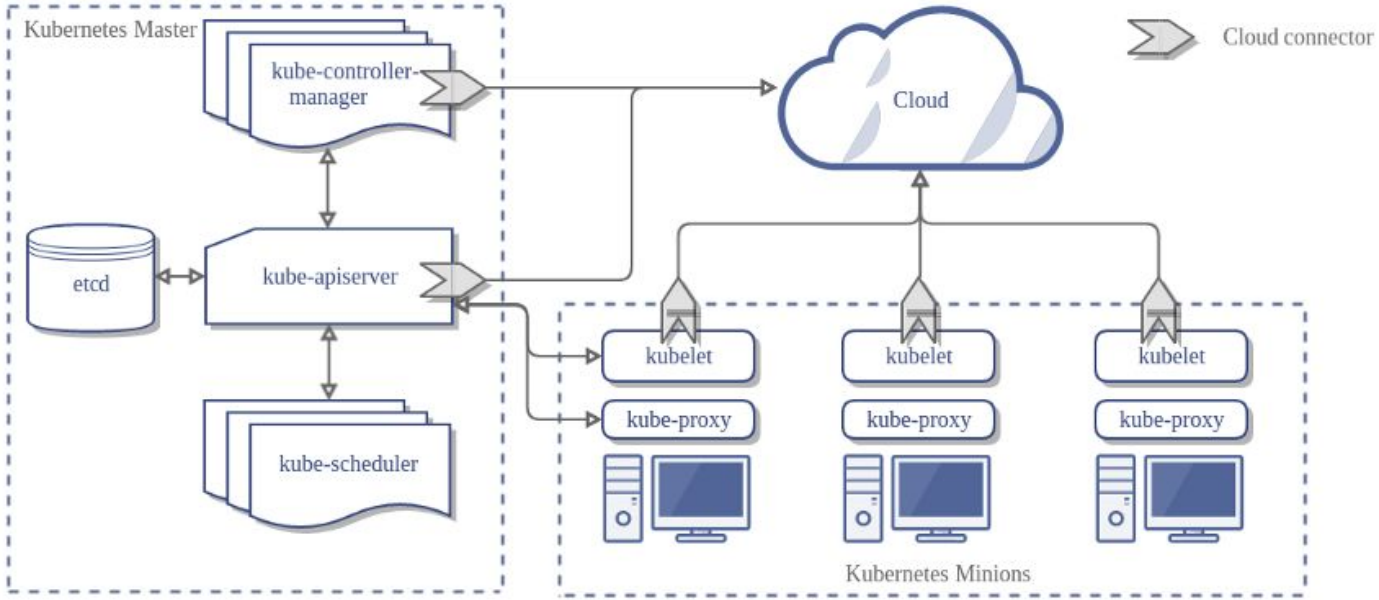
K8S - CORE CONCEPTS

SUMMARY NODE



K8S - CORE CONCEPTS

SUMMARY



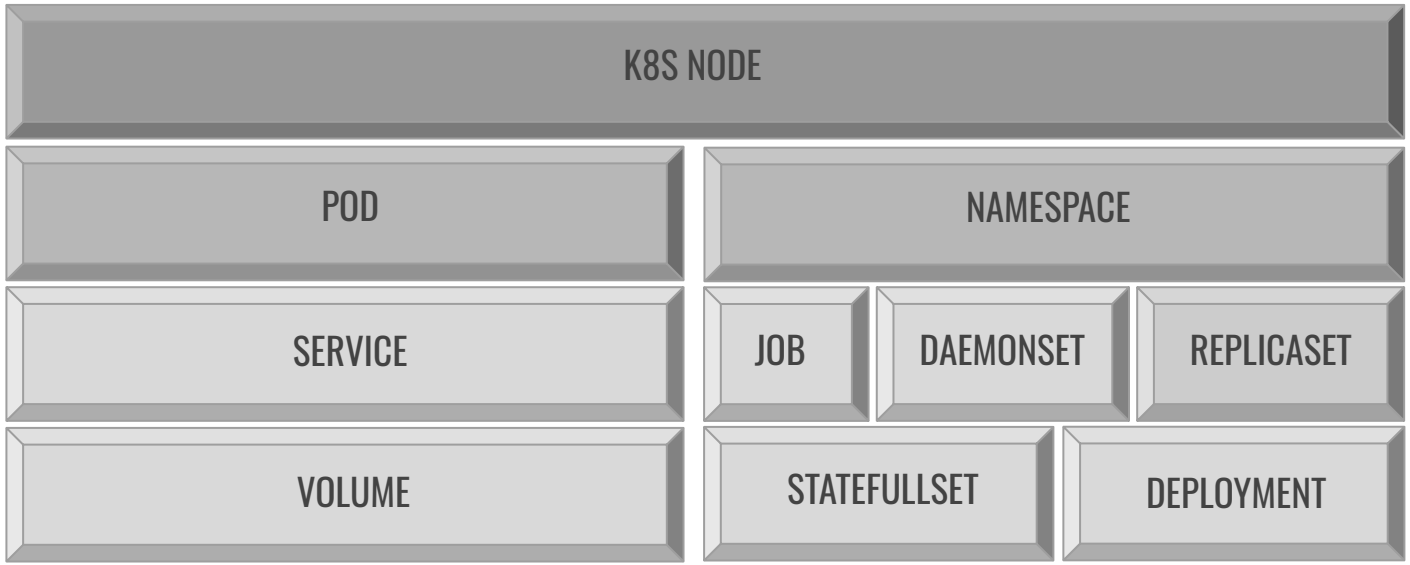
NAYA College

Course Demos and Labs:

git clone https://github.com/levep/devops-k8s.git

- K8S Objects -

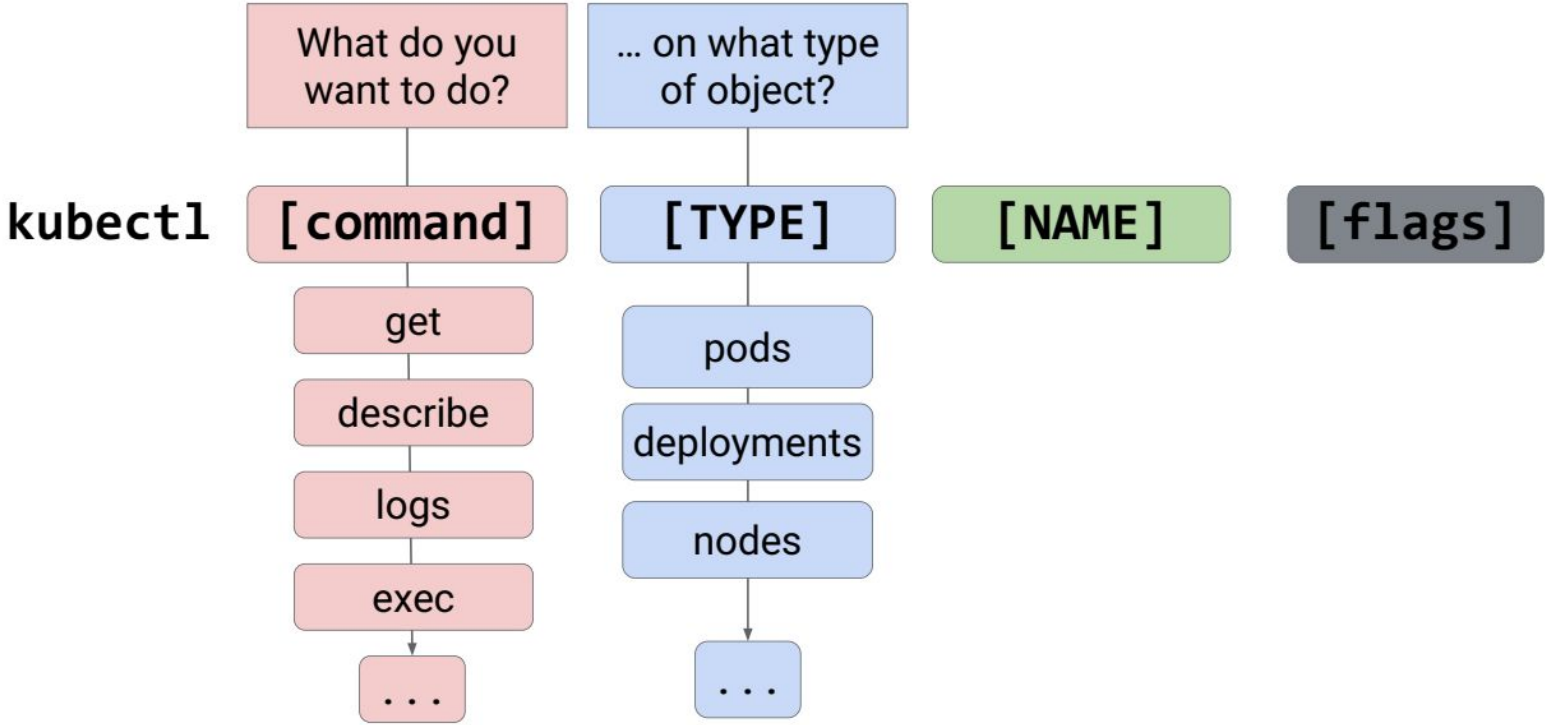
K8S OBJECTS & CONTROLLERS



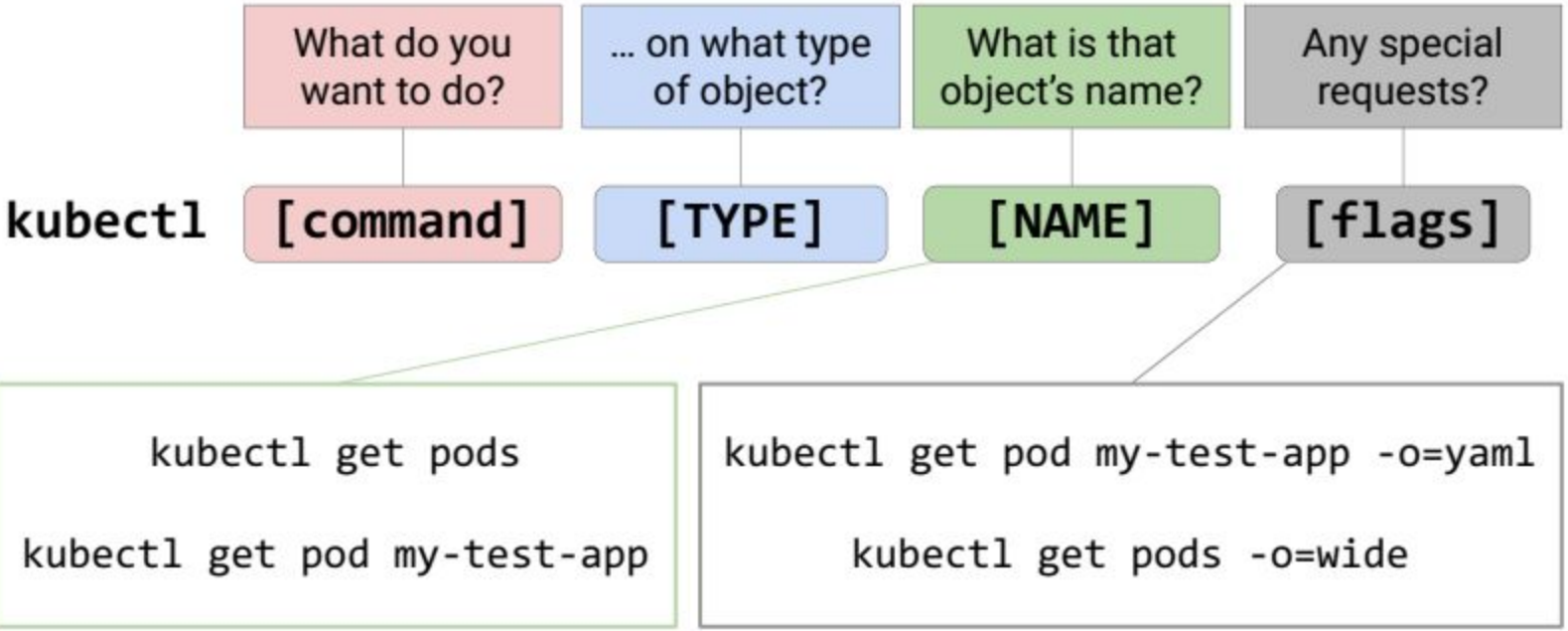
NAYA College

Before we begin let’s explore **kubectl**

The kubectl command syntax has several parts



The kubectl command syntax has several parts



NAYA College

The kubectl command has many uses

- Create Kubernetes objects
- View objects
- Delete objects
- View and export configurations

https://kubernetes.io/docs/reference/kubectl/cheatsheet/

- K8S POD -

NAYA College

K8S OBJECTS: PODS

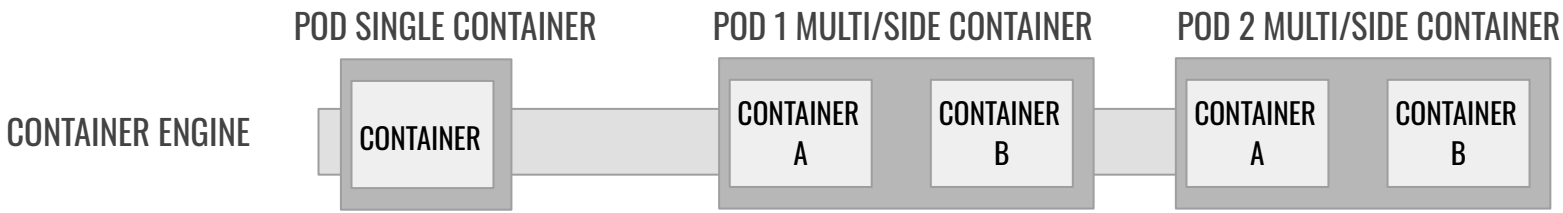
POD - A Pod is the basic building block of Kubernetes–the smallest and simplest unit in the Kubernetes object model that you create or deploy. A Pod represents a running process on your cluster. A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.

The **pod** is an **abstraction** layer that allows k8s to manage and group containers using the node **kubelet** agent

K8S OBJECTS: PODS

There are two ways to describe a POD in a K8S cluster

- **Pods that run a single container** - The “one-container-per-Pod” model is the most common Kubernetes use case; in this case, you can think of a Pod as a wrapper around a single container, and Kubernetes manages the Pods rather than the containers directly
- **Pods that run multiple containers** that need to work together aka as SIDECARS



K8S OBJECTS: PODS

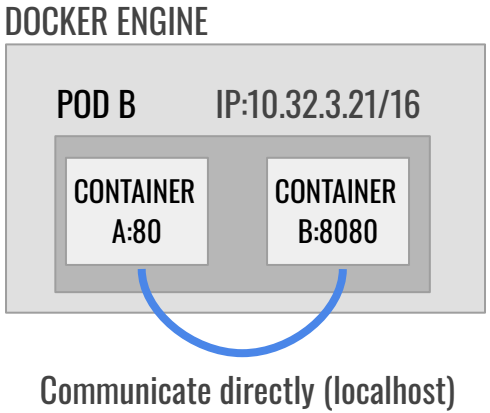
When do we run Multi/SideCar container setup in a pod?

- **Pods** that run **multiple containers** that need to work together - A Pod might encapsulate an application composed of multiple co-located containers that are tightly coupled and **need to share resources** such as memory, storage or network with zero latency (multi containers in a pod are always hosted in the same NODE). These co-located containers might form a single cohesive unit of service–one container serving files from a shared volume to the public, while a separate “sidecar” container refreshes or updates those files. The Pod wraps these containers and storage resources together as a single manageable entity.

K8S OBJECTS: PODS

Pods provide two kinds of shared resources for their constituent containers: networking and storage.

Networking - Each Pod is assigned a unique IP address. Every container in a Pod shares the network namespace, including the IP address and network ports. Containers inside a Pod can communicate with one another using localhost. When containers in a Pod communicate with entities outside the Pod, they must coordinate how they use the shared network resources (such as ports).

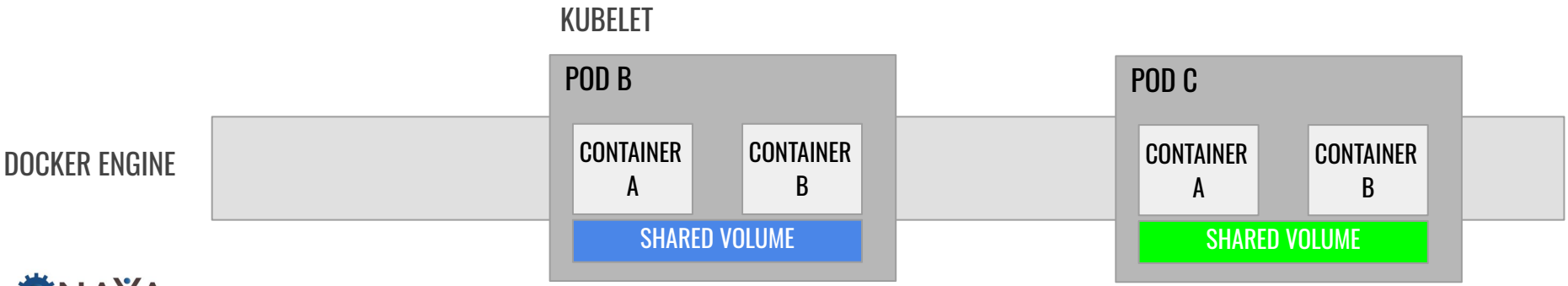


NAYA College

K8S OBJECTS: PODS

Pods provide two kinds of shared resources for their constituent containers: networking and storage.

- **Storage** - A Pod can specify a set of shared storage volumes. All containers in the Pod can access the shared volumes, allowing those containers to share data. **Volumes also allow persistent data in a Pod to survive in case one of the containers within needs to be restarted.** [TBD: Volumes advanced and how k8s implement shared storage in a pod and in a cluster]



NAYA College

- K8S VOLUMES BASICS -

MAKE PODS STORAGE PERSISTENT

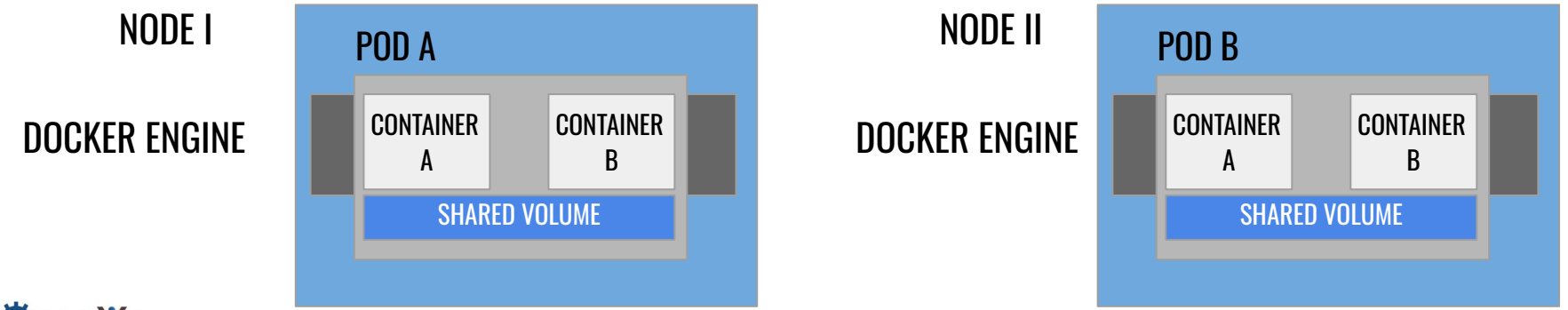
ADVANCED SG/PV/PVC TBD IN THE NEXT PART

NAYA College

K8S VOLUMES

Files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers.

- First, when a container crashes, kubelet will restart it, but the files will be lost - the container starts with a clean state.
- Secondly when running containers together in a Pod it is often necessary to share files between those containers. The Kubernetes Volume abstraction solves both of these problems as seen below



NAYA College

K8S SPECS - VOLUME

Making our pods data persistent using volumes

Same as in docker (but not really the same as docker can support only one Volume driver per container and K8S multiple drivers as described here) we use Volume to store persistent data cross containers which will outlive containers

```
# 1. We define the volume and driver type as followed

volumes:
  - name: redis-persistent-storage # Volume name
    emptyDir: {} # Volume type

# 2. We mount the volume within a container definition

volumeMounts:
 - name: redis-persistent-storage  # name must match the volume name defined in volumes
   mountPath: /data/redis # mount path within the container
```

NAYA College

Demo/Labs

Let’s explore POD:

In our repository

$ cd cognyte-devops-32/k8s-1/runnig_pod/

Follow README.MD

NAYA College

- K8S Deployment -

NAYA College

K8S OBJECTS: DEPLOYMENT

Deployment - A controller that provides declarative **UPDATES** for Pods and ReplicaSets.

[TBD: Replicaset define the number of replicas we wants for each deployments PODS]

We describe a **desired state** in a Deployment object, and the Deployment controller changes the actual state to the desired state at a controlled rate.

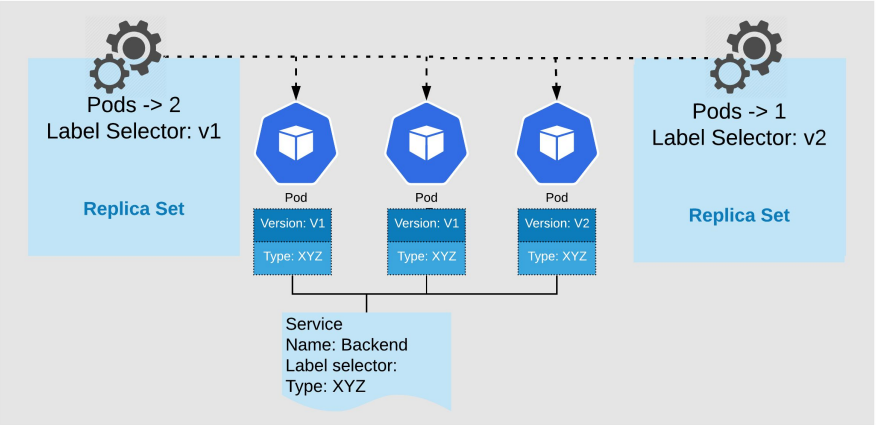
We use the deployment controller to basically run our application containers.

Deployment Use - Case

- **Create a Deployment**
  - A deployment is created. Once that is done the RplicaSet automatically creates Pods in the background.
- **Update Deployment**
  - A new ReplicaSet is created and the Deployment is updated. Each new ReplicaSet updates the revision of Deployment.
- **Rollback Deployment**
  - Used in case when the current state of the deployment is not stable. Only the container image gets updated.
- **Scale Deployment**
  - Each and every deployment can be scaled up or scaled down based on the requirement.
- **Pause the Deployment**
  - Pause the deployment to apply multiple fixes and then the deployment can be resumed.

K8S - ReplicaSet

- ReplicaSet ensure how many replica of pod should be running. It can be considered as a replacement of replication controller.



You may never need to manipulate ReplicaSet objects: use a Deployment instead, and define your application in the spec section.

NAYA College

Demo/Labs

K8S replicaSet:

In our repository

$ cd cognyte-devops-32/k8s-2/replicaSet/

Follow README.MD

Demo/Labs

K8S Deployment:

In our repository

$ cd cognyte-devops-32/k8s-2/deployment/

Follow README.MD

- K8S Namespaces -

NAYA College

K8S - Namespaces

Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called **namespaces**.

- Names of resources need to be unique within a namespace, but not across namespaces.
- Namespaces can not be nested inside one another and each Kubernetes resource can only be in one namespace.
- Namespaces are a way to divide cluster resources between multiple users

Initial namespaces

- **default**
  - Kubernetes includes this namespace so that you can start using your new cluster without first creating a namespace.
- **kube-node-lease**
  - This namespace holds Lease objects associated with each node. Node leases allow the kubelet to send heartbeats so that the control plane can detect node failure.
- **kube-public**
  - This namespace is readable by all clients (including those not authenticated). This namespace is mostly reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster. The public aspect of this namespace is only a convention, not a requirement.
- **kube-system**
  - The namespace for objects created by the Kubernetes system.

Demo/Labs

K8S Namespace:

In our repository

$ cd cognyte-devops-32/k8s-1/namespace_plus_pod/

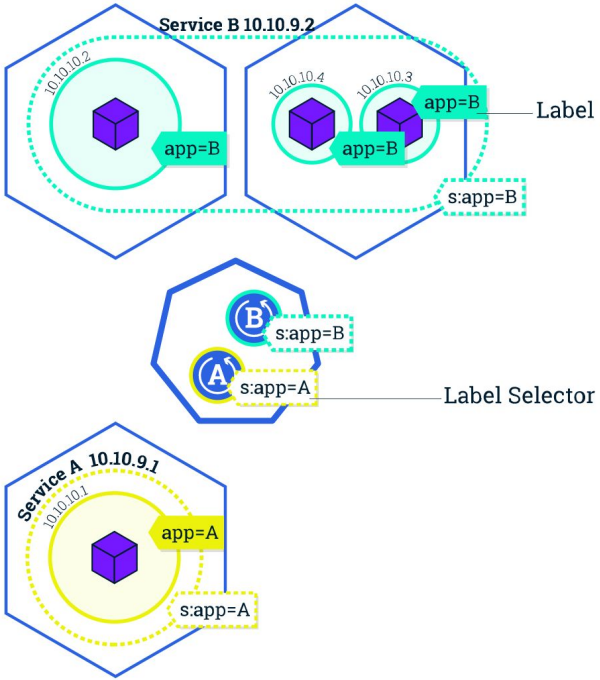
Follow README.MD

- K8S Labels -

Labels and Selectors

How does our newly created service knows how to group PODS as his backend?

Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in K8S. Labels are key/value pairs attached to objects and can be used in any number of ways



Labels and Selectors

Valid label value:

- must be 63 characters or less (can be empty),
- unless empty, must begin and end with an alphanumeric character ([a-z0-9A-Z]),
- could contain dashes (-), underscores (_), dots (.), and alphanumerics between.

Demo/Labs

Let's explore Labels:

In our repository

$ cd cognyte-devops-32/k8s-1/runnig_pod/labels

Follow README.MD

- K8S Annotations -

NAYA College

Annotations

You can use Kubernetes annotations to attach arbitrary non-identifying metadata to objects. Clients such as tools and libraries can retrieve this metadata.

Annotations, like labels, are key/value maps:

```
"metadata": {
  "annotations": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

NAYA College

- K8S Services -

A WINDOW TO THE WORLD

NAYA College

K8S OBJECTS: SERVICES

A **Services** routes traffic across a set of Pods. Services are the **abstraction** that allow pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) is handled by Kubernetes Services.

- K8S SERVICE -

WHY DO WE NEED IT?

NAYA College

K8S OBJECTS: SERVICES

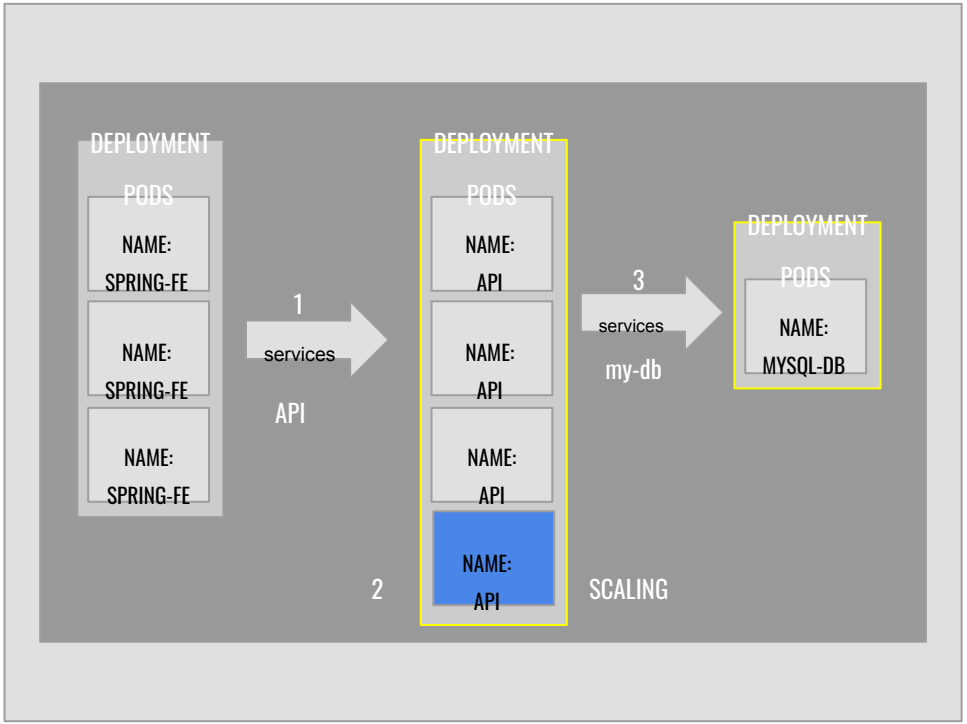
Pods have a short lifetime,

They are born and when they die, they are not resurrected - therefor there is no guarantee about the IP address they are / will served on.

ReplicationControllers in particular create and destroy Pods dynamically (e.g. when scaling out or in or when doing rolling updates). This could make the communication of microservices hard.

Imagine a typical Front End communication with Backend services - how do those frontends find out and keep track of which backends are in that set?

K8S OBJECTS: SERVICES



In the left scenario we need to build a setup in K8S where of SPRING-FE needs to Communicate with the API which in turn writes it's transaction received from the FE to the MYSQL-DB.

Problems:

1. How to tell the FE where are his API endpoints (every POD has its own IP)
   a. We wish to use DNS names
2. What will happen if our API PODS will scale up or down and their IP's will change?
3. API Needs to reach the MYSQL POD with DNS Name

NAYA College

K8S OBJECTS: SERVICES

To solve all of those questions and many more K8s has introduced the concept of a Services

which is an **abstraction on top of a number of pods**, typically requiring to run a proxy on top, for other services to communicate with it via a Virtual IP address. This is where you can configure **load balancing for your numerous pods and expose them via a service.**

K8S OBJECTS: SERVICES

As mentioned before,

Each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow our applications to receive traffic. Services can be exposed in different ways by specifying a type in the ServiceSpec.

The types available for us to use are:

- **ClusterIP** (Default) - Exposes the Service **on an internal IP** in the cluster only
- **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>
- **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 by returning a CNAME record with the name.

NAYA College

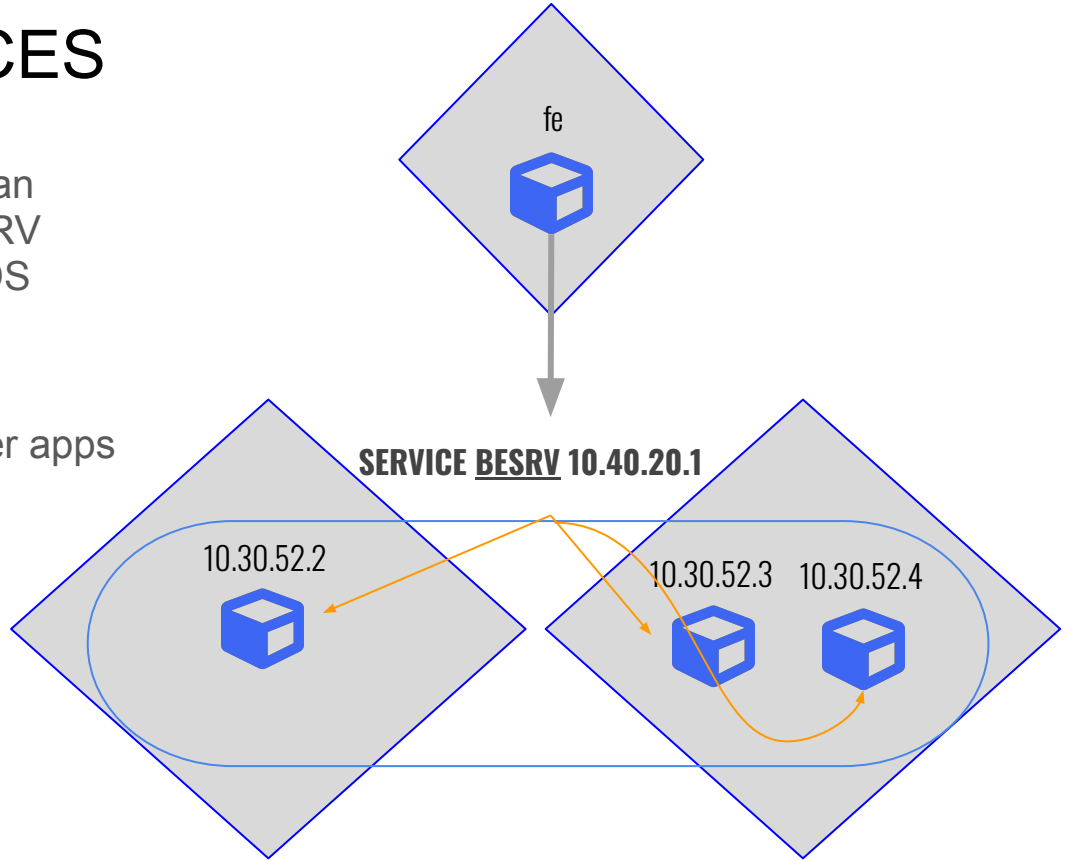
Lets try and understand how each of them works

K8S OBJECTS: SERVICES

ClusterIP - By creating a cluster IP we can allow a client Application to point to BESRV DNS and reach out to all 3 backend PODS under our BESRV ClusterIP Service.

- Service inside your cluster that other apps inside your cluster can access.
- You can access it using k8s proxy:

  $ kubectl proxy --port=8080



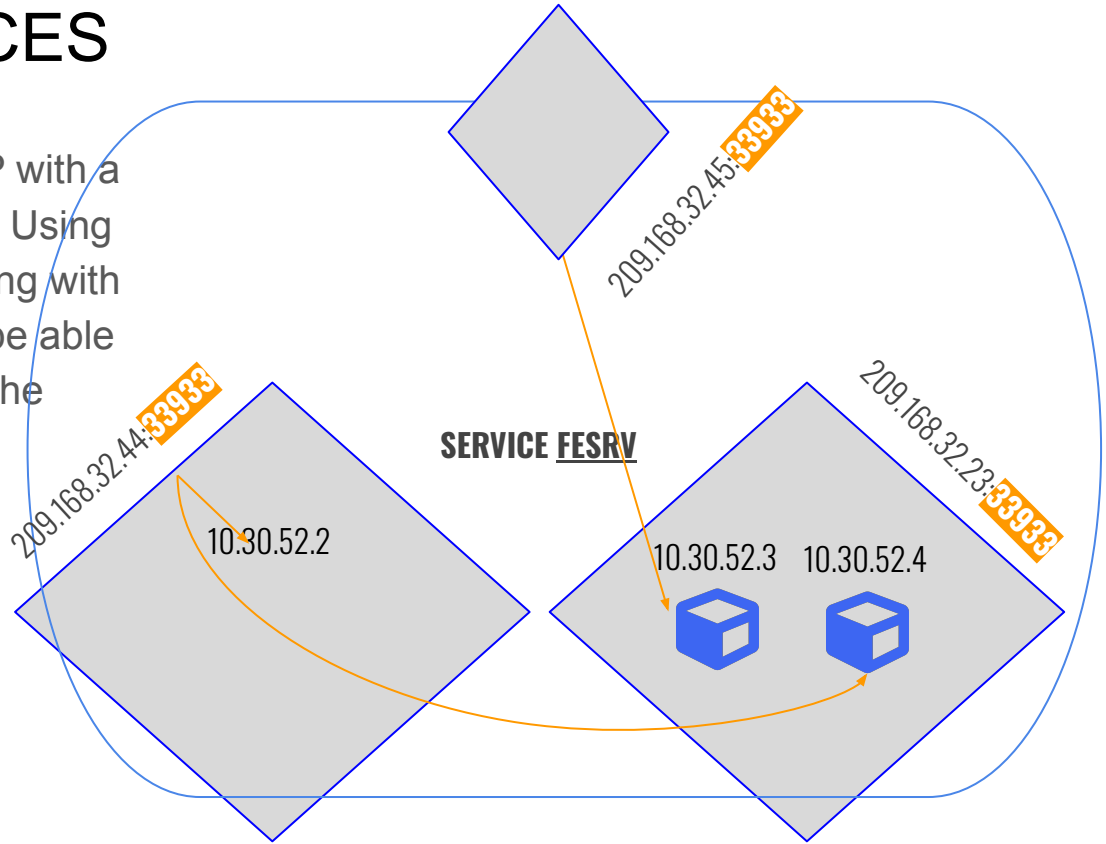
POD

NODE

NAYA College

K8S OBJECTS: SERVICES

NodePort Service creates a VIRTUAL IP with a dedicated Port for the PODS we expose. Using The IP of one or each of our NODES along with the port created by the Services we will be able to reach to our pods from the outside of the cluster.



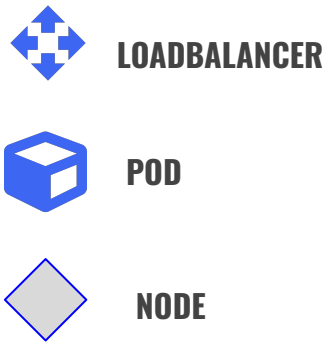
POD

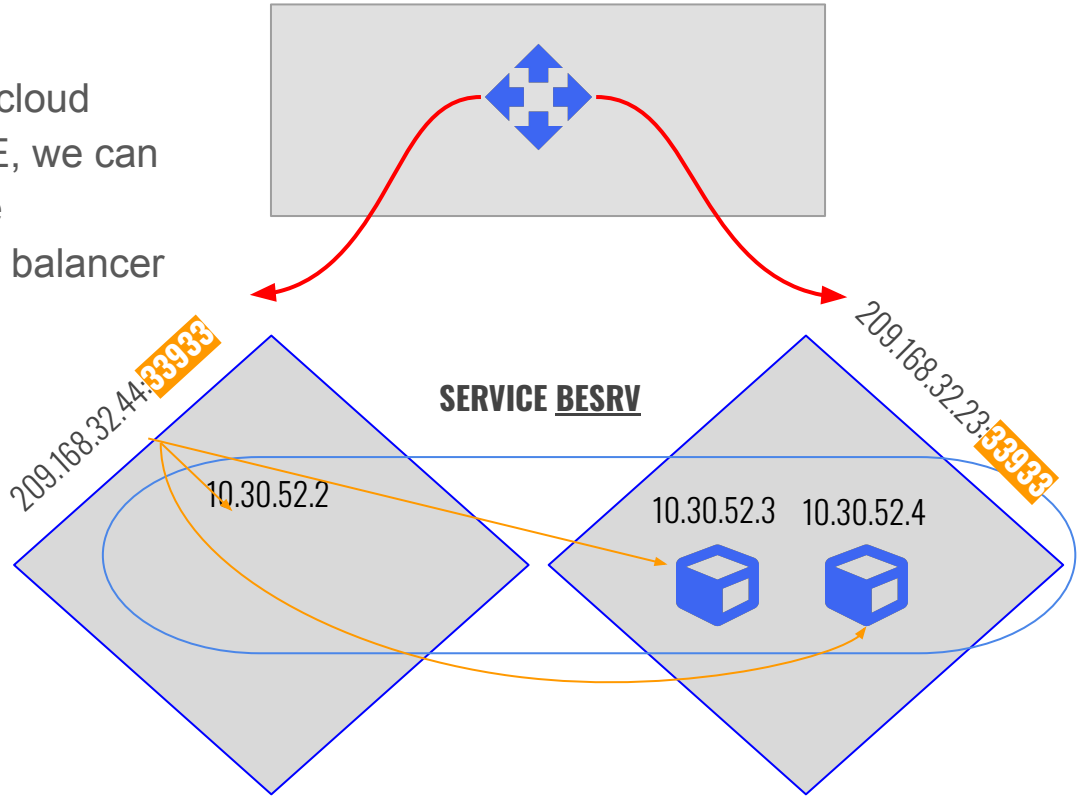
NODE

NAYA College

K8S OBJECTS: SERVICES

LOADBALANCER When hosted on a cloud provider such as AWS, GCP Or AZURE, we can use TYPE LB and Exposes the Service externally using a cloud provider's load balancer





NAYA College

Demo/Labs

service discovery:

In our repository

$ cd cognyte-devops-32/k8s-1/service-discovery

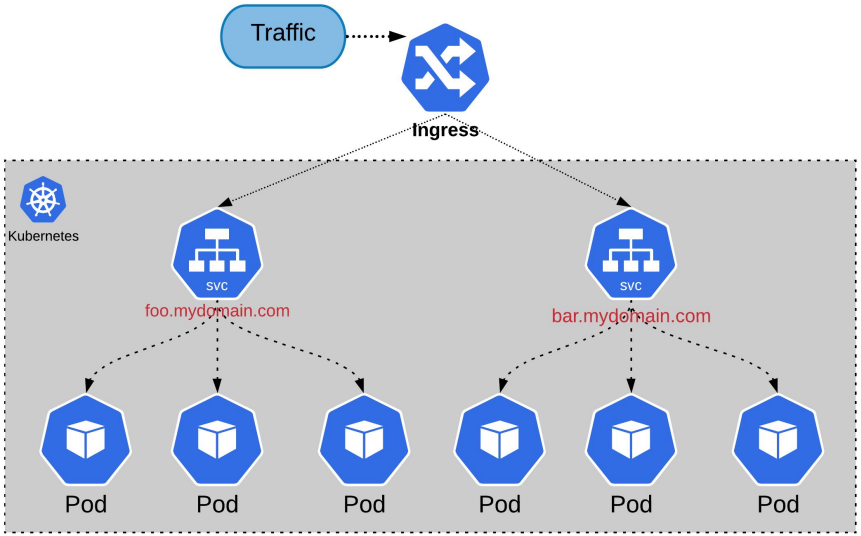
Follow README.MD

- K8S Ingress -

NAYA College

K8S OBJECTS: Ingress

- **Ingress** is a load balancer that will front multiple different pods.
- Traffic is routed via http URI or dns names.
- In Linode Ingress controller will spin up a Node Balancer.



Demo/Labs

Ingress:

In our repository

$ cd cognyte-devops-32/k8s-1/service-discovery/ingress

Follow README.MD or Verbal Instruction.

Q&A

NAYA College