Kubernetes Resources

AGENDA

1. What is YAML & Declerative
2. Names & UID’s
3. Namespace
4. Pod
5. Controllers
6. ReplicaSet Controller
7. Deployment Controller
8. Service
9. Types of Service
10. DeamonSet Controller
11. StatefulSet Controller
12. Job
13. Batch Job

“Declarative” in Kubernetes

One important Kubernetes concept to keep in mind: “Declarative” is an operation based on YAML files.

Kubernetes depends on a YAML file to check if the desired Pod, Deployment, or Service is running as defined.

We can perform kubectl apply a hundred times with no undesirable or unexpected outcome.

Kubernetes just checks if the system is running according to the desired state as defined in the YAML file

Kubernetes Resources

Kubernetes objects are the entries in the cluster, which are stored in etcd.

They represent the desired state of your cluster. When we create an object, we send the request to the API server by kubectl or a RESTful API.

The API server will check whether the request is valid, store the state in etcd, and interact with other master components to ensure the object exists

Names & UIDs

Names: All objects in the Kubernetes REST API are unambiguously identified by a Name and a UID.
Only one object of a given kind can have a given name at a time. However, if you delete the object, you can make a new object with the same name.

By convention, the names of Kubernetes resources should be up to maximum length of 253 characters and consist of lower case alphanumeric characters

UIDs: A Kubernetes systems-generated string to uniquely identify objects.

Every object created over the whole lifetime of a Kubernetes cluster has a distinct UID. It is intended to distinguish between historical occurrences of similar entities.

Namespaces

Kubernetes namespaces allow us to implement isolation of multiple virtual clusters. Objects in different namespaces are invisible to each other.

This is useful when different teams or projects share the same cluster.

Kubernetes has three namespaces:

default
kube-system
kube-public

If we don't explicitly assign a namespace to a namespaced resource, it'll be located in the namespace of the current context. If we never add a new namespace, a default namespace will be used.

Namespaces

Kube-system namespaces are used by objects created by the Kubernetes system, such as addon, which are the pods or services that implement cluster features

Best Practices:

Kubernetes Object Spec

The object ***spec*** describes the desired state of Kubernetes objects. Most of the time, we write an object ***spec*** and send it to the API server via kubectl.

Kubernetes will try to fulfill that desired state and update the object's status.

The object spec could be written in YAML (http://www.yaml.org/) or JSON (http://www.json.org/).

YAML is more common in the Kubernetes world.

We'll use YAML to write object spec.

Kubernetes Objects

The following code block shows a YAML-formatted spec fragment:

```
apiVersion: Kubernetes API version
kind: object type
metadata:
  spec metadata, i.e. namespace, name, labels and annotations
spec:
  the spec of Kubernetes object
```

Kubernetes 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.

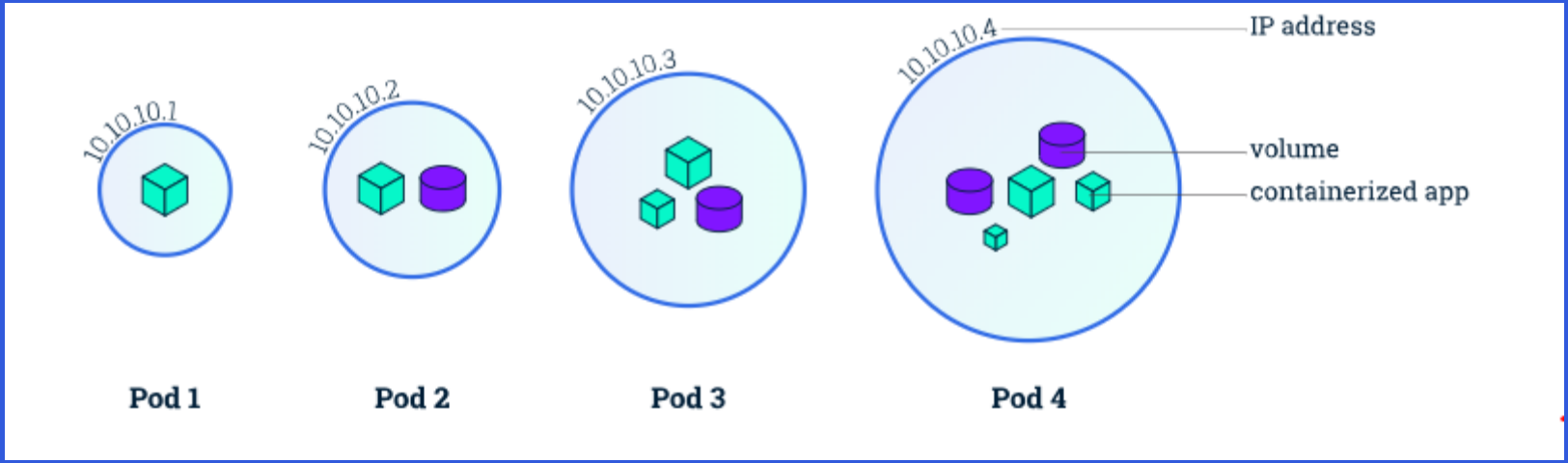
- A Pod encapsulates an application container (or, in some cases, multiple containers), storage resources, a unique network IP, and options that govern how the container(s) should run.

- A Pod represents a unit of deployment: a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources.

Kubernetes Pod

- Group of one or more containers that are always co-located, co-scheduled, and run in a shared context
- Containers in the same pod have the same hostname
- Each pod is isolated by
    - Process ID (PID) namespace
    - Network namespace
    - Inter Process Communication (IPC) namespace
    - Unix Time Sharing (UTS) namespace
- Alternative to a VM with multiple processes

Kubernetes Pod



Labels & Selectors

- Key/value pairs associated with Kubernetes objects
- Used to organize and select subsets of objects
- Attached to objects at creation time but modified at any time.
- Labels are the essential glue to associate one API object with other
    - Replication Controller -> Pods
    - Service -> Pods
    - Pods -> Nodes

Kubernetes Controller

A Controller can create and manage multiple Pods for you, handling replication and rollout and providing self-healing capabilities at cluster scope.

For example, if a Node fails, the Controller might automatically replace the Pod by scheduling an identical replacement on a different Node.

ReplicaSet
Deployment
StatefulSet
DeamonSet
Job
Batch Job

Kubernetes ReplicaSet

A ReplicaSet is responsible for a group of identical Pods, or replicas. If there are too few (or too many) Pods, compared to the specification, the ReplicaSet controller will start (or stop) some Pods to rectify the situation.

Kubernetes Services

Running multiple instances of an application will require a way to distribute the traffic to all of them.

A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods.

Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment.

Each Pod and its corresponding ports will be mapped to this Service definition

Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.

Kubernetes Service

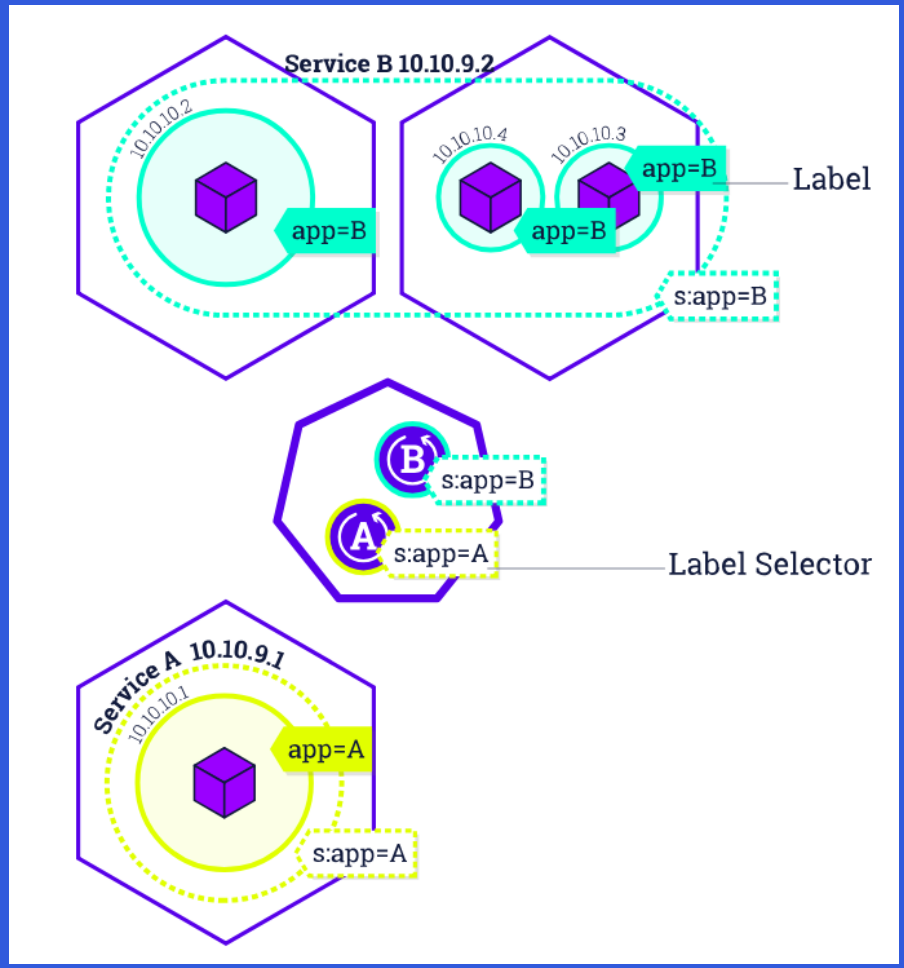
Services match a set of Pods using labels and selectors, a grouping primitive that allows logical operation on objects in Kubernetes.

Using selectors

```
kind: Service
metadata:
   name: frontend
spec:
   type: NodePort
   selector:
      app: nginx
   ports:
     - port: 80
       targetPort: 80
       nodePort: 30002
```

matches

```
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```



Kubernetes Services

Services can be exposed in different ways by specifying a type in the ServiceSpec:

ClusterIP (default) - Exposes the Service on an internal IP in the cluster. This type makes the Service only reachable from within the cluster.

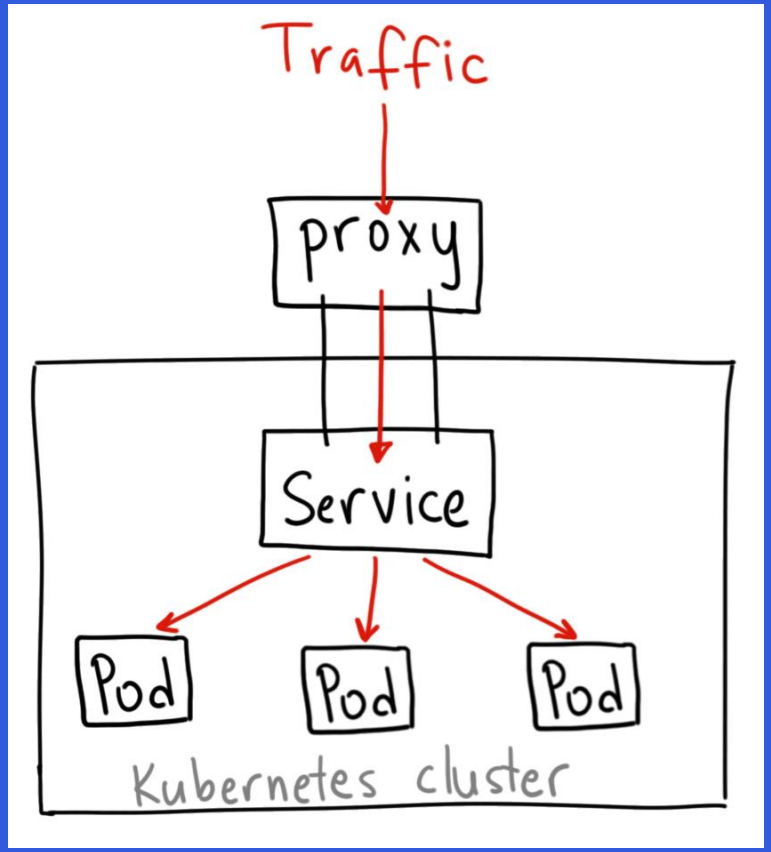
NodePort - Exposes the Service on the same port of each selected Node in the cluster using NAT. Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>. Superset of ClusterIP.

LoadBalancer - Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP to the Service. Superset of NodePort.

ClusterIP Service

A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.

If you can't access a ClusterIP service from the internet, why am I talking about it? Turns out you can access it using the Kubernetes proxy!



When would you use this?

There are a few scenarios where you would use the Kubernetes proxy to access your services.

1. Debugging your services, or connecting to them directly from your laptop for some reason
2. Allowing internal traffic, displaying internal dashboards, etc.
3. Allow your services to be accessed only within cluster, not outside

Because this method requires you to run kubectl as an authenticated user, you should NOT use this to expose your service to the internet or use it for production services.

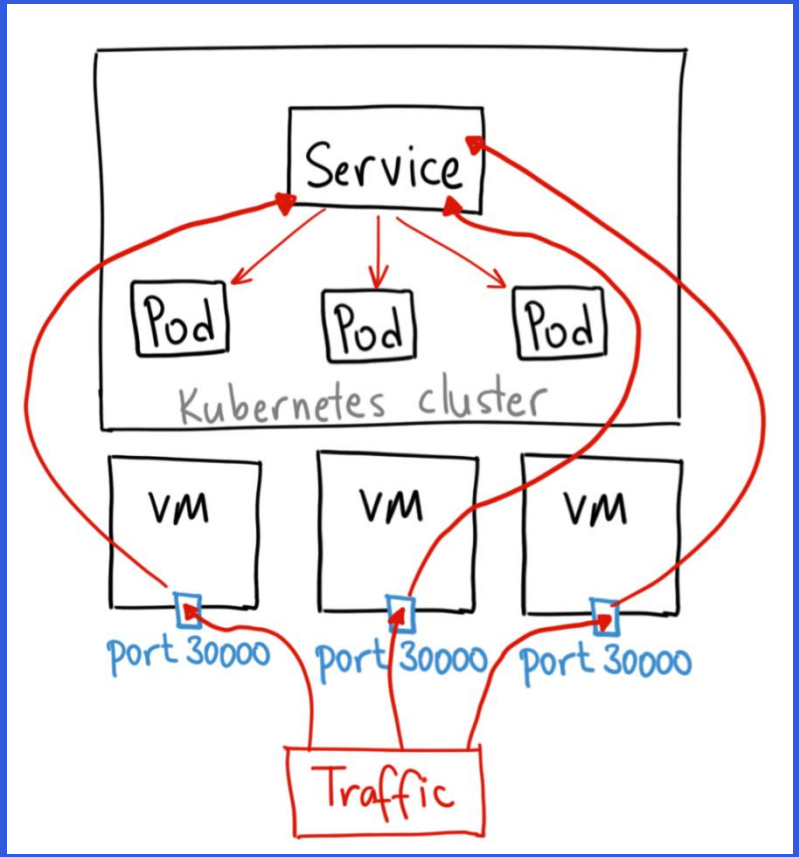
NodePort Service

A NodePort service is the most primitive way to get external traffic directly to your service.

NodePort, as the name implies, opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service.

Basically, a NodePort service has two differences from a normal “ClusterIP” service.

1. the type is “NodePort.”
2. Has an additional port called nodePort



When would you use this?

There are many downsides to this method:

1. You can only have once service per port
2. You can only use ports 30000–32767

If your Node/VM IP address change, you need to deal with that
For these reasons, I don’t recommend using this method in production to directly expose your service.

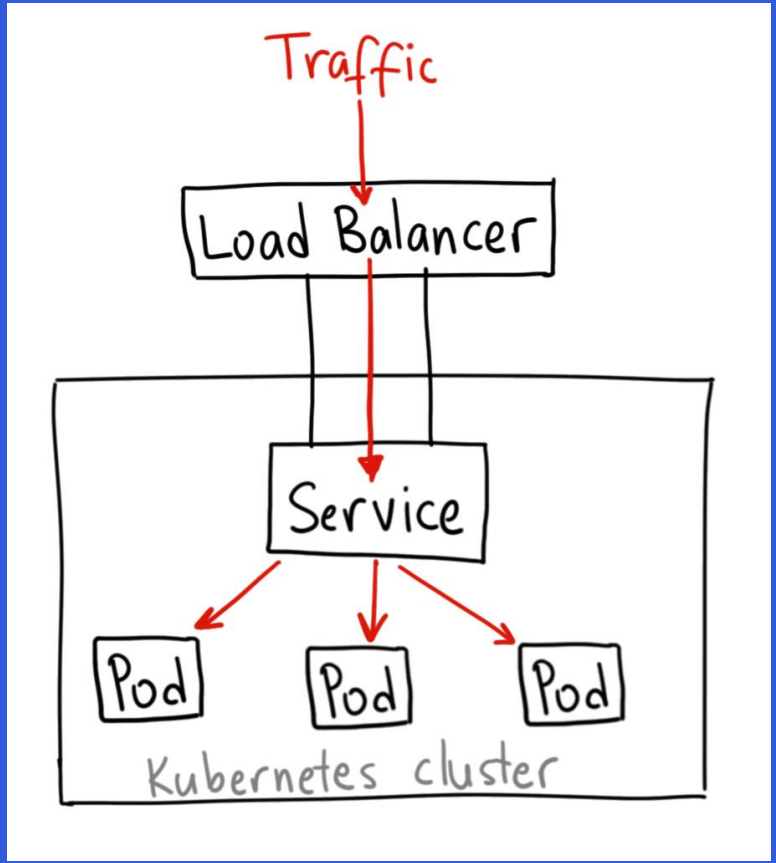
If you are running a service that doesn’t have to be always available, or you are very cost sensitive, this method will work for you. A good example of such an application is a demo app or something temporary.

LoadBalancer

A LoadBalancer service is the standard way to expose a service to the internet.

On GKE, this will spin up a Network Load Balancer that will give you a single IP address that will forward all traffic to your service.

This service can only be utilized within a Cloud environment.



When would you use this?

If you want to directly expose a service, this is the default method.

All traffic on the port you specify will be forwarded to the service. There is no filtering, no routing, etc. This means you can send almost any kind of traffic to it, like HTTP, TCP, UDP, Websockets, gRPC, or whatever.

The big downside is that each service you expose with a LoadBalancer will get its own IP address, and you have to pay for a LoadBalancer per exposed service, which can get expensive!

Kubernetes StatefulSet

The

https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/

Kubernetes NodeAffinity, Pod Affinity

The

https://supergiant.io/blog/learn-how-to-assign-pods-to-nodes-in-kubernetes-using-nodeselector-and-affinity-features/

Kubernetes

The

Kubernetes Pod

You'll rarely create individual Pods directly in Kubernetes–even singleton Pods.

Pods do not, by themselves, self-heal. If a Pod is scheduled to a Node that fails, or if the scheduling operation itself fails, the Pod is deleted; likewise, a Pod won't survive an eviction due to a lack of resources or Node maintenance.

Docker is the most common container runtime used in a Kubernetes Pod, but Pods support other container runtimes as well.

Kubernetes uses a higher-level abstraction, called a Controller

Kubernetes

The

Kubernetes

The

Kubernetes

The

Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler automatically scales the number of pods in a replication controller, deployment or replica set based on observed CPU utilization (or, with custom metrics support, on some other application-provided metrics).

Note that Horizontal Pod Autoscaling does not apply to objects that can't be scaled, for example, DaemonSets.

The Horizontal Pod Autoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The controller periodically adjusts the number of replicas in a replication controller or deployment to match the observed average CPU utilization to the target specified by user.

Kubernetes

The

Q & A