Kubernetes - Introduction

What is Micro Service ?

Micro services Features

Independent Modules

Each module can be written in different language

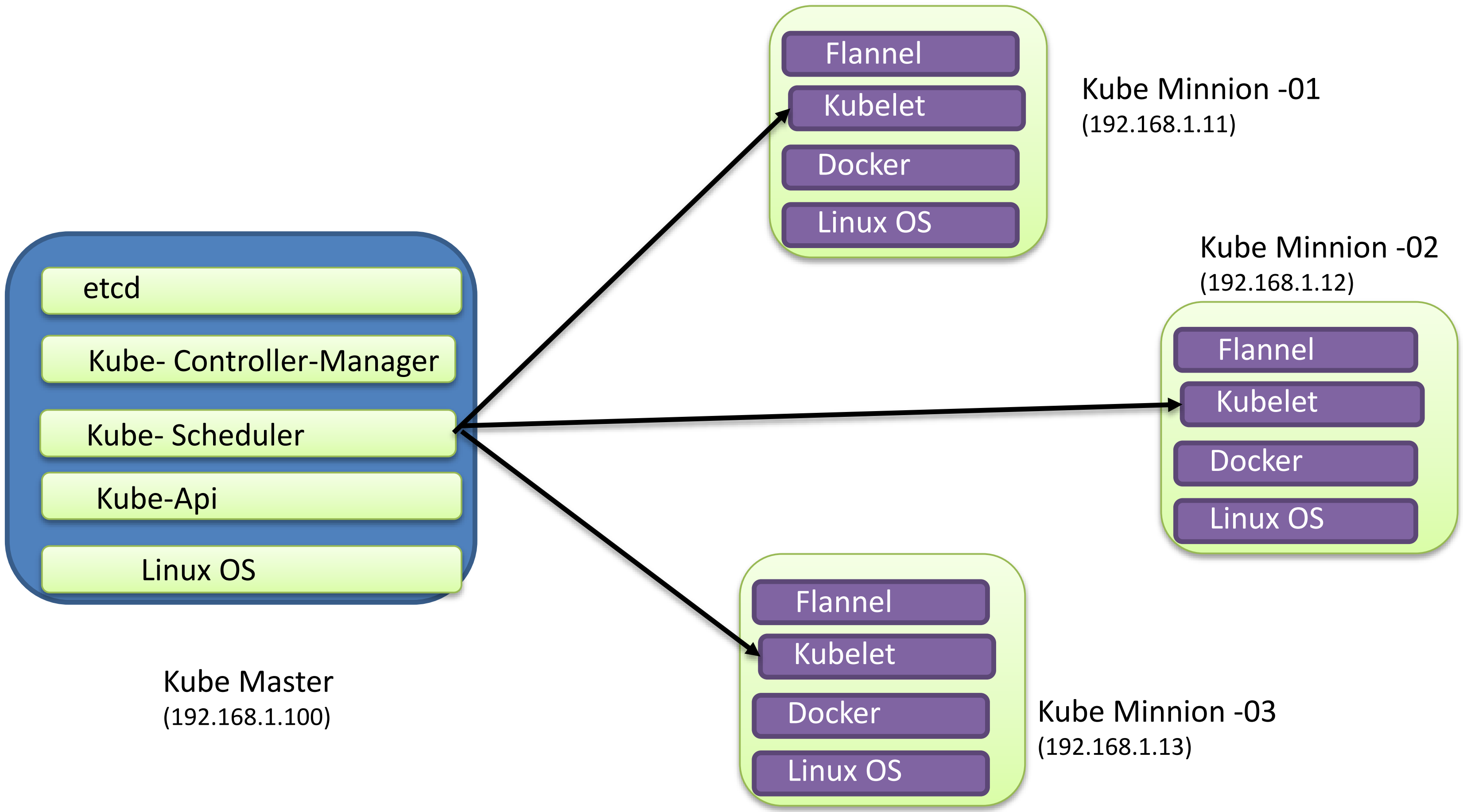
Micro service expose themselves via restful API's

Interact with each other via Messaging bus (ex rabbitmq)

Decoupled

Issue in one module does not impact other modules functioning

Fix issue related one module and not all


Flannel
Kubelet
Docker
Linux OS
Kube Minnion -01
(192.168.1.11)
Kube Minnion -02
(192.168.1.12)
Flannel
Kubelet
Docker
Linux OS
etcd
Kube- Controller-Manager
Kube- Scheduler
Kube-Api
Linux OS
Kube Master
(192.168.1.100)
Flannel
Kubelet
Docker
Linux OS
Kube Minnion -03
(192.168.1.13)

Master Node Architecture

Etcd: It is an open source key-value store developed by CoreOs team. Kubernetes uses ‘Etcd’ to store the configuration data accessed by all nodes (minions and master) in the cluster.

Kube-ApiServer: The Kubernetes api-server generally validates the configuration data store in ‘Etcd’ and the details of the deployed container that are in agreement. It also provides a RESTful interface to make communication easy.

Kube-Schedule Server: It is responsible for assigning task to minions in the cluster.

Kube-Controller-Manager: It is generally responsible for handling the cluster level function such as replication controller. Whenever the desired state of the cluster changes it is written to Etcd and then the controller manager tries to bring up the cluster in the desired state.

Master Node Architecture

- A single master host will manage the cluster and run several core Kubernetes services.

- **API Server -** The REST API endpoint for managing most aspects of the Kubernetes cluster.
- **Replication Controller** - Ensures the number of specified pod replicas are always running by starting or shutting down pods.
- **Scheduler** - Finds a suitable host where new pods will be reside.
- **etcd -** A distributed key value store where Kubernetes stores information about itself, pods, services, etc.
- **Flannel** - A network overlay that will allow containers to communicate across multiple hosts.

Minion Node Architecture

- The minion hosts will run the following services to manage containers and their network.

- **Kubelet -**
    - Host level pod management
    - It is an agent process that runs on each node.
    - It is responsible for managing pods and their containers.
    - It deal with pods specifications which are defined in YAML or JSON format.
    - Kubelet takes the pod specifications and checks whether the pods are running healthy or not.

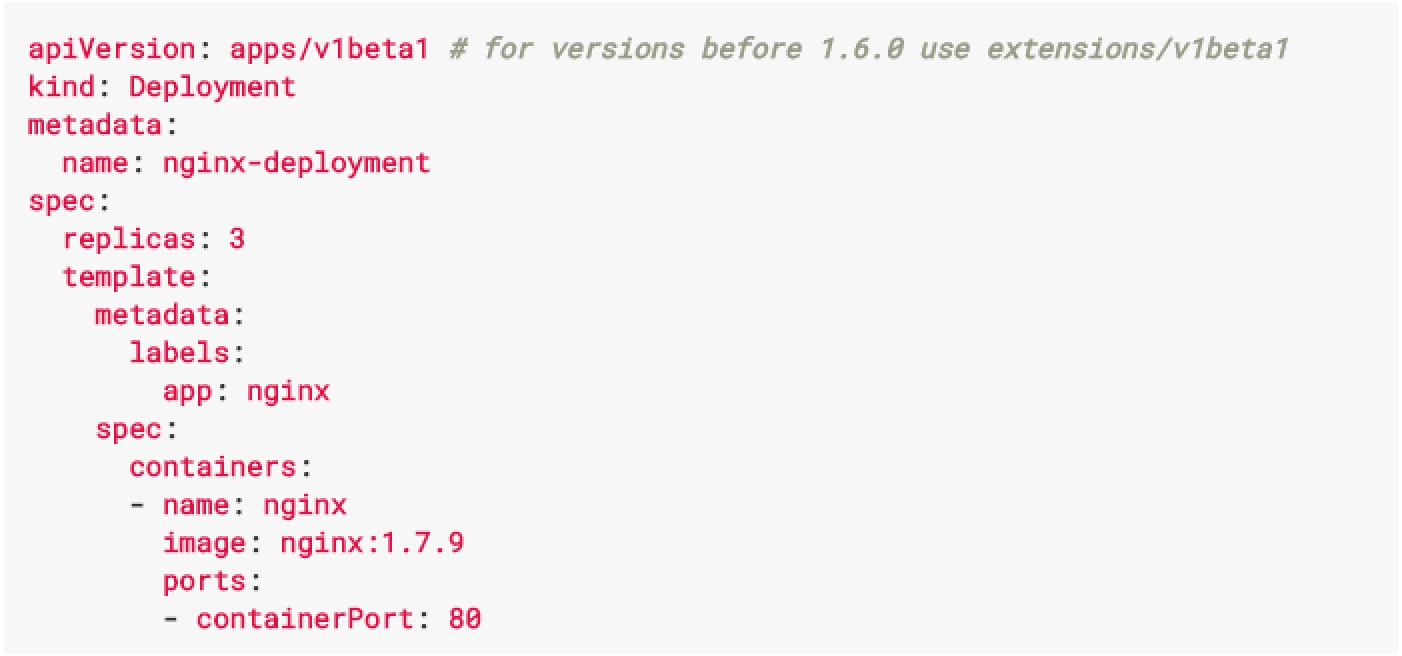
- **Proxy –**
    - Every node in the cluster runs a simple network proxy. Using proxy node in cluster routes request to the correct container in a node.

    - Manages the container network (IP addresses and ports) based on the network service manifests received from the Kubernetes master.

- **Docker -** An API and framework built around Linux Containers (LXC) that allows for the easy management of containers and their images.

- **Flannel** - A network overlay that will allow containers to communicate across multiple hosts.

Example yaml



```yaml
apiVersion: apps/v1beta1 # for versions before 1.6.0 use extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

In this example

- A Deployment named nginx-deployment is created, indicated by the metadata: name field.

- The Deployment creates **three** replicated Pods, indicated by the replicas field.

- The Pod template's specification, or template:

- **spec** field, indicates that the Pods run one container, *nginx*, which runs the nginx Docker Hub image at version 1.7.9.

- The Deployment opens port 80 for use by the Pods.

- The **template** field contains the following instructions:
  - The Pods are labeled **app: nginx**
  - Create one container and name it **nginx.**
  - Run the nginx image at **version 1.7.9**.
  - **Open port 80** so that the container can send and accept traffic.

Namespaces

Overview

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

When to Use Multiple Namespaces

Namespaces are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to create or think about namespaces at all. Start using namespaces when you need the features they provide.

Namespaces are a way to divide cluster resources between multiple users (via resource quota).

Kubernetes starts with three initial namespaces:

default The default namespace for objects with no other namespace
kube-system The namespace for objects created by the Kubernetes system
kube-public The namespace is created automatically and readable by all users (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

Resource quotas

Namespaces are a way to divide cluster resources between multiple users (via resource quota).

Resource quotas

When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources.
Resource quotas are a tool for administrators to address this concern

Examples of policies that could be created using namespaces and quotas are:
In a cluster with a capacity of 32 GiB RAM, and 16 cores, let team A use 20 GiB and 10 cores, let B use 10GiB and 4 cores, and hold 2GiB and 2 cores in reserve for future allocation.

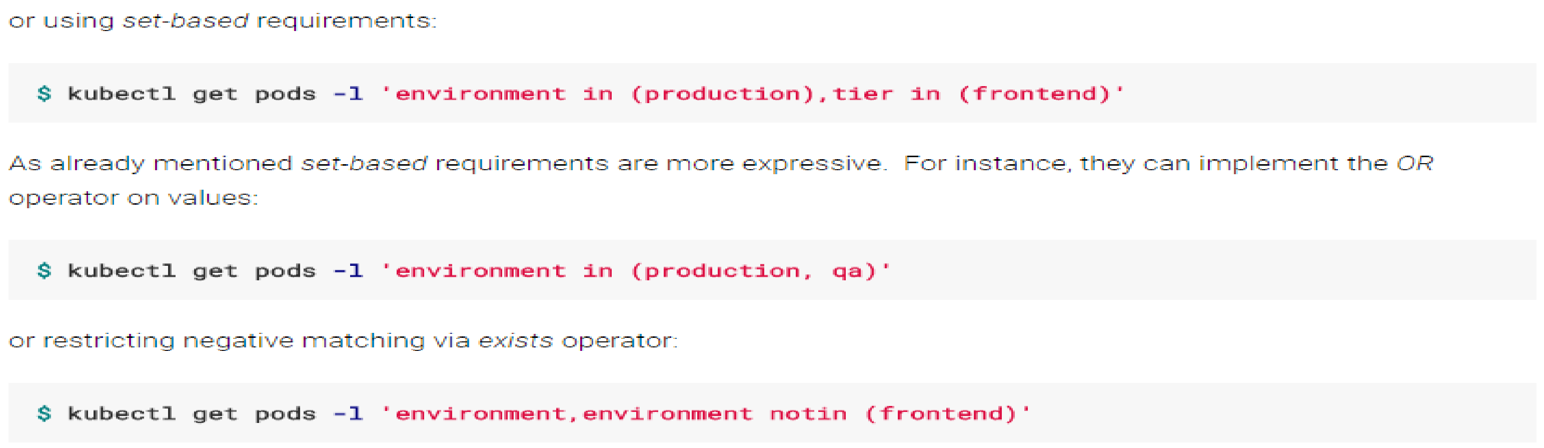
https://kubernetes.io/docs/tasks/administer-cluster/quota-api-object/#before-you-begin

Label Selectors

Unlike names and UIDs, labels do not provide uniqueness. In general, we expect many objects to carry the same label(s).

Equality-based requirement
Equality- or *inequality-based* requirements allow filtering by label keys and values

Three kinds of operators are admitted =,==,!=



or using *set-based* requirements:

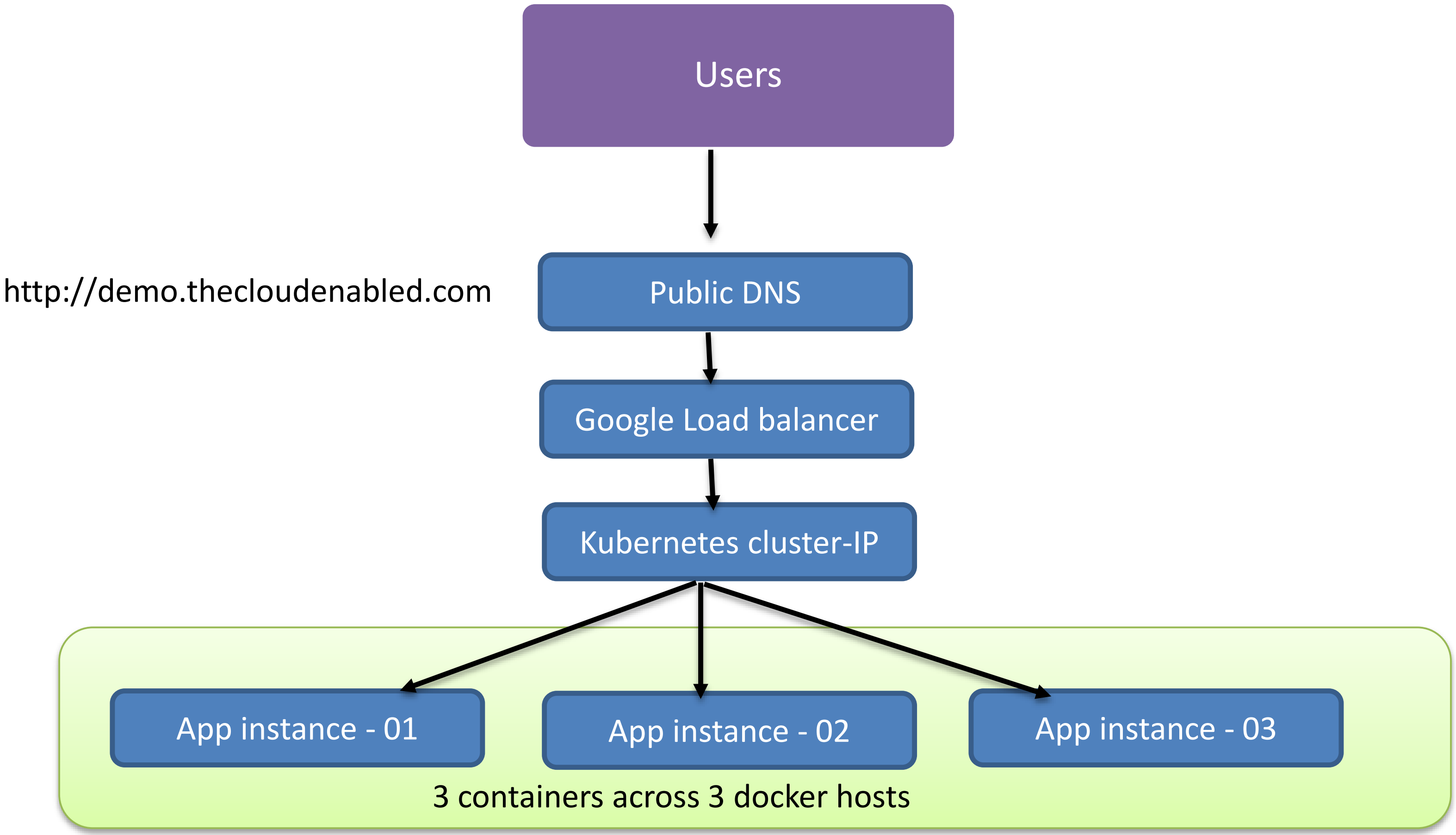
```
$ kubectl get pods -l 'environment in (production),tier in (frontend)'
```

As already mentioned *set-based* requirements are more expressive. For instance, they can implement the *OR* operator on values:

```
$ kubectl get pods -l 'environment in (production, qa)'
```

or restricting negative matching via *exists* operator:

```
$ kubectl get pods -l 'environment,environment notin (frontend)'
```


Users
http://demo.thecloudenabled.com
Public DNS
Google Load balancer
Kubernetes cluster-IP
App instance - 01
App instance - 02
App instance - 03
3 containers across 3 docker hosts

Live Demo

Thank You