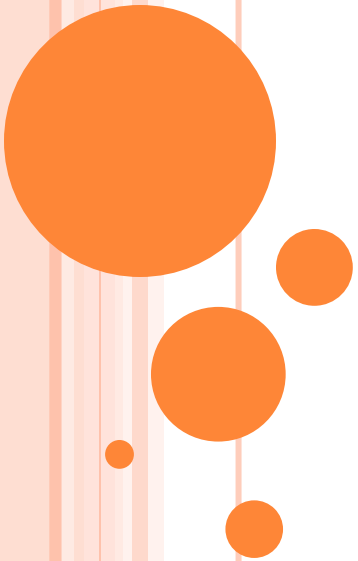
KUBERNETES



OBJECTIVES

- Introduction to Kubernetes
- Kubernetes Architecture
- Pods in Kubernetes
- Controllers
- Services
- Demo

Introduction to Kubernetes

- What is Kubernetes?

  Kubernetes is an open-source system for automating deployment, scaling the management of containerized applications.

  Why Kubernetes comes into the picture?

  Issues with container:

  - Scaling
  - Communication between containers
  - Traffic distribution

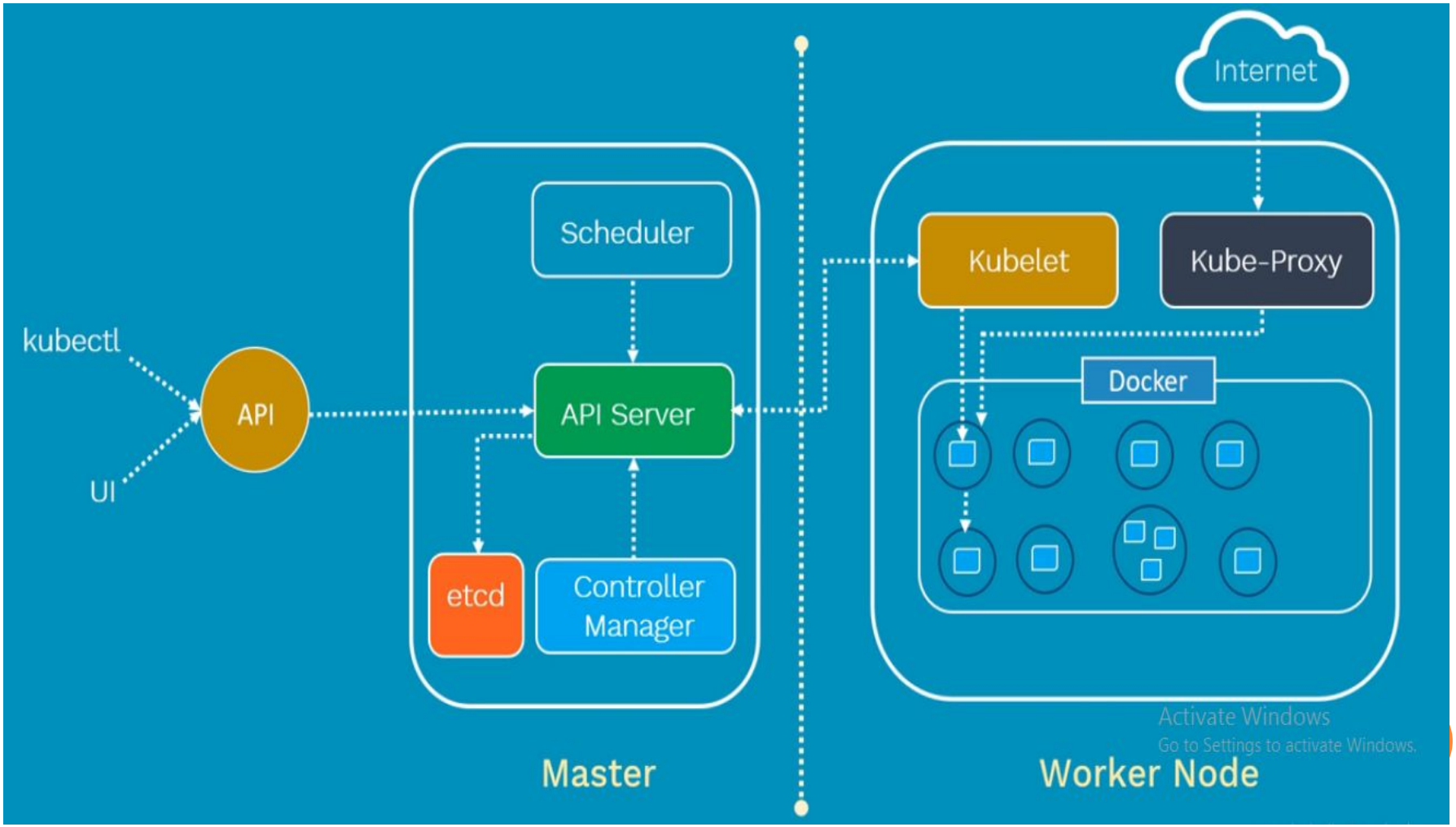
Introduction to Kubernetes

- Features:
  - Self Healing
  - Load Balancing
  - Scaling and Descaling
  - Rollback and Rollout

Kubernetes Architecture

- Key Components
  - Master
    - API Server
    - Scheduler
    - Control Manager
    - etcd
  - Worker Node
    - Kubelet
    - Kube-Proxy
    - Container

Kubernetes Architecture



Kubernetes Architecture

- Master
  - Responsibilities
    - Manage entire cluster
    - Coordinate all activities
    - Communication with worker node
  - Master Components
    - API Server
      - Act as a gatekeeper
      - Object create, update, display, delete
      - API object validation and configuration
      - Expose operation API

Kubernetes Architecture

- Master Components
  - Scheduler
    - Responsible for physical scheduling of pods across multiple nodes
    - Scheduling of pods by constraints mentioned in the configuration file
  - Control Manager
    - Node, Replication, Endpoint and Service Account Controller
    - All these controllers are responsible for the overall health of entire cluster
    - They ensure
      - Nodes are running and up
      - Correct no of pods are running as mentioned in the configuration file

Kubernetes Architecture

- Master Components
  - etcd
    - Distributed key-value lightweight DB
    - Central DB to store current cluster state at any point of time
    - All configuration information about cluster states is stored in the etcd in the form of key/value pairs.
    - Any component of Kubernetes can query to etcd to understand the state of the cluster

Kubernetes Architecture

- Worker Node Components
  - Kubelet
    - Prime node agent runs on each WN
    - Periodically checks the health of the containers in a pod
    - Looks into the pod spec that submitted to the API server on KM and ensures that containers described in that pods spec are running and healthy
    - If found any issues with running pods then it will restart the pod in the same node

Kubernetes Architecture

- Worker Node Components
  - Kube - Proxy
    - Responsible for maintaining the internet network configuration
    - Runs in each node for load distribution among the pods and makes services available to the external host
    - It uses iptable rules or round robin to forward requests to the correct containers
  - Pod and Container
    - A pod is a deployment unit in the K8S with a single IP address
    - Act as a wrapper around containers
    - Container provide run-time environment for an application

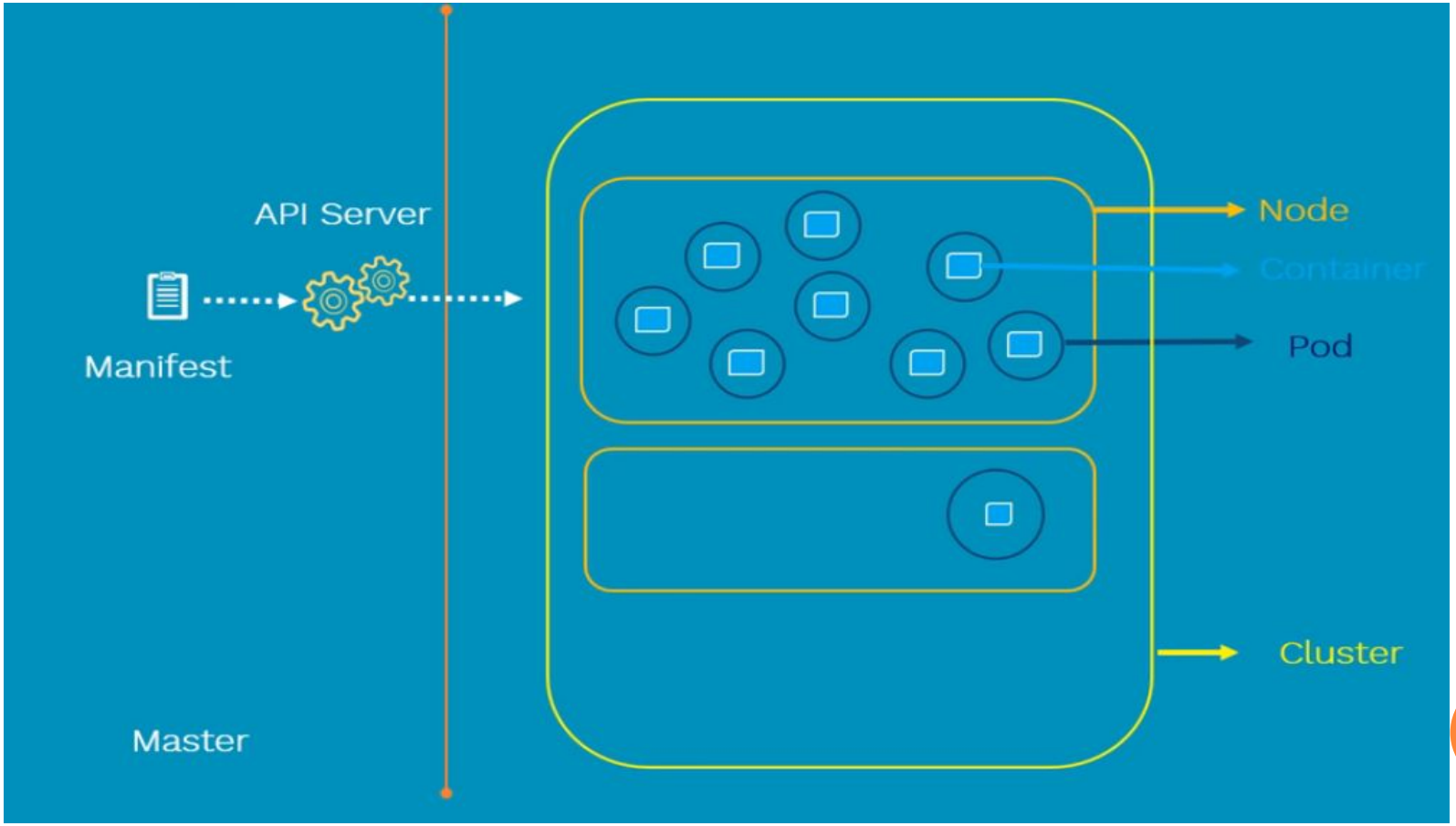
Pods

- A pod is a deployment unit in the K8S with a single IP address

- Pod Deployment
  - Write manifest file
  - Manifest file consist of container images
  - Submit manifest file to API server
  - API server and scheduler will decide and deploy these pods on worker node

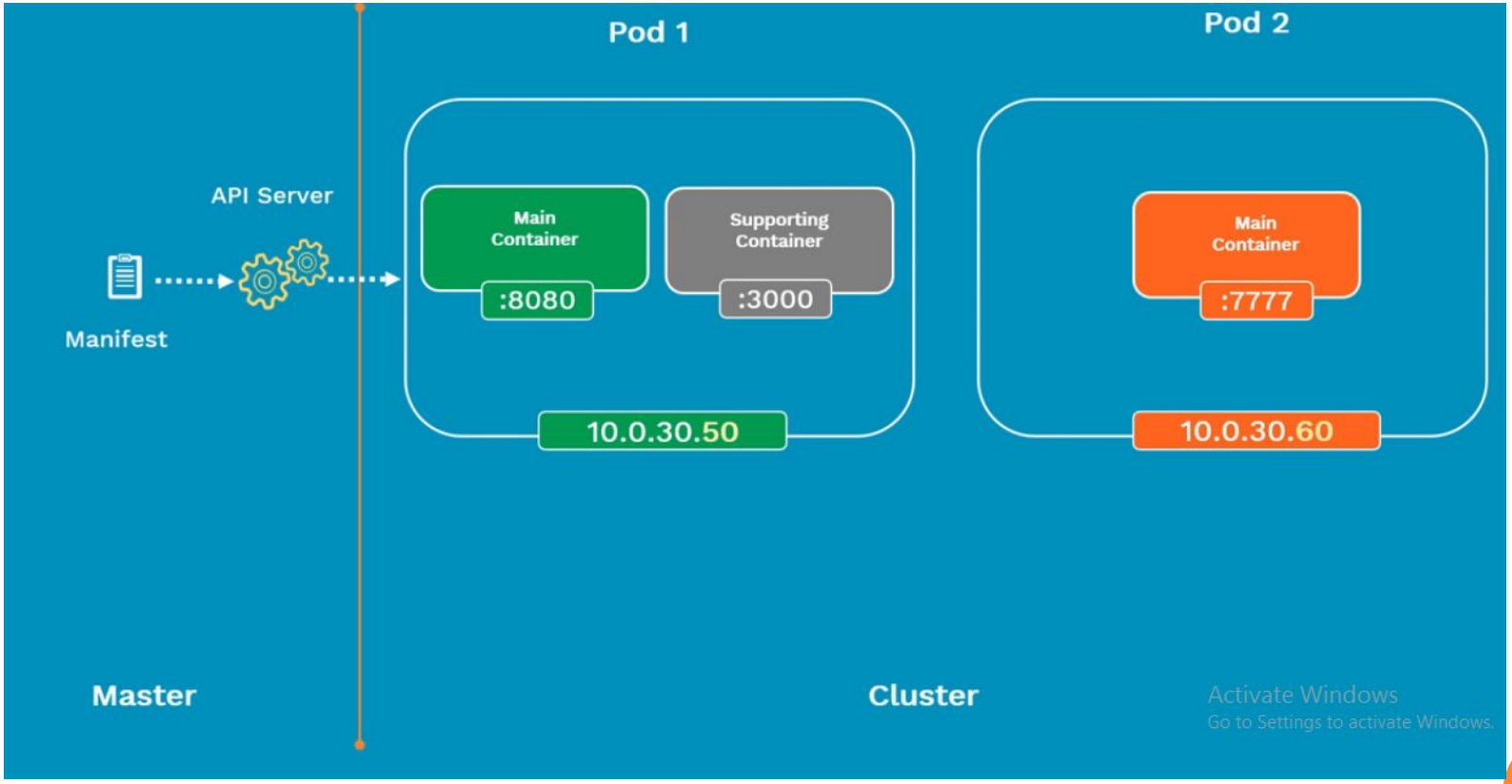
Pods

- Pod Deployment



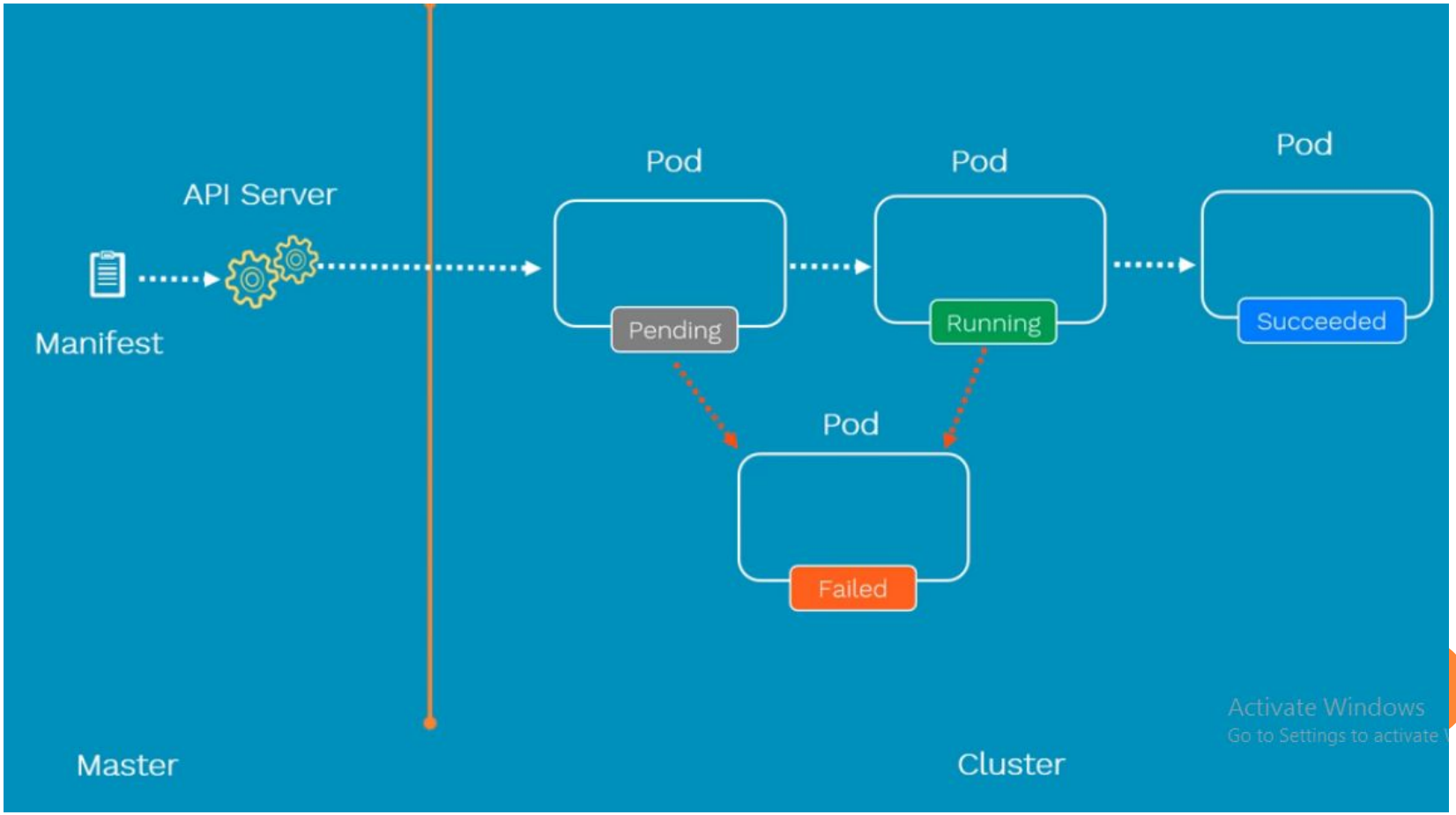
Pods

- Pod Networking



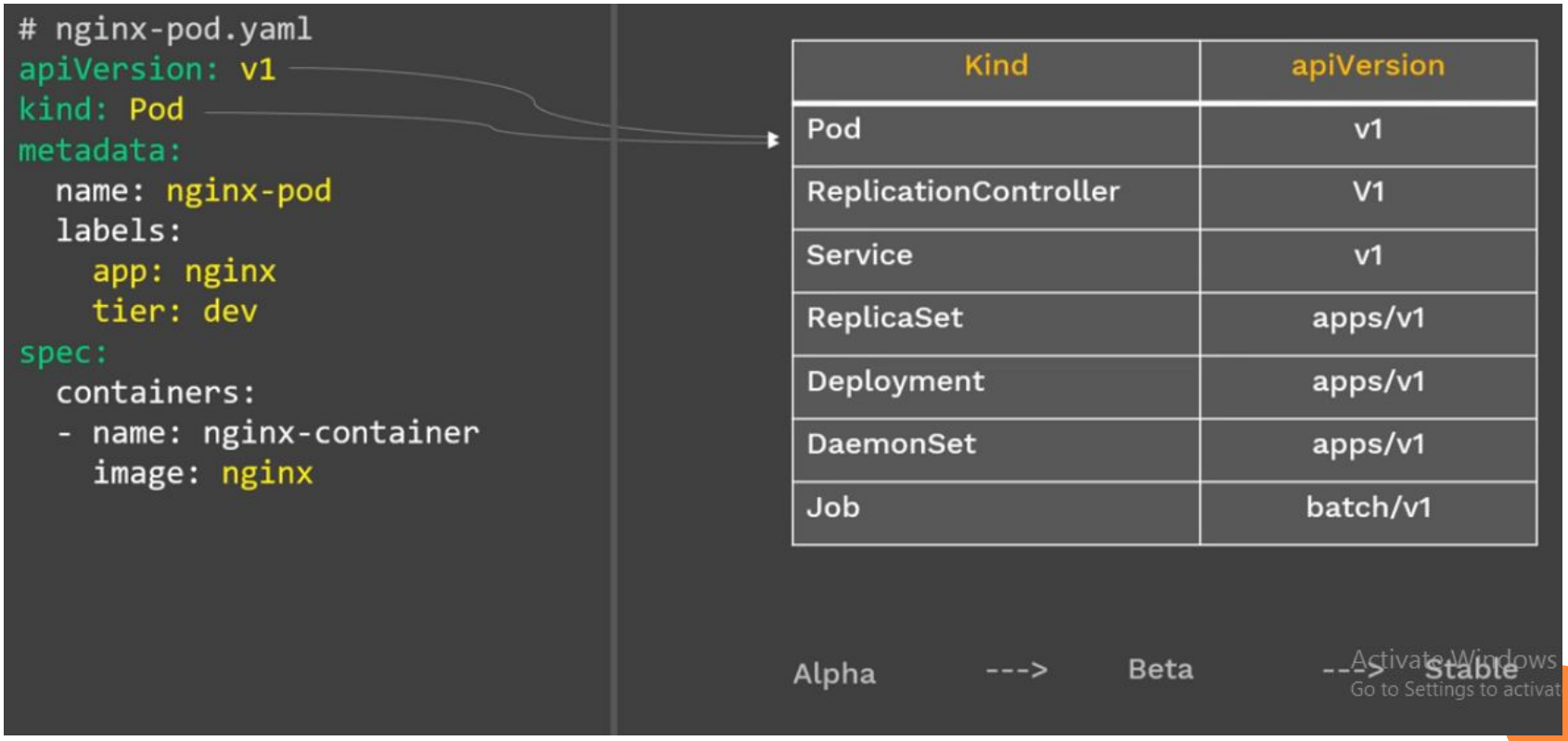
Pods

- Pod Lifecycle



Pods

- Pod Config



Activate Windows
Go to Settings to activat

Replication controller

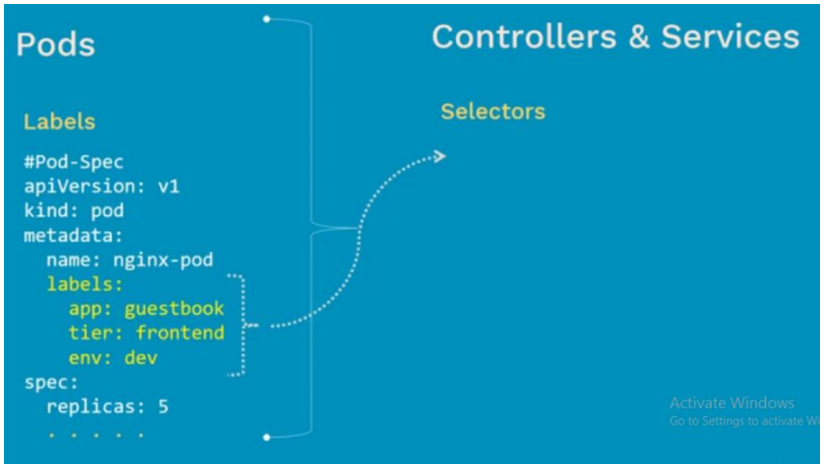
- Ensures that a specified number of pods are running at any time
  - If there are excess Pods, they get killed and vice versa
  - New pods are launched when they get failed, get deleted or terminated
- Replication controller and Pods are associated with “labels”
- Creating a rc with count of 1 ensures that a pod is always available

ReplicaSet

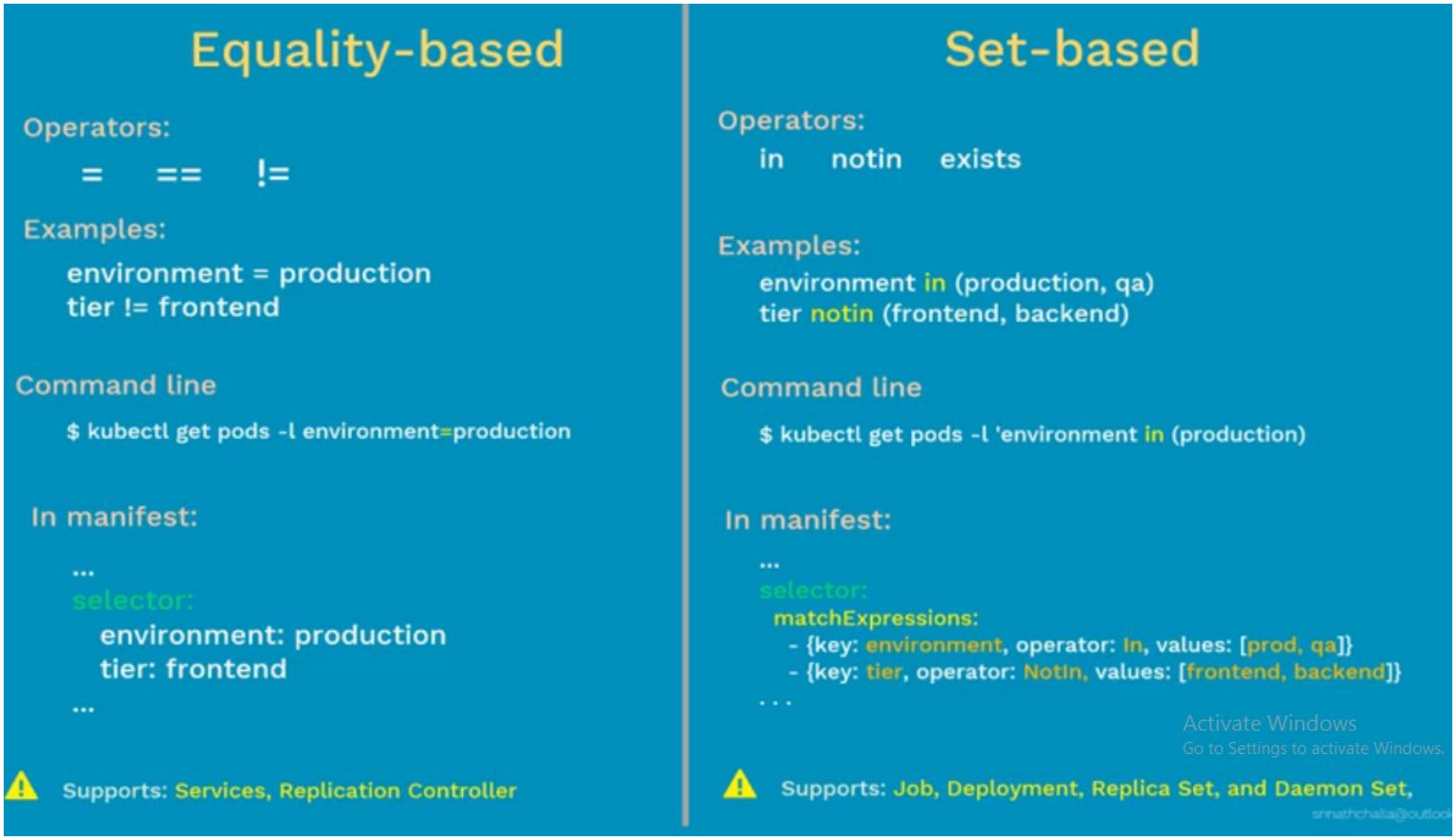
- Ensures that a specified number of pods are running at any time
  - If there are excess Pods, they get killed and vice versa
  - New pods are launched when they get failed, get deleted or terminated
- Replication controller and Pods are associated with “labels”

Replication controller vs Replica set

- ReplicaSet is a Next-generation Replication Controller
- Selectors
  - ReplicaSet
    - Supports Set-based selectors
  - Replication Controller
    - Supports equality-based selectors

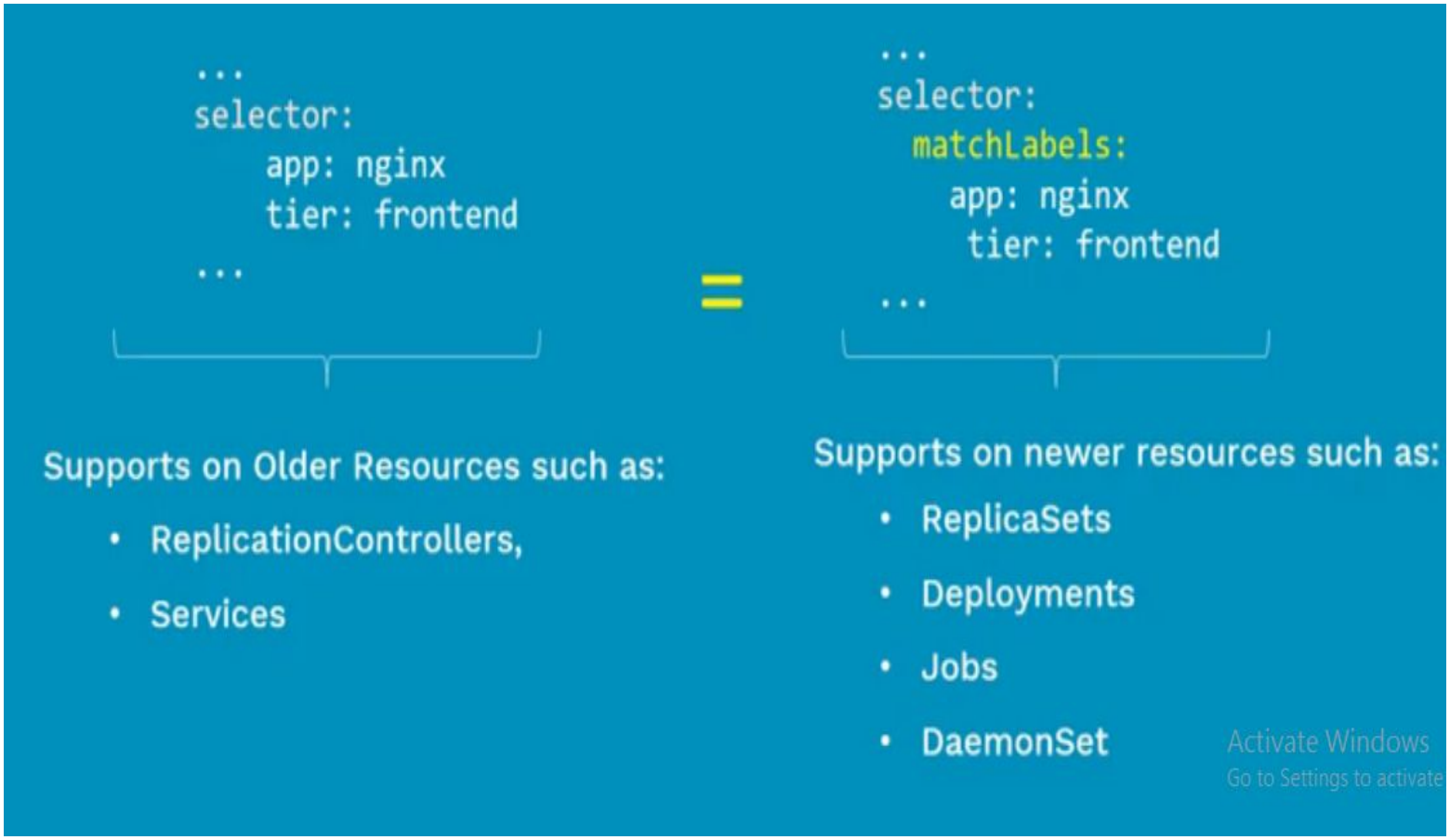


Replication controller vs Replica set



Replication controller vs Replica set

- Where to use what



DEPLOYMENTS

- Scenario

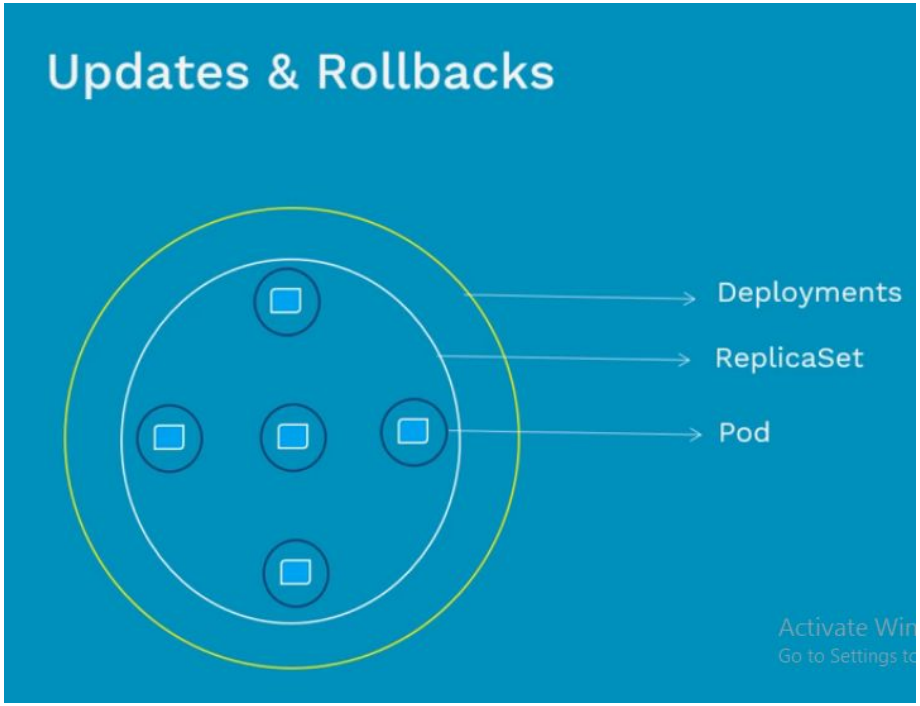
Scenario

Imagine you are upgrading an application from version v1 to version v2

- Upgrade with zero downtime
- Upgrade sequentially, one after another
- Pause and Resume upgrade process
- Rollback upgrade to previous normal stable release

DEPLOYMENTS

- What is deployments?
  - It is just a controller
  - It all about updates and rollback
  - RS does not provide this upgrade and downgrade feature



Deployment Types

- Recreate:
  - First shut down version v1 then start deploying version v2
  - Downtime is always there
- Rolling-Update:
  - Slowly roll out the version of an application by replacing instances one after the other until all the instances successfully rolled out
- Canary:
  - Consist of gradually shifting production traffic from v1 to v2

Deployment Types

- Blue/Green
  - Version v2 which is green id deployed along with the side version v1 which is blue with exactly same amount of instances
  - After testing new version with all the requests, the traffic gets switched from v1 to v2 at load balancer level

Scenario

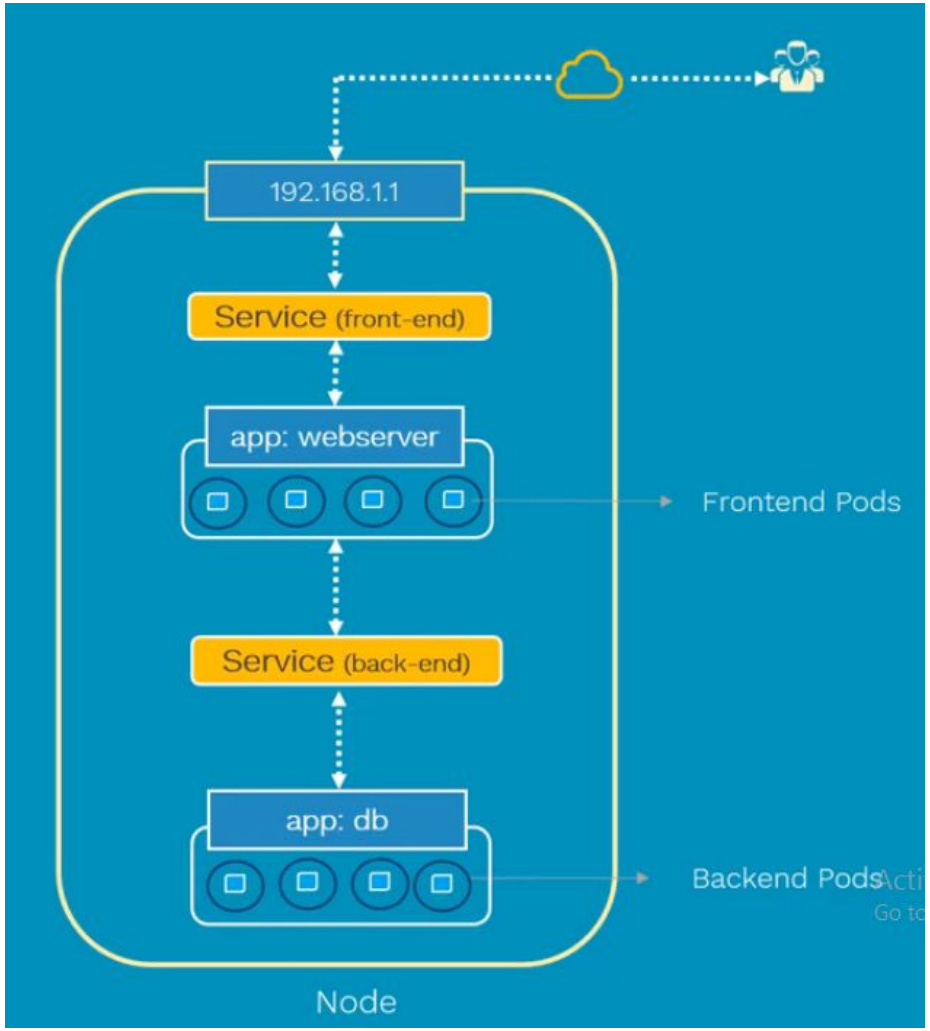
Imagine, you have been asked to deploy web app

- How does this front end web app is exposed to the outside world?
- How do front end web app is connected to backend DB?
- How do we resolve Pod IP changes, when they die?

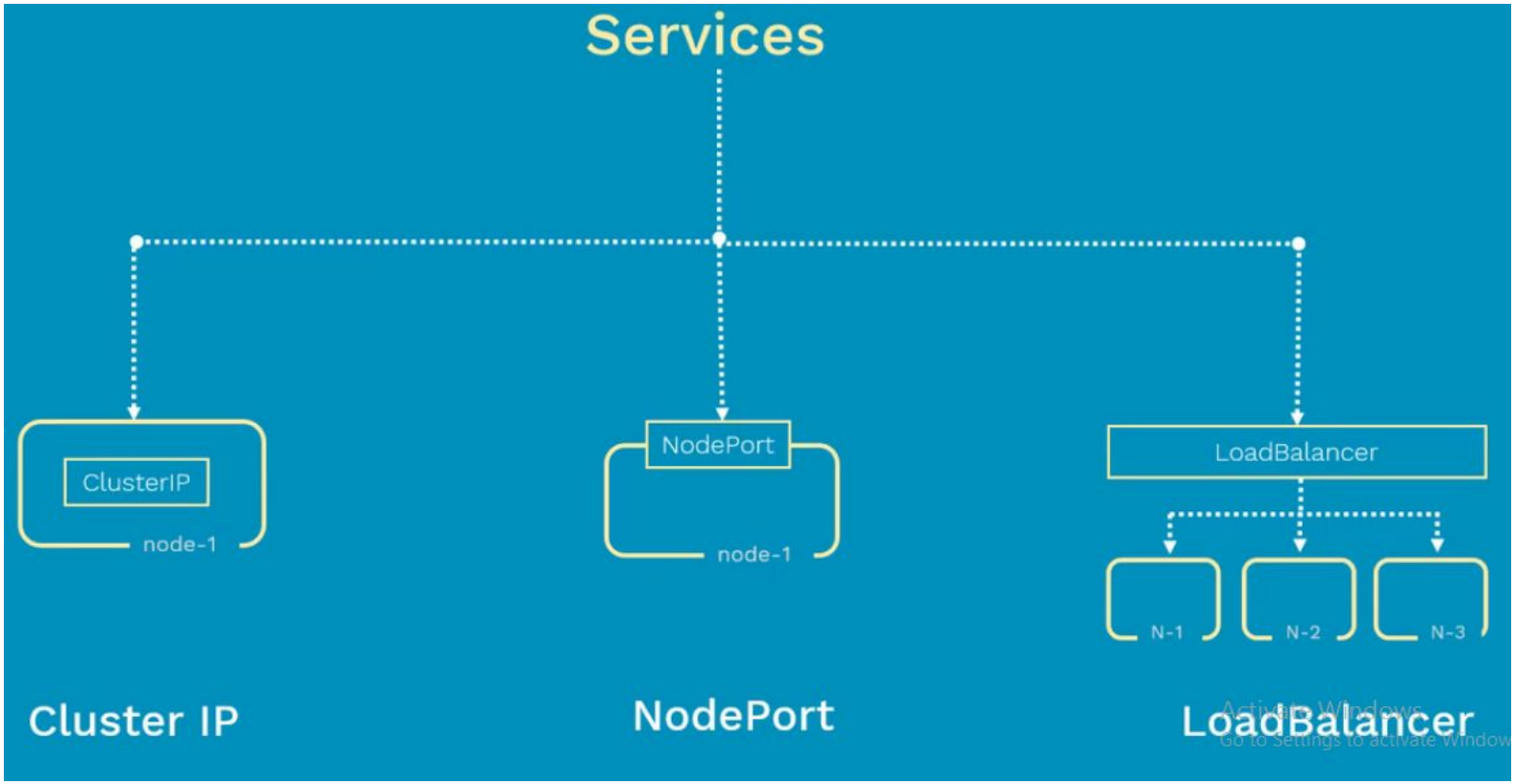
Services

- Why we need a Service
  - Is it possible to have permanent IP addresses?
  - How do various components connect and communicate?
  - How do applications are exposed to outside world?
- What is a Service
  - It is a way of grouping of pods that are running on the cluster
  - Features
    - Load balancing
    - Service discovery between applications

Services


192.168.1.1
Service (front-end)
app: webserver
Frontend Pods
Service (back-end)
app: db
Backend Pods
Node

Type of Services

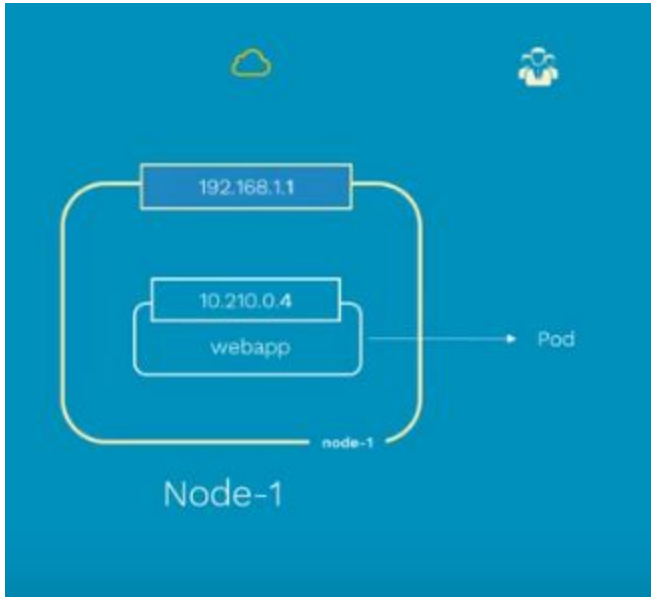


Scenario

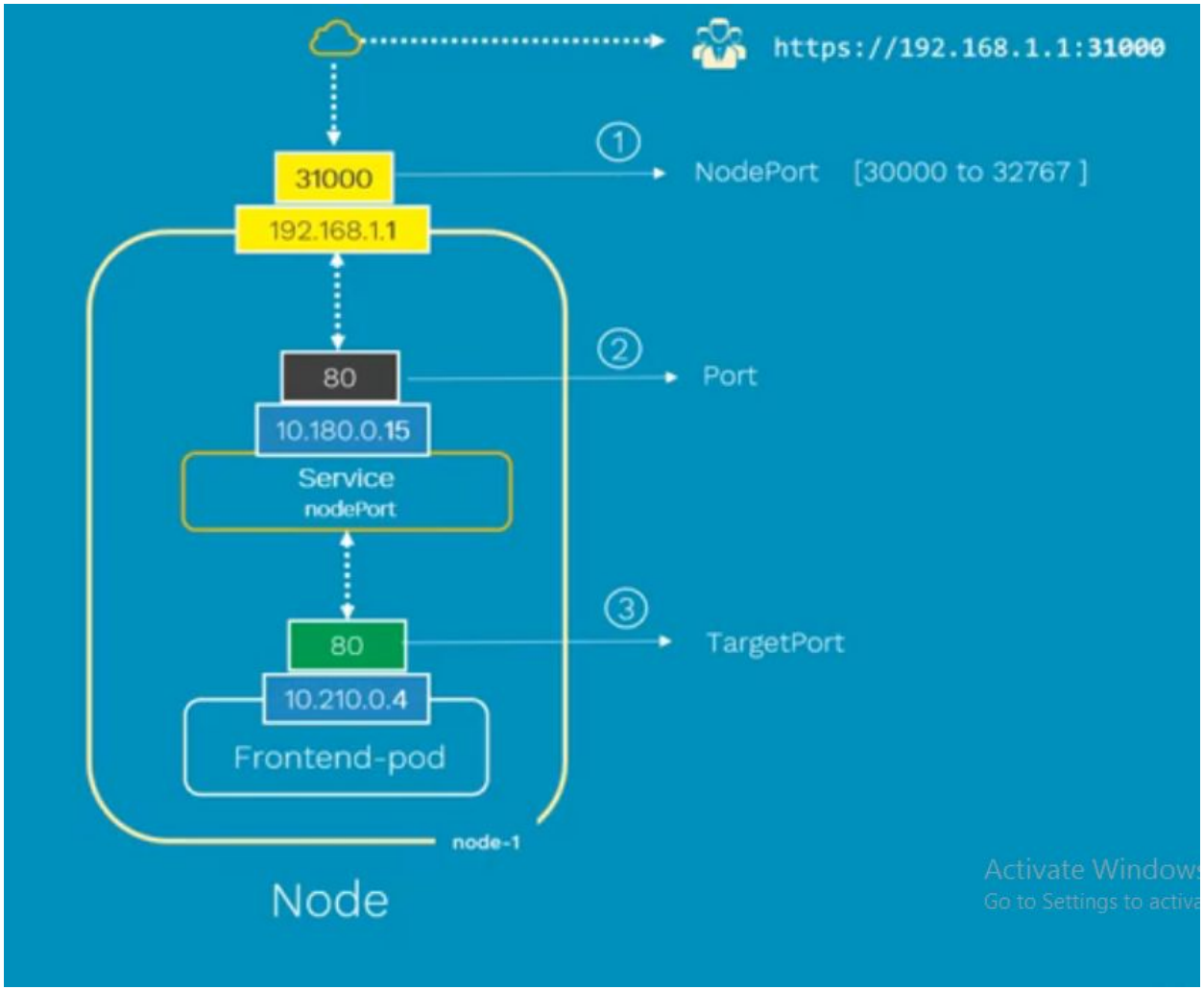
- Imagine that, you have deployed your web app on to Kubernetes cluster
- Now, you need to expose it outside the world on the internet

NodePort Service

- Why we need NodePort?
  - What if pod dies
  - No connectivity between users and pod



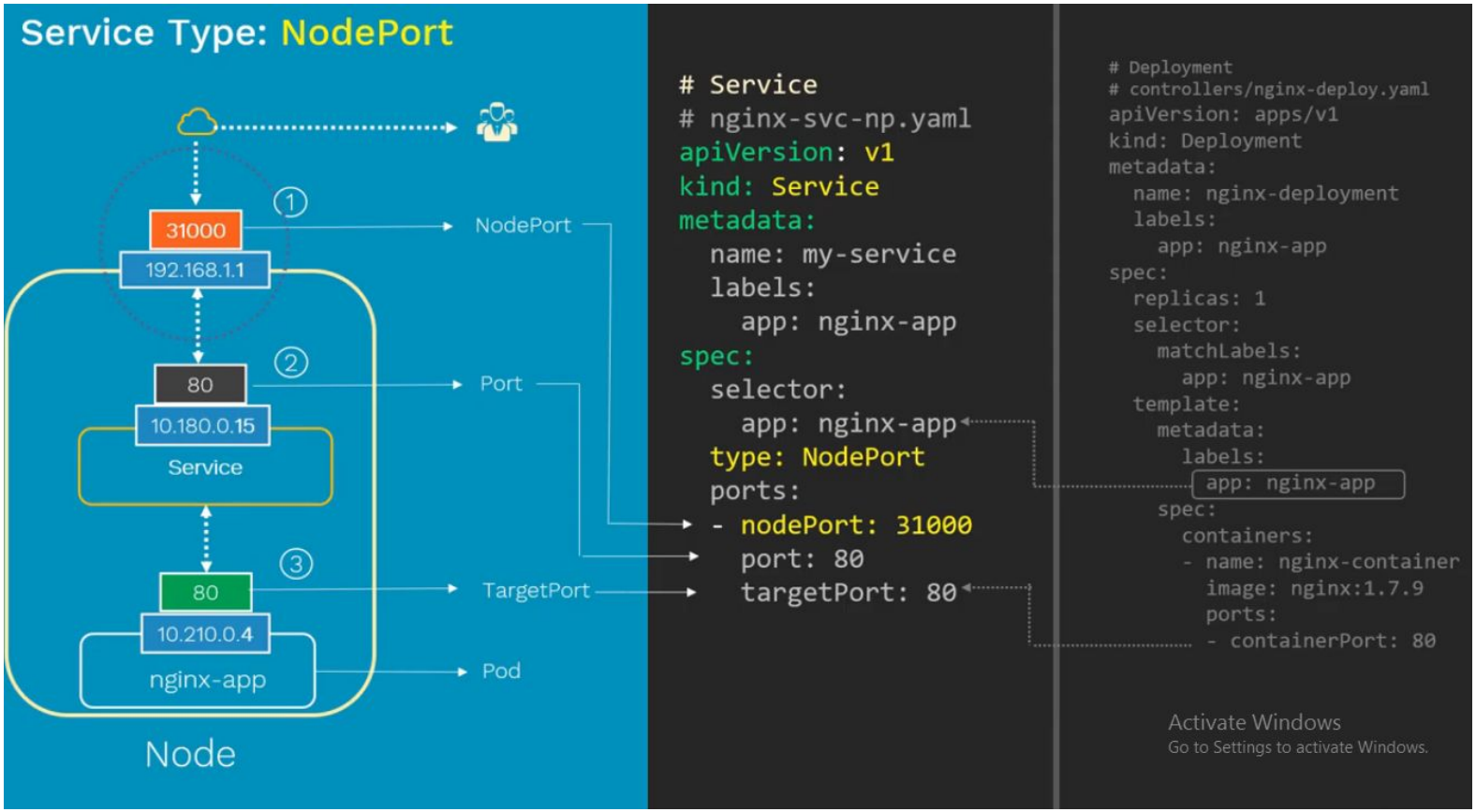
NodePort Service



NodePort Service

- NodePort
  - Port on a node where pod is running
- Port
  - Port on service itself
- TargetPort
  - Port on actual pod where web app is running
  - Typically Service port and target port will be same

NodePort Service



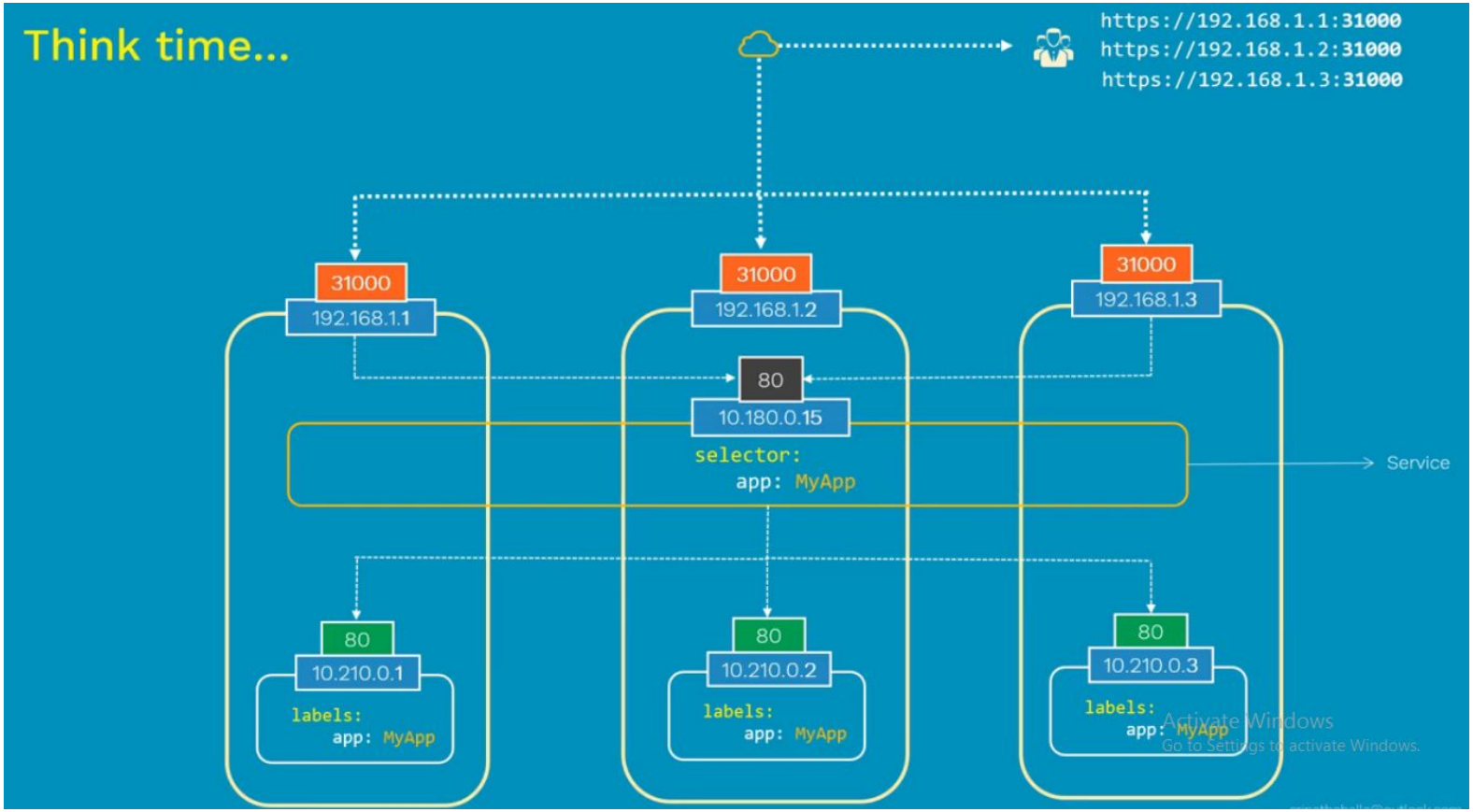
Scenario

- Imagine that, using NodePort service type you exposed your web app to outside the world on the internet
- Which node IP and nodePort will you provide to end users?

Load Balancer service

- Why we need Load Balancer Service

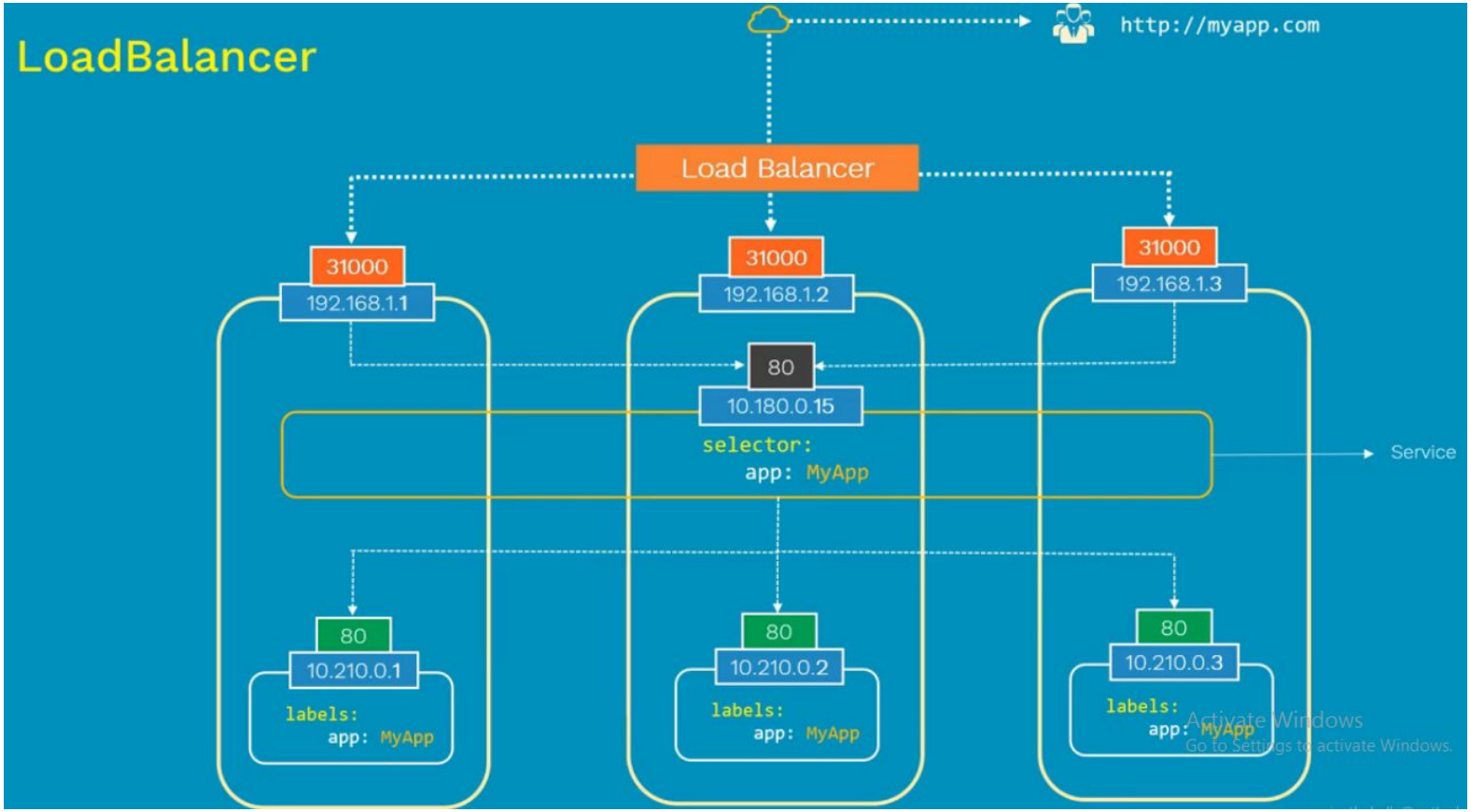
Load Balancer service



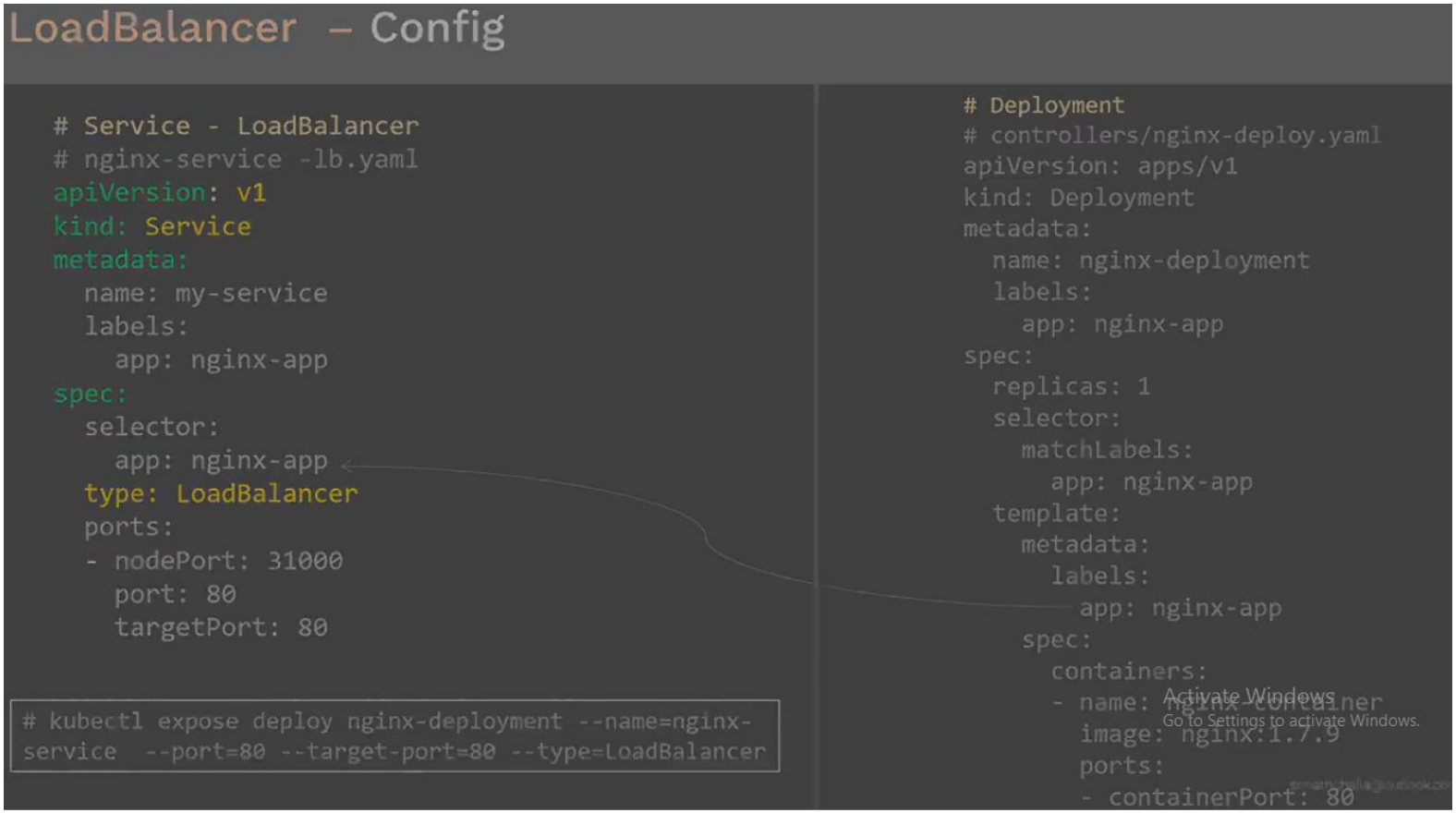
Load Balancer service

- It is a standard goto solution to expose your application on to the internet
- If you create service of type LB in AWS, Azure etc. it will create the public IP address that we can use to access your application publicly

Load Balancer service



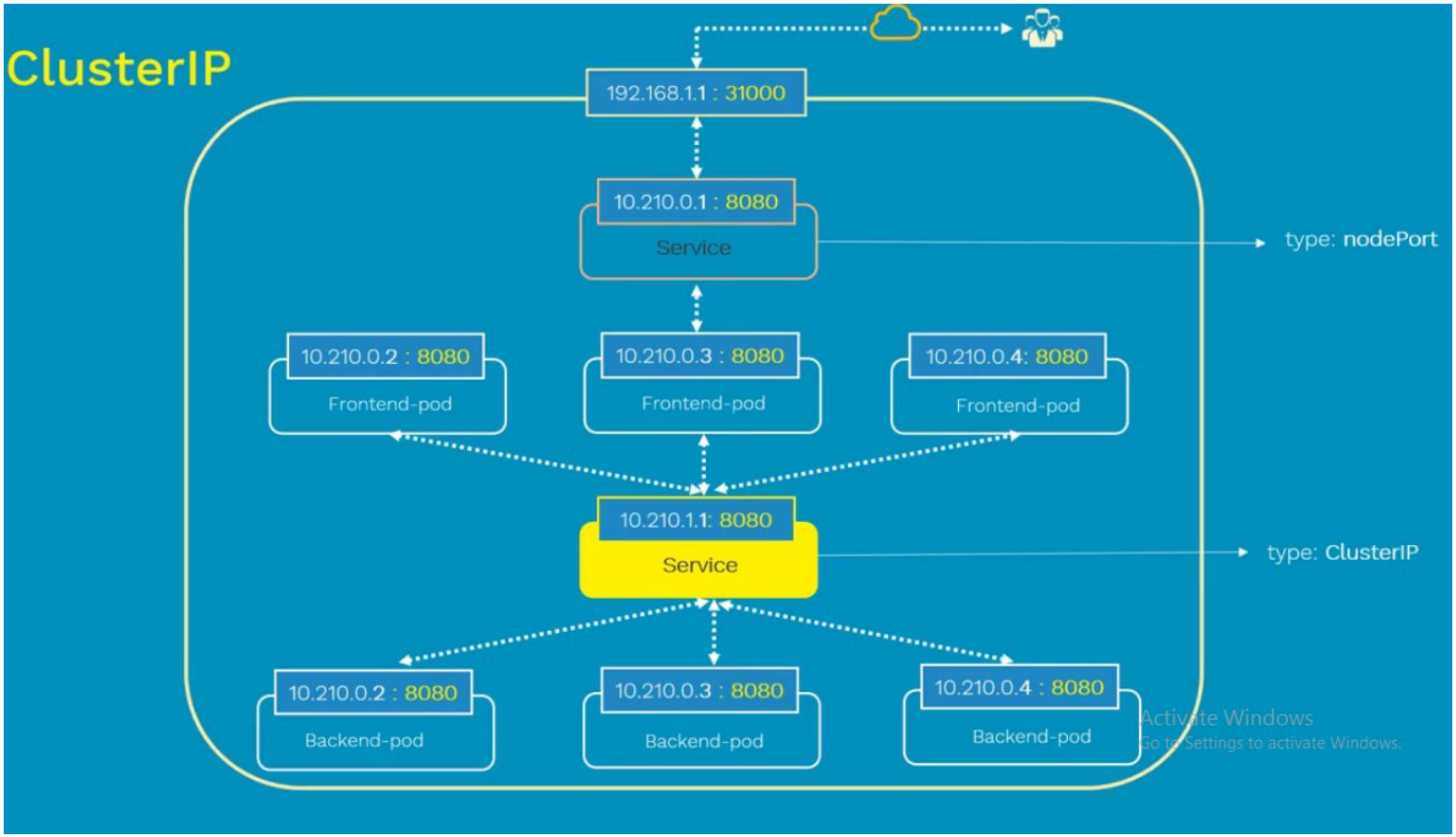
Load Balancer service



Scenario

- Imagine that, you need to deploy one full fledge app which consists of frontend application and backed database
- How can we restrict access of backend database to only within the kubernetes cluster?

ClusterIp Service



Thank you!!!