# Kubernetes

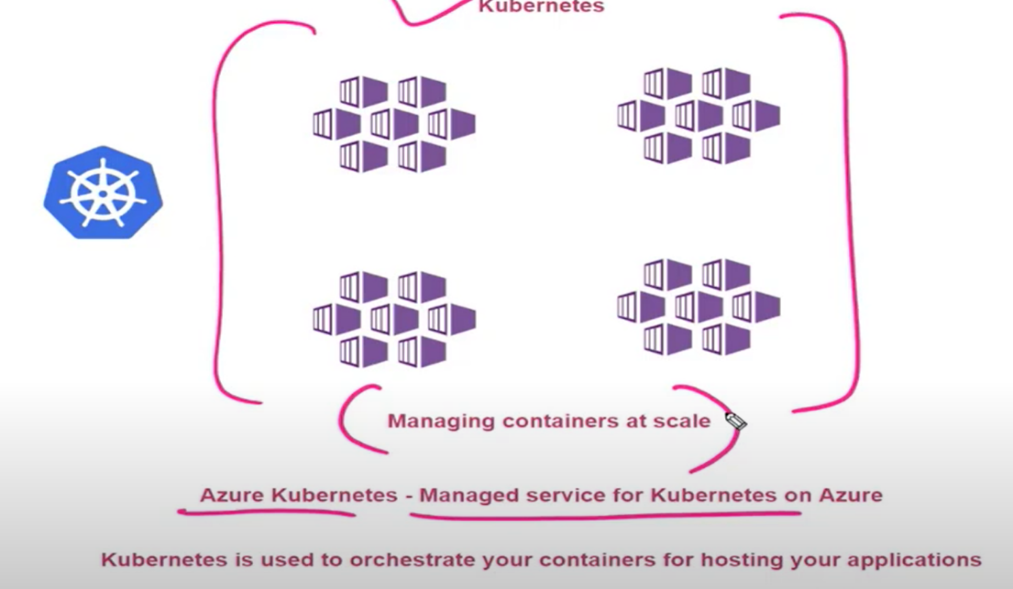
# 

1. **Github Repository Links**

* Azure AKS Kubernetes - Masterclass | Azure DevOps, Terraform: <https://github.com/stacksimplify/azure-aks-kubernetes-masterclass>
* Azure DevOps for Kubernetes Workloads running on Azure AKS Cluster: <https://github.com/stacksimplify/azure-devops-github-acr-aks-app1>
* Provision Azure AKS Cluster using Terraform and Azure DevOps: <https://github.com/stacksimplify/azure-devops-aks-kubernetes-terraform-pipeline>
* Docker Fundamentals: <https://github.com/stacksimplify/docker-fundamentals>
* Presentation with 250 Slides outlining the various architectures and designs we are going to do in this course: <https://github.com/stacksimplify/azure-aks-kubernetes-masterclass/tree/master/ppt-presentation>

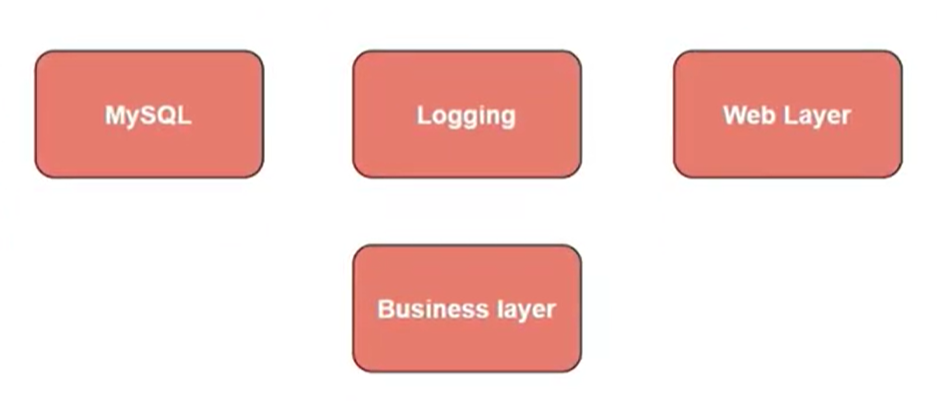
1. **Kubernetes Fundamental:**

Kubernetes helps to manage container at scale -> Companies develop applications that might have lot of containers -> manage these containers -> managing these containers is challenging

****

Kubernetes helps to deploy these applications on scale along with other feature to manage like load balancing your container and ensure that containers are always are up and running.

So Kubernetes orchestrate container for hosting your application -> You might have individual container for all of below components -> and combining all these components make your application



You can go and deploy these containers as an application on Kubernetes cluster -> Cluster is responsible to ensuring that these containers are running underline Nodes.

So instead of deploying each component one by one as container -> better to deploy them as set of clusters.

Kubernetes is portable, extensible, open-source platform for managing containerized workloads. It offers.

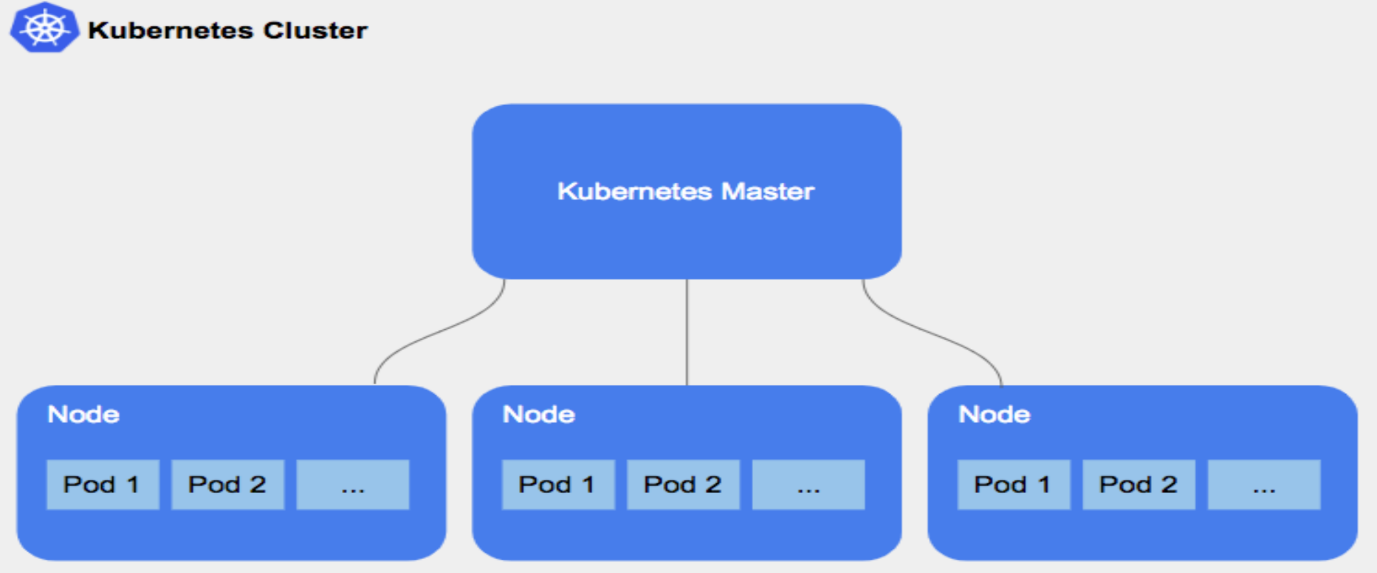
* Load balancing.
* Storage Orchestration
* Automated scale up and scale down
* Secret and configuration management

1. **Initial Configuration Kubernetes**

Refer - [azure-aks-kubernetes-masterclass/01-Create-AKS-Cluster at master · stacksimplify/azure-aks-kubernetes-masterclass (github.com)](https://github.com/stacksimplify/azure-aks-kubernetes-masterclass/tree/master/01-Create-AKS-Cluster)

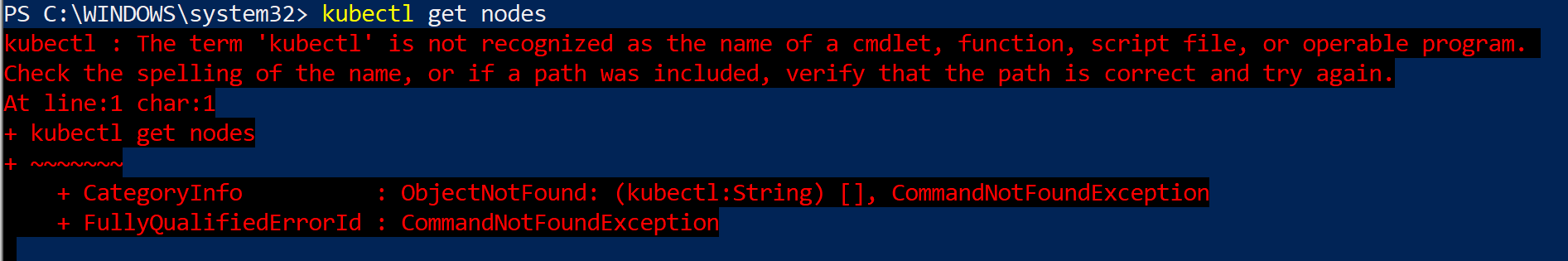
1. **Kubernetes Pods**

* With Kubernetes our core goal will be to deploy our applications in the form of container on worker nodes in a Kubernetes cluster.
* Kubernetes does not deploy containers directly on the worker nodes.
* container is encapsulated into a Kubernetes object named POD.
* A POD is a single instance of an application.
* A POD is the smallest object that we can create in Kubernetes
* PODs generally have one to one relationship with container.
* To scale up we create new POD and delete when need to scale down.
* We can have multi container in same POD only in case if containers are different kind or have different operation.

****

1. **Create Pods and Understand more and delete Pods**
   1. **Local Desktop - Install Azure CLI and Azure AKS CLI**

* # Install Azure CLI (MAC) -> brew update && brew install azure-cli
* # Login to Azure -> az login
* # Install Azure AKS CLI -> az aks install-cli
* # Configure Cluster Creds (kube config) -. az aks get-credentials --resource-group aks-rg1 --name aksdemo1
* # List AKS Nodes -> kubectl get nodes || kubectl get nodes -o wide
  1. **Get Worker Nodes Status**
* First -> az login
* Install Azure AKS CLI -> az aks install-cli
* Configure Cluster Creds (kube config) for Azure AKS Clusters from local desktop-> az aks get-credentials --resource-group rtlkubernetesgroup --name rtlkubernetes
* Get Worker Node Status -> kubectl get nodes
* If need additional information -> kubectl get nodes -o wide
  1. **Explore Cluster Control Plane and Workload inside that**
* Get the list of all namespace running in Pod -> kubectl get pods –all-namespaces
* List Namespaces -> kubectl get ns
* List all k8s objects from Cluster Control plane -> kubectl get all --all-namespaces
  1. **Error**



**Step 1:** Download the latest version of kubectl.exe

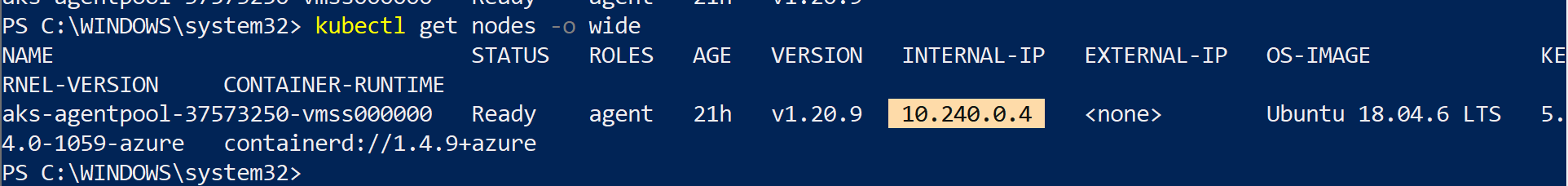
[Install and Set Up kubectl on Windows | Kubernetes](https://kubernetes.io/docs/tasks/tools/install-kubectl-windows/)

**Step 2:** Save the exe file in C drive -> **C:\Kubernetes**

**Step 3:** Give the path into System variable

* 1. **Create a Pod**
* kubectl run <desired-pod-name> --image <Container-Image>
* Replace Pod Name, Container Image - kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0
  1. **List Pods**
* kubectl get pods
* kubectl get pods -o wide
  1. **Describe Pod**
* Describe the POD, primarily required during troubleshooting.
* Events shown will be of a great help during troubleshooting.
* To get list of pod names -> kubectl get pods
* Describe the Pod -> kubectl describe pod <Pod-Name>
* kubectl describe pod my-first-pod
  1. **Access Application**

When we deploy the image on Pod -> it actually run on private IP, So we can’t access the application deployed directly.

****

* To access it externally, we need to create a NodePort or Load Balancer Service.
* Services is one very important concept in Kubernetes.
  1. **Delete Pod**
* To get list of pod names -> kubectl get pods
* Delete Pod -> kubectl delete pod <Pod-Name>
* kubectl delete pod my-first-pod

1. **Load Balancer Service in Kubernetes:**

We can expose an application running on a set of PODs using different types of services available in Kubernetes.

* ClusterIP service (Internal to Kubernetes cluster)
* NodePort service (To Internet)
* Load Balancer Service (To Internet)
* Ingress Service (To Internet)

To access our application outside of Azure AKS cluster, we can use Kubernetes Load Balancer service -> Which will be eventually mapped to Azure Standard Load Balancer

So In Azure Cloud -> We create Kubernetes Service (Has Kubernetes Cluster) -> At the same time It creates Load Balancer

When we deploy container in Node -> You deploy your service as well -> Any service for Node and Port (Service Port or we can say Cluster IP Port) and Target Port (Container Port in a POD)

1. **Expose POD with Service and Access in Browser**

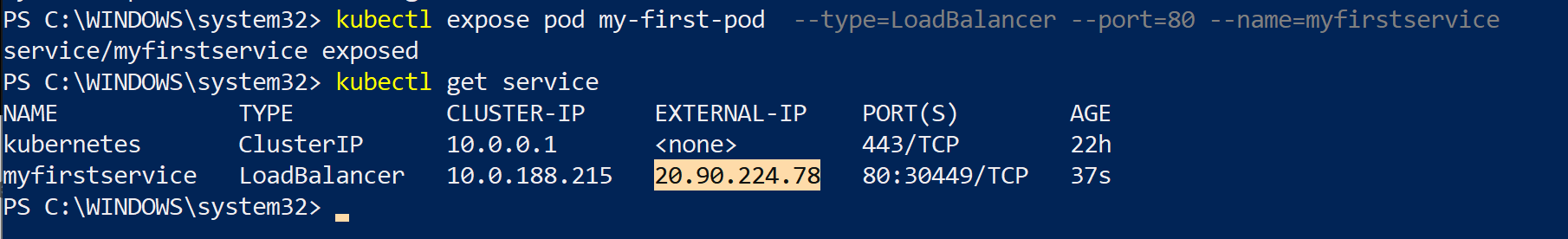
Expose pod with a service (Load Balancer Service) to access the application externally (from internet)

Ports

* **Port:** Port on which node port service listens in Kubernetes cluster internally
* **TargetPort:** We define container port here on which our application is running.

**Steps:**

* Search Public IP created during Kubernetes Service creation
* Now go to the Load Balancer, cross verify frontend IP and backend Pool
* # Create a Pod -> kubectl run <desired-pod-name> --image <Container-Image>
* # Expose Pod as a Service -> kubectl expose pod <Pod-Name> --type=LoadBalancer --port=80 --name=<Service-Name>
* kubectl expose pod my-first-pod --type=LoadBalancer --port=80 --name=my-first-service
* # Get Service Info -> kubectl get service, kubectl get svc
* # Describe Service -> kubectl describe service my-first-service
* # Access Application -> http://<External-IP-from-get-service-output>



Use this public IP and Browse the application deployed in POD

* Now when you go to Public IP in Azure -> You will se a new public IP is created and this Public IP also mapped to Azure Load balancer ( We can check into Frontend IP in Load Balancer)
* And New Load Balancing Rule is created in Load Balancer Resource

1. **Interact with Pods, Logs, connect to Pod and Clean-up**

* Get Pod Name -> kubectl get pods
* Dump Pod logs -> kubectl logs <pod-name>
* kubectl logs my-first-pod
* Stream pod logs with -f option and access application to see logs -> kubectl logs <pod-name>
* kubectl logs -f my-first-pod
  1. **Get YAML Output**
* Get pod definition YAML output -> kubectl get pod my-first-pod -o yaml
* Get service definition YAML output -> kubectl get service my-first-service -o yaml
  1. **Clean-Up**

# Get all Objects in default namespace -> kubectl get all

# Delete Services -> kubectl delete svc my-first-service

# Delete Pod -> kubectl delete pod my-first-pod

# Get all Objects in default namespace -> kubectl get all

1. **Replica Sets**

Replica Sets help us achieving high availability and reliability for our application hosted on Kubernetes.



So, if our application crashes or die in any Pods -> ReplicaSet automatically creates another PODs with the running instance of application.

* 1. **Create Replica Set**

kubectl create -f replicaset-demo.yml

* 1. **List ReplicaSets**
* Get list of ReplicaSets -> kubectl get replicaset or kubectl get rs
* Describe ReplicaSet -> Describe the newly created ReplicaSet -> kubectl describe rs/<replicaset-name>
* kubectl describe rs/my-helloworld-rs [or] kubectl describe rs my-helloworld-rs
  1. **List of Pods**
* Get list of Pods -> kubectl get pods
* Get list of Pods with Pod IP and Node in which it is running ->kubectl get pods -o wide
  1. **Verify the Owner of the Pod and nametag**
* kubectl get pods <pod-name> -o yaml
* kubectl get pods my-helloworld-rs-c8rrj -o yaml
  1. **Expose ReplicaSet as a Service**

Expose ReplicaSet with a service (Load Balancer Service) to access the application externally (from internet)

* **Expose ReplicaSet as a Service**

kubectl expose rs <ReplicaSet-Name> --type=LoadBalancer --port=80 --target-port=8080 --name=<Service-Name-To-Be-Created>

kubectl expose rs my-helloworld-rs --type=LoadBalancer --port=80 --target-port=8080 --name=my-helloworld-rs-service

* **Get Service Info**

kubectl get service

kubectl get svc

* **Access the Application using External or Public IP**

http://<External-IP-from-get-service-output>/hello

* 1. **Test Replicaset Reliability or High Availability**
* Test how the high availability or reliability concept is achieved automatically in Kubernetes
* Whenever a POD is accidentally terminated due to some application issue, ReplicaSet should auto-create that Pod to maintain desired number of Replicas configured to achive High Availability.

# **To get Pod Name** -> kubectl get pods

# **Delete the Pod** -> kubectl delete pod <Pod-Name>

# **Verify the new pod got created automatically**-> kubectl get pods (Verify Age and name of new pod)

* 1. **ReplicaSet Scalability feature**

Change the Yaml file replica from 3 to 6 and re-deploy

# **Apply latest changes to ReplicaSet** -> kubectl replace -f replicaset-demo.yml

# **Verify if new pods got created** ->kubectl get pods -o wide

* 1. **Delete ReplicaSet & Service**

**Delete ReplicaSet**

* # Delete ReplicaSet -> kubectl delete rs <ReplicaSet-Name>
* # Sample Commands -> kubectl delete rs/my-helloworld-rs [or] kubectl delete rs my-helloworld-rs
* # Verify if ReplicaSet got deleted -> kubectl get rs

**Delete Service created for ReplicaSet**

* # Delete Service -> kubectl delete svc <service-name>
* # Sample Commands
* kubectl delete svc my-helloworld-rs-service [or] kubectl delete svc/my-helloworld-rs-service
* # Verify if Service got deleted -> kubectl get svc

1. **Kubernetes Deployment**

Deployment is superset of ReplicaSets -> It means Deployment provides additional feature on top of Replica Sets

There is no difference between accessing the Application in ReplicaSets or Deployment



* 1. **Create a Deployment**
* When we Create Deployment -> automatically a ReplicaSet created
* kubectl create deployment <Deplyment-Name> --image=<Container-Image>
* Image to be used -> **Docker Image Location:** <https://hub.docker.com/repository/docker/stacksimplify/kubenginx>

kubectl create deployment my-first-deployment --image=stacksimplify/kubenginx:1.0.0

* 1. **Verify Deployment**

kubectl get deployments

* 1. **Describe Deployment**
* kubectl describe deployment <deployment-name>

kubectl describe deployment my-first-deployment

* Verify ReplicaSet -> kubectl get rs
* Verify Pod -> kubectl get po
  1. **Scaling a Deployment**
* Scale the deployment to increase the number of replicas (pods)

**# Scale Up the Deployment**

* kubectl scale --replicas=10 deployment/<Deployment-Name>
* kubectl scale --replicas=10 deployment/my-first-deployment

**# Verify Deployment**

* kubectl get deploy

**# Verify ReplicaSet**

* kubectl get rs

**# Verify Pods**

* kubectl get po

**# Scale Down the Deployment**

* kubectl scale --replicas=2 deployment/my-first-deployment
* kubectl get deploy
  1. **Expose Deployment as a Service**
* Expose Deployment with a service (LoadBalancer Service) to access the application externally (from internet)

**# Expose Deployment as a Service**

* kubectl expose deployment <Deployment-Name> --type=LoadBalancer --port=80 --target-port=80 --name=<Service-Name-To-Be-Created>
* kubectl expose deployment my-first-deployment --type=LoadBalancer --port=80 --target-port=80 --name=my-first-deployment-service

**# Get Service Info ->** kubectl get svc

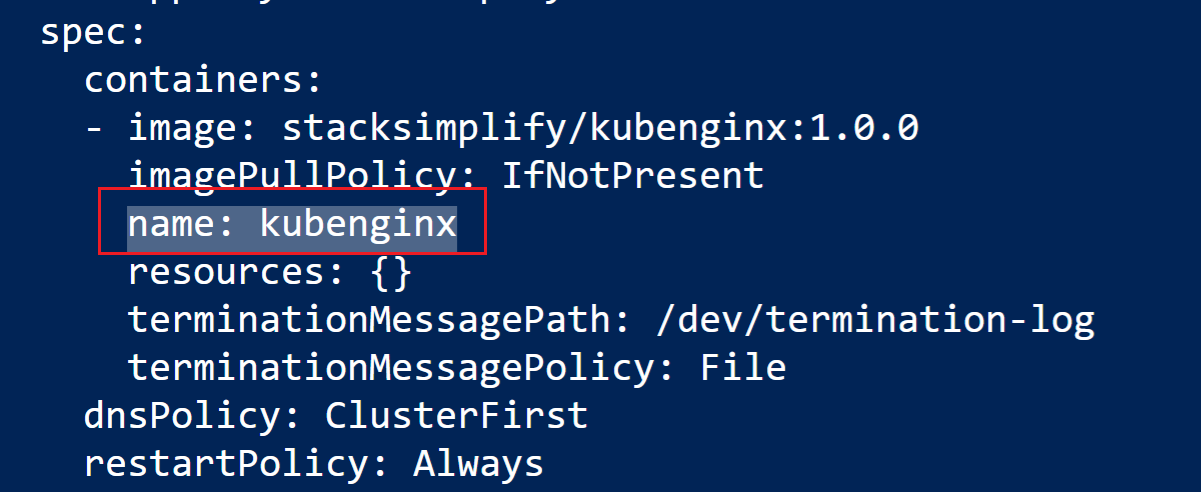
**Access the Application using Public IP ->** http://<External-IP-from-get-service-output>

1. **How to update Deployment**

Here we are going to upgrade the version of application deployed.

**# Get Container Name from current deployment**

* kubectl get deployment my-first-deployment -o yaml



**# Update Deployment - SHOULD WORK NOW**

* kubectl set image deployment/<Deployment-Name> <Container-Name>=<Container-Image> --record=true
* kubectl set image deployment/my-first-deployment kubenginx=stacksimplify/kubenginx:2.0.0 --record=true
  1. **Verify Rollout History of a Deployment**

We have the rollout history, so we can switch back to older revisions using revision history available to us.

# **Check the Rollout History of a Deployment**

* kubectl rollout history deployment/<Deployment-Name>
* kubectl rollout history deployment/my-first-deployment
  1. **Rollback Deployment to previous version**

We can rollback a deployment in two ways.

* Previous Version
* Specific Version

**Check the Rollout History of a Deployment**

# List Deployment Rollout History

* kubectl rollout history deployment/<Deployment-Name>
* kubectl rollout history deployment/my-first-deployment

**Verify changes in each revision**

List Deployment History with revision information

* kubectl rollout history deployment/my-first-deployment --revision=1
* kubectl rollout history deployment/my-first-deployment --revision=2

**Rollback to previous version**

Observation: If we rollback, it will go back to revision-2 and its number increases to revision-4

# Undo Deployment

* kubectl rollout undo deployment/my-first-deployment

1. **Kubernetes Services:**



1. **ConfigMap**

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

Note: ConfigMap does not provide secrecy or encryption. If the data you want to store are confidential, use a Secret rather than a ConfigMap

**Example:** Mp below configMap with SQL deployment, so that when SQL installed, it creates a database name webappdb

apiVersion: v1

kind: ConfigMap

metadata:

  name: usermanagement-dbcreation-script

data:

  mysql\_usermgmt.sql: |-

    DROP DATABASE IF EXISTS webappdb;

    CREATE DATABASE webappdb;

1. **Kubernetes Secrets**

* Kubernetes Secrets let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
* Storing confidential information in a Secret is safer and more flexible than putting it directly in a Pod definition or in a container image.

[azure-aks-kubernetes-masterclass/07-Kubernetes-Secrets at master · stacksimplify/azure-aks-kubernetes-masterclass (github.com)](https://github.com/stacksimplify/azure-aks-kubernetes-masterclass/tree/master/07-Kubernetes-Secrets)

apiVersion: v1

kind: Secret

metadata:

  name: mysql-db-password

type: Opaque

data:

  db-password: ZGJwYXNzd29yZDEx

1. **AKS Storage – Azure File**

These are simple secure and fully managed cloud file share. We can secure data at REST.

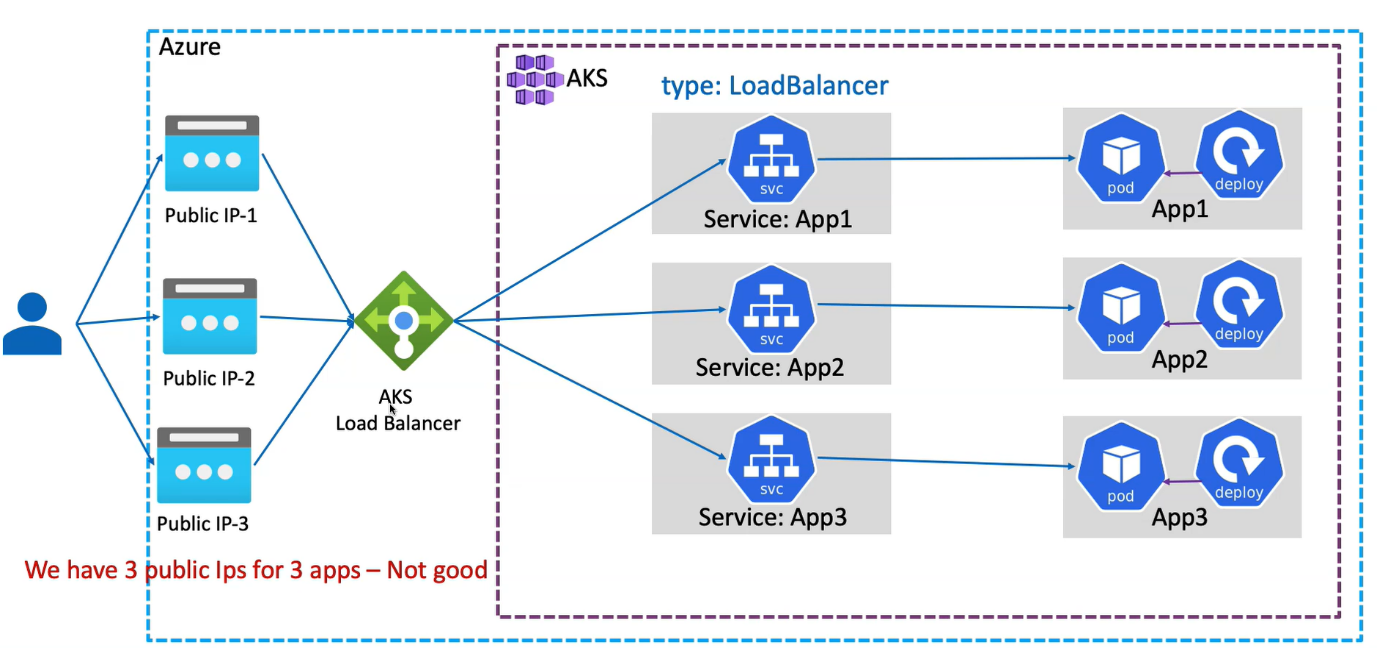
With default AKS created storage classes only below two options are available for us.

* Standard\_LRS - standard locally redundant storage (LRS)
* Premium\_LRS - premium locally redundant storage (LRS)

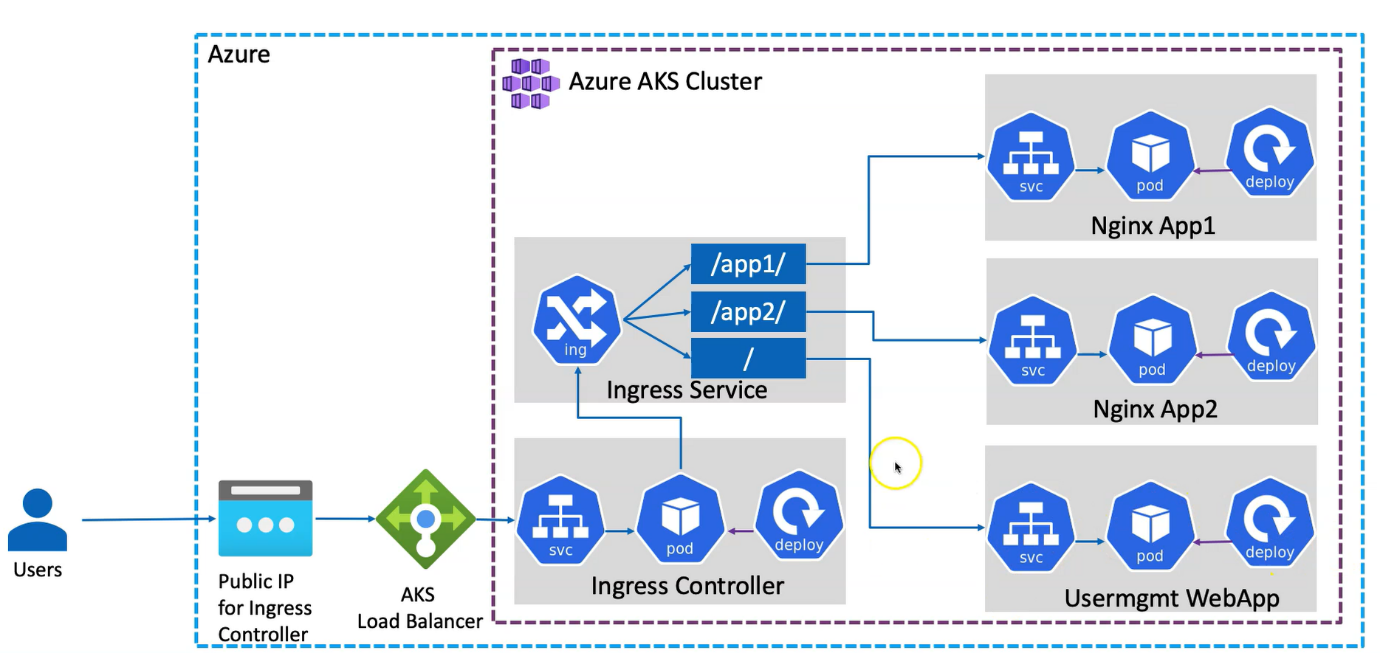
1. **AKS – Ingress**

Ingress is a Kubernetes resource that lets us configure an HTTP load balancer for applications running on Kubernetes, with advanced capabilities at HTTP layer.

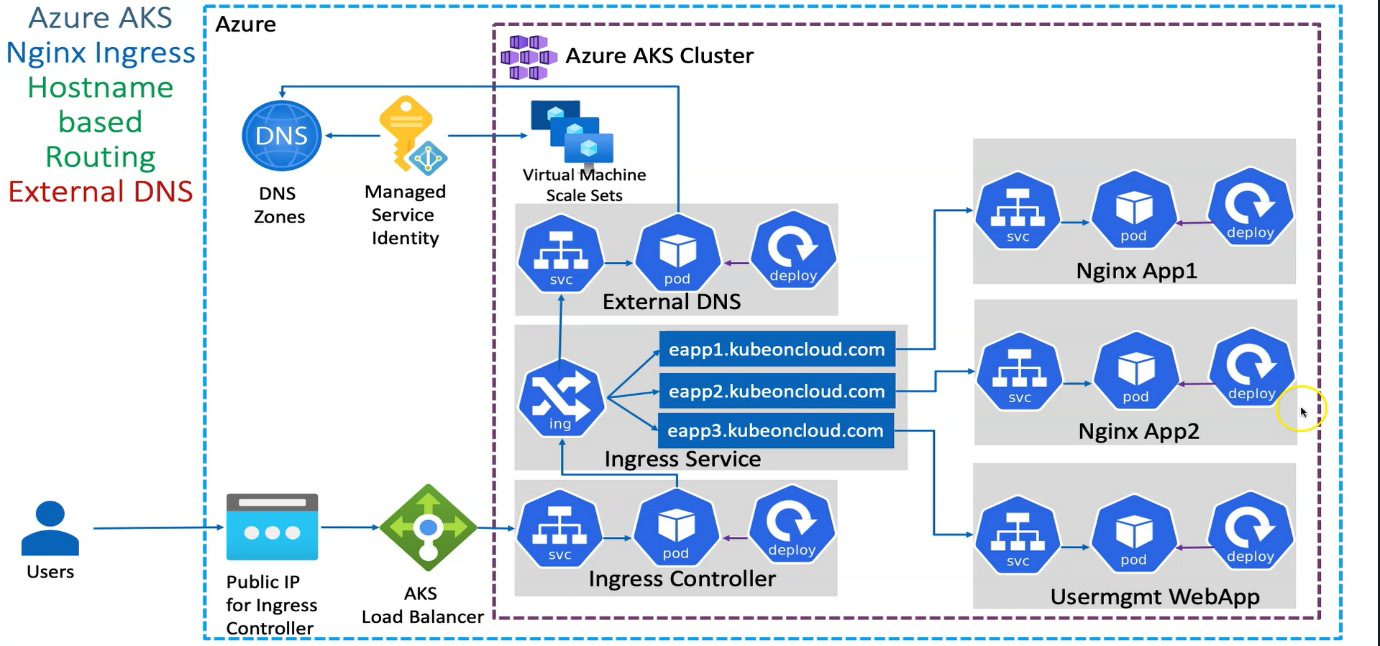
Usually when we deploy multiple application in AKS it goes to different Pods and for each App we have unique Load Balancer and unique public IP with which we access the application running in AKS cluster.



But if all need to be exposed with single IP, we use Ingress. So, with single public IP we can do context path-based routing, which defined into ingress service



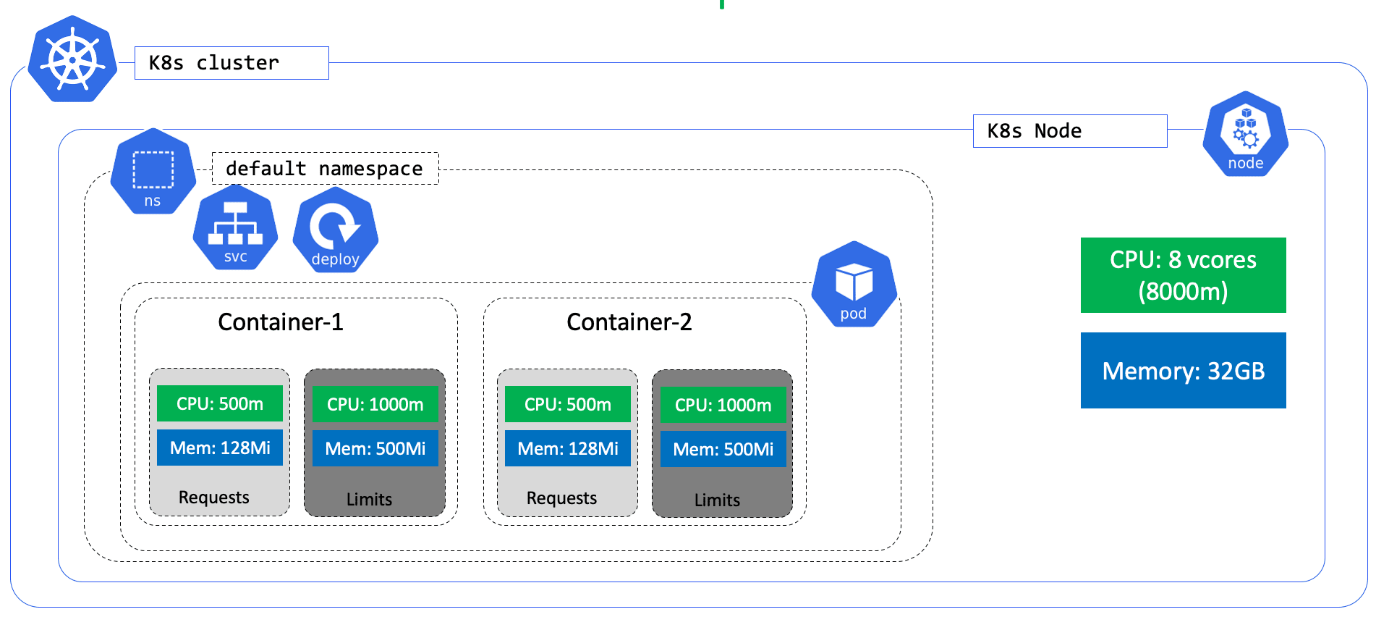
We can also register our DNS name and with DNS we can do context path-based routing.



* We are going to create a **Static Public IP** for Ingress in Azure AKS
* Associate that Public IP to **Ingress Controller** during installation.
* We are going to create a namespace ingress-basic for Ingress Controller where all ingress controller related things will be placed.
* Create / Review Ingress Manifest
* Deploy a simple Nginx App1 with Ingress manifest and test it
* Clean-Up or delete application after testing

Refer to - [azure-aks-kubernetes-masterclass/09-Ingress-Basic at master · stacksimplify/azure-aks-kubernetes-masterclass (github.com)](https://github.com/stacksimplify/azure-aks-kubernetes-masterclass/tree/master/09-Ingress-Basic)

1. **Kubernetes – Request & Limit**



We can set how much memory and CPU usage can be accessed by each container.

containers:

        - name: usermgmt-webapp

          image: stacksimplify/kube-usermgmt-webapp:1.0.0-MySQLDB

          imagePullPolicy: Always

          # Requests & Limits for usermgmt-webapp Container

          resources:

            requests:

              cpu: "500m"

              memory: "128Mi"

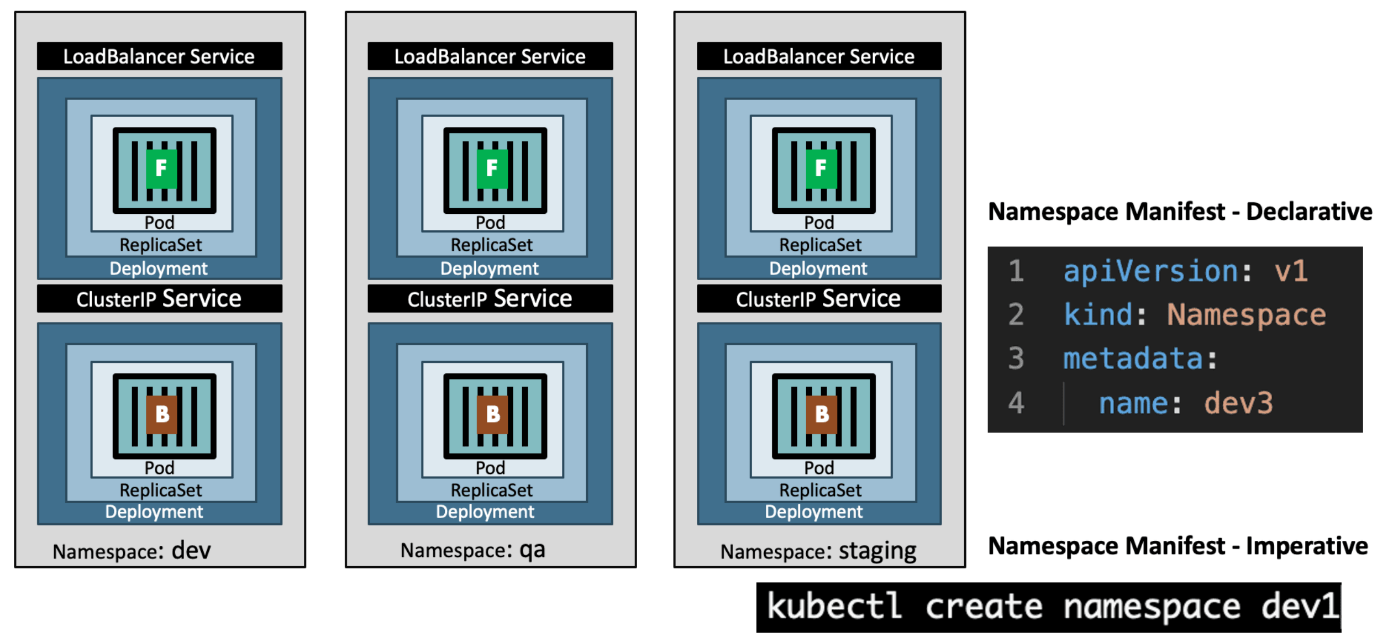
            limits:

              cpu: "1000m"

              memory: "500Mi"

1. **Kubernetes – Namespace**

* Namespaces allow to split-up resources into different groups.
* Resource names should be unique in a namespace
* We can use namespaces to create multiple environments like dev, staging and production etc
* Kubernetes will always list the resources from default namespace unless we provide exclusively from which namespace we need information from.



# Deploy All k8s Objects

kubectl apply -f kube-manifests/

kubectl apply -f kube-manifests/ -n dev1

kubectl apply -f kube-manifests/ -n dev2

1. **Azure Container Registry**

* Azure Container Registry is a managed, private Docker registry service.
* We can create and maintain Azure container registry to store and manage our private Docker container images.
* With ACR, we can simplify our container lifecycle management.

