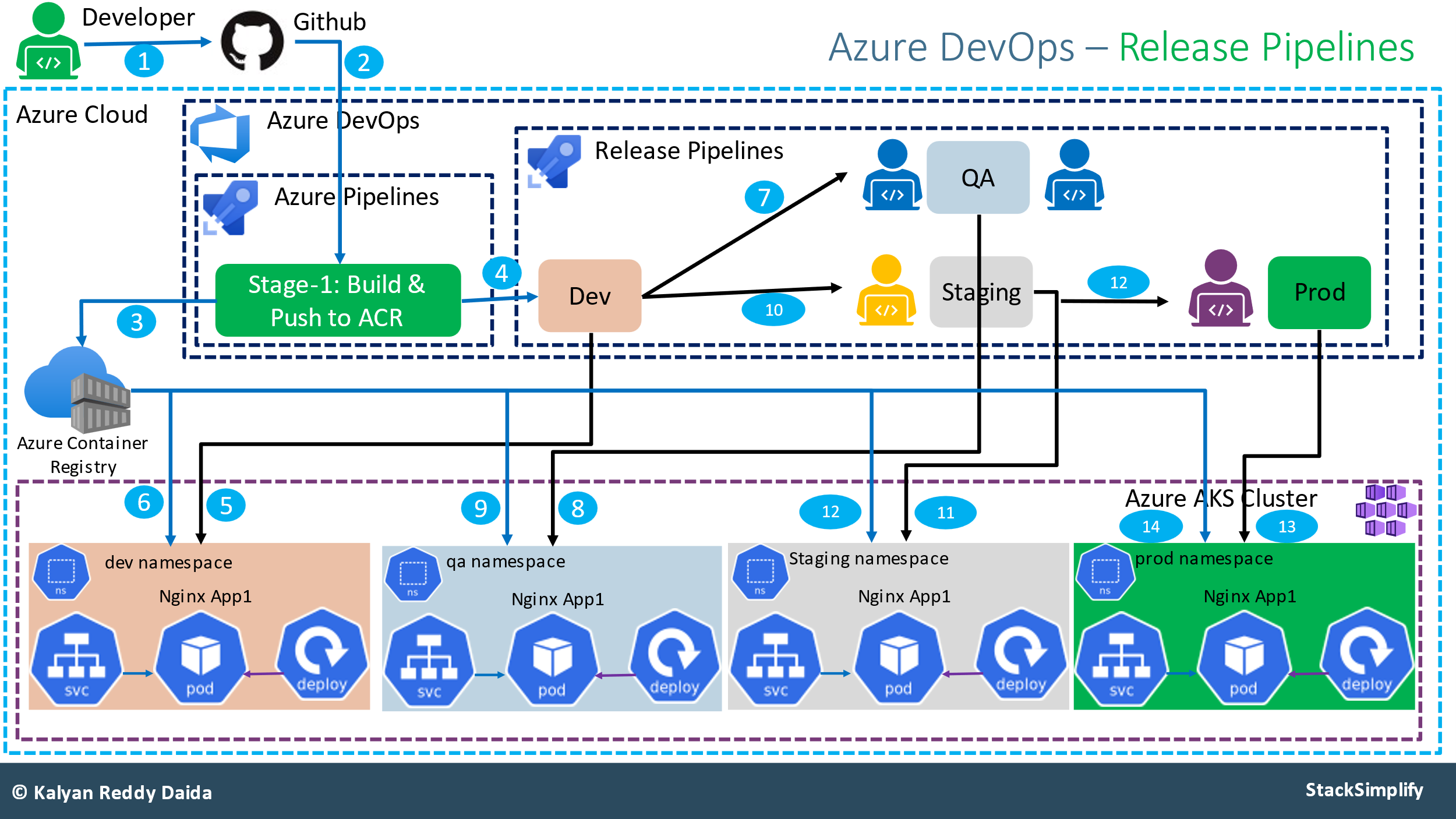
**Azure Kubernetes Service with Azure DevOps and Terraform**

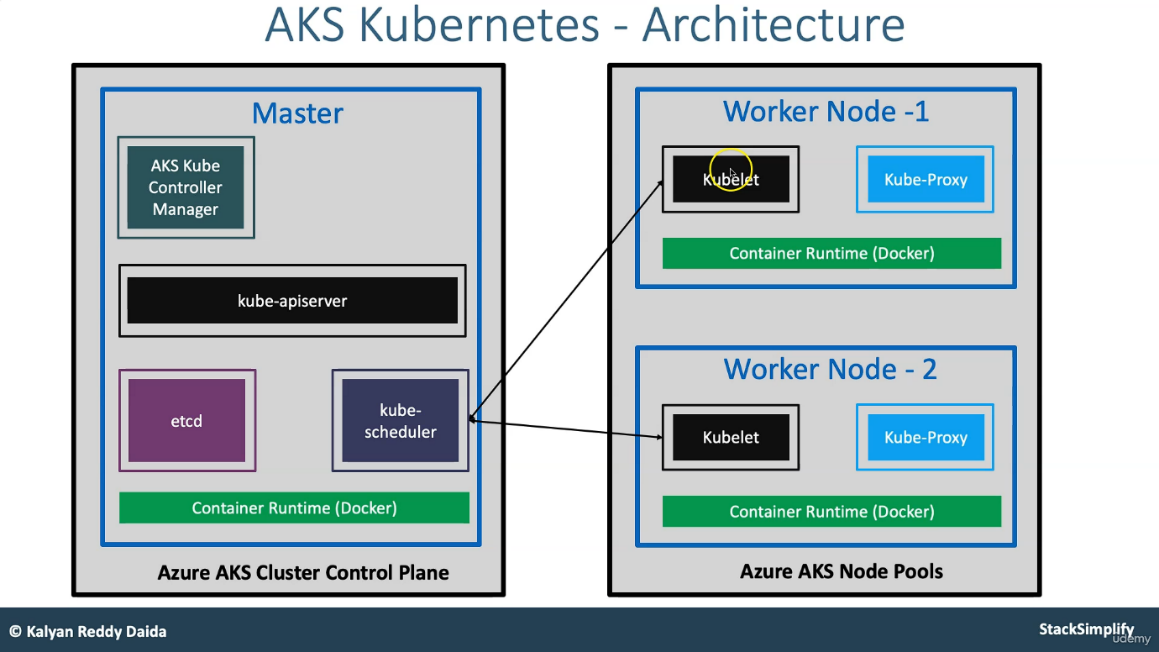
|  |  |
| --- | --- |
| Github Repository Links | Github Repositories used in this Course |
| Azure AKS Kubernetes - Masterclass | Azure DevOps, Terraform | https://github.com/stacksimplify/azure-aks-kubernetes-masterclass |
| Azure DevOps for Kubernetes Workloads running on Azure AKS Cluster | https://github.com/stacksimplify/azure-devops-github-acr-aks-app1 |
| 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 |

**Important Note:** Please go to these repositories and FORK these repositories and make use of them during the course.

|  |
| --- |
| # 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 -o wide  # Deploy Application  kubectl apply -f kube-manifests/  # Verify Pods  kubectl get pods  # Verify Deployment  kubectl get deployment  # Verify Service (Make a note of external ip)  kubectl get service  # Access Application  http://<External-IP-from-get-service-output>  Mannuval  kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0  kubectl get pods -o wide  kubectl describe pod my-first-pod |



AKS Architecture



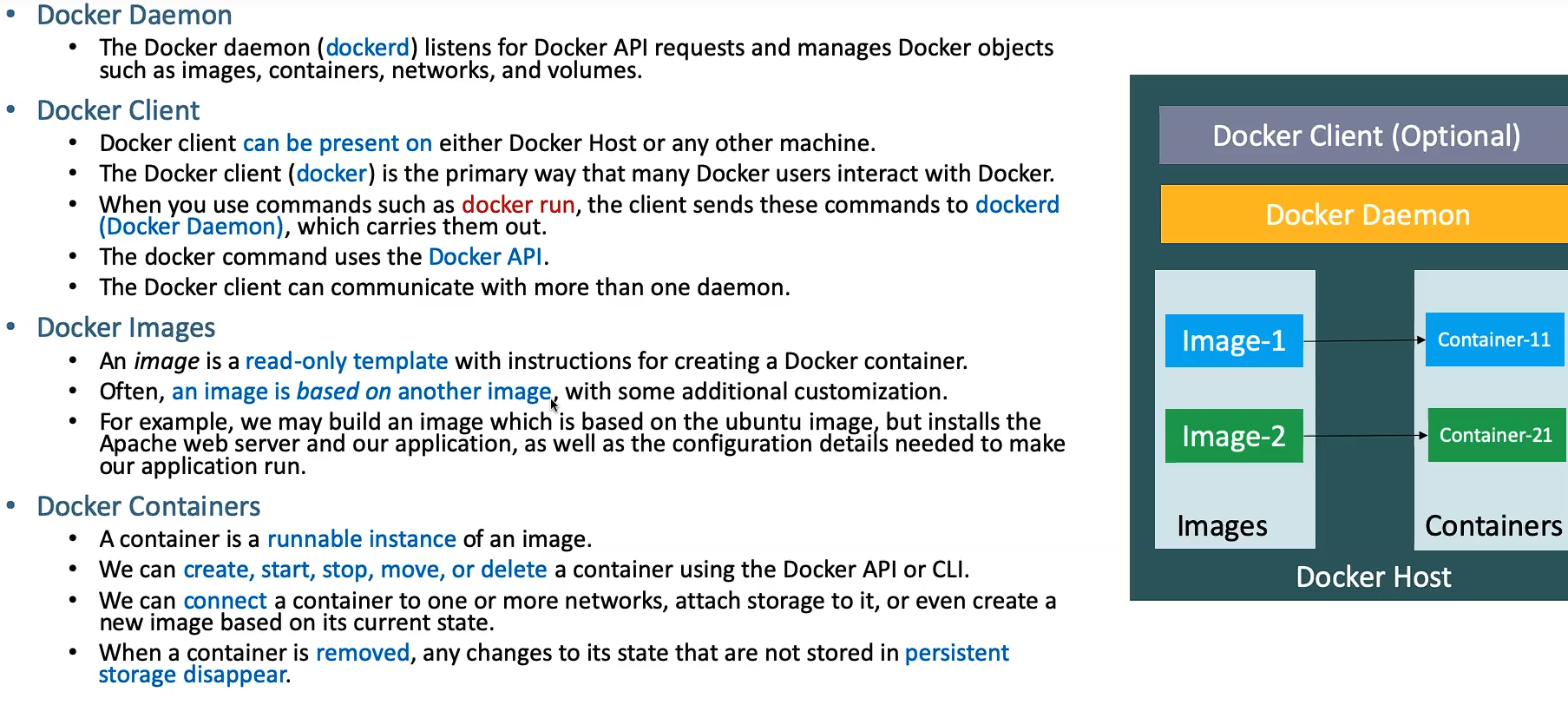
**Kubernetes Resources:**

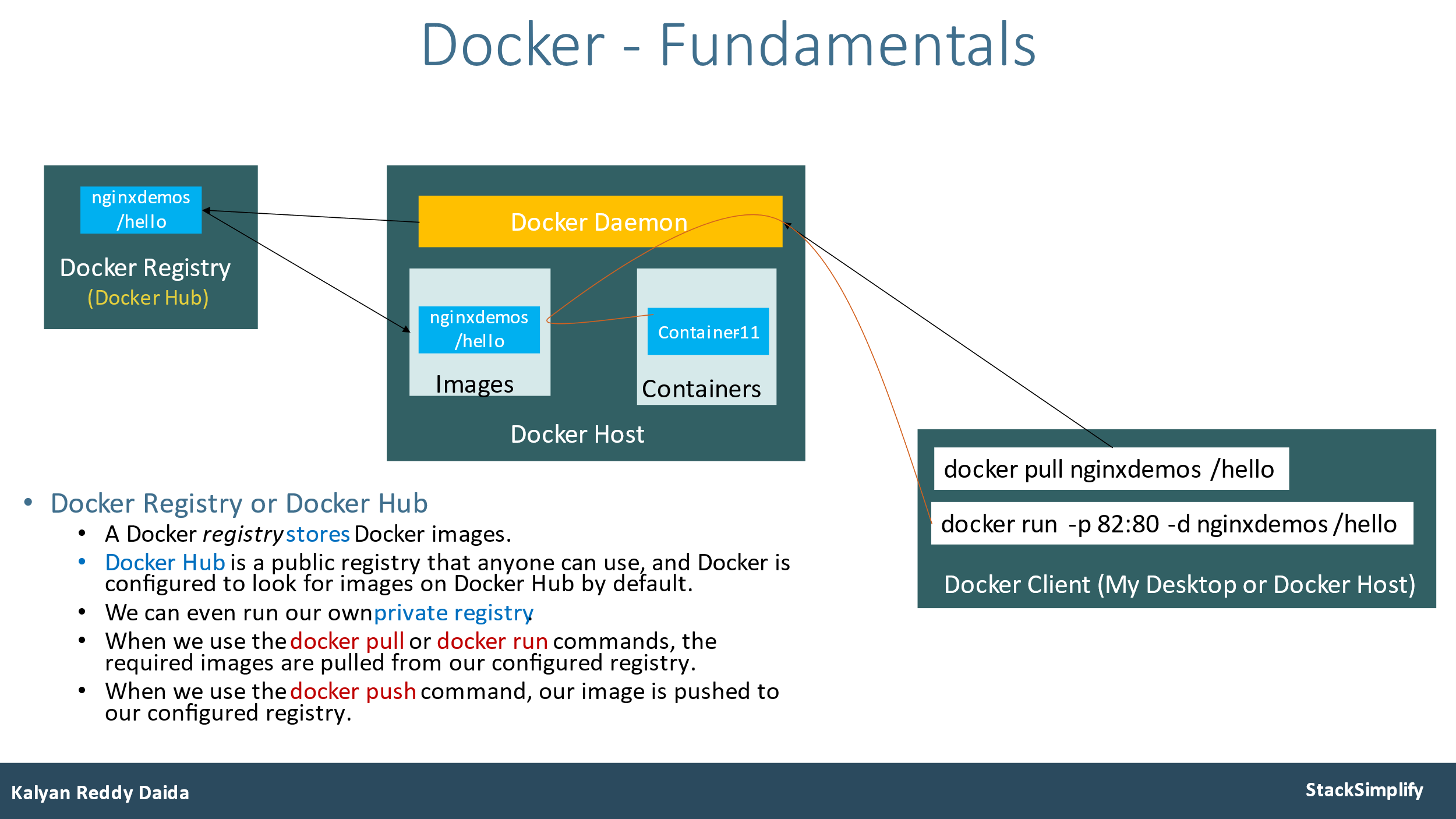
1. Namespaces (Preview)
2. Workload (Preview)
3. Service and Ingress (Preview)

Container🡪 flexible, portable, light weight, loose coupled, scalable, secure

**Docker Architecture:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| docker version  docker login  FROM nginx  COPY index.html /usr/share/nginx/html  docker build -t stacksimplify/mynginx\_image1:v1 .  docker run --name mynginx1 -p 80:80 -d stacksimplify/mynginx\_image1:v1  <http://localhost/hello>  docker images  docker tag stacksimplify/mynginx\_image1:v1 stacksimplify/mynginx\_image1:v1-release  docker push stacksimplify/mynginx\_image1:v1-release   | **Commands** | **Description** | | --- | --- | | docker ps | List all running containers | | docker ps -a | List all containers stopped, running | | docker stop container-id | Stop the container which is running | | docker start container-id | Start the container which is stopped | | docker restart container-id | Restart the container which is running | | docker port container-id | List port mappings of a specific container | | docker rm container-id or name | Remove the stopped container | | docker rm -f container-id or name | Remove the running container forcefully | | docker pull image-info | Pull the image from docker hub repository | | docker pull stacksimplify/springboot-helloworld-rest-api:2.0.0-RELEASE | Pull the image from docker hub repository | | docker exec -it container-name /bin/sh | Connect to linux container and execute commands in container | | docker rmi image-id | Remove the docker image | | docker logout | Logout from docker hub | | docker login -u username -p password | Login to docker hub | | docker stats | Display a live stream of container(s) resource usage statistics | | docker top container-id or name | Display the running processes of a container | | docker version | Show the Docker version information | |





**1.Service:** In Kubernetes, a Service is a method for exposing a **network application** that is running as one or more Pods in your cluster.

**Expose Pod with a Service**

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.

Verify the following before LB Service creation

Azure Standard Load Balancer created for Azure AKS Cluster

* Frontend IP Configuration
* Load Balancing Rules

Azure Public IP

kubectl run my-first-pod --image stacksimplify/kubenginx:1.0.0

# Expose Pod as a Service

kubectl expose pod my-first-pod --type=**LoadBalancer** --port=80 --name=my-first-service

kubectl get service my-first-service -o yaml

# Get Service Info

kubectl get service

kubectl get svc

# Describe Service

kubectl describe service -n dev my-first-service

# Access Application

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

# Sample Commands

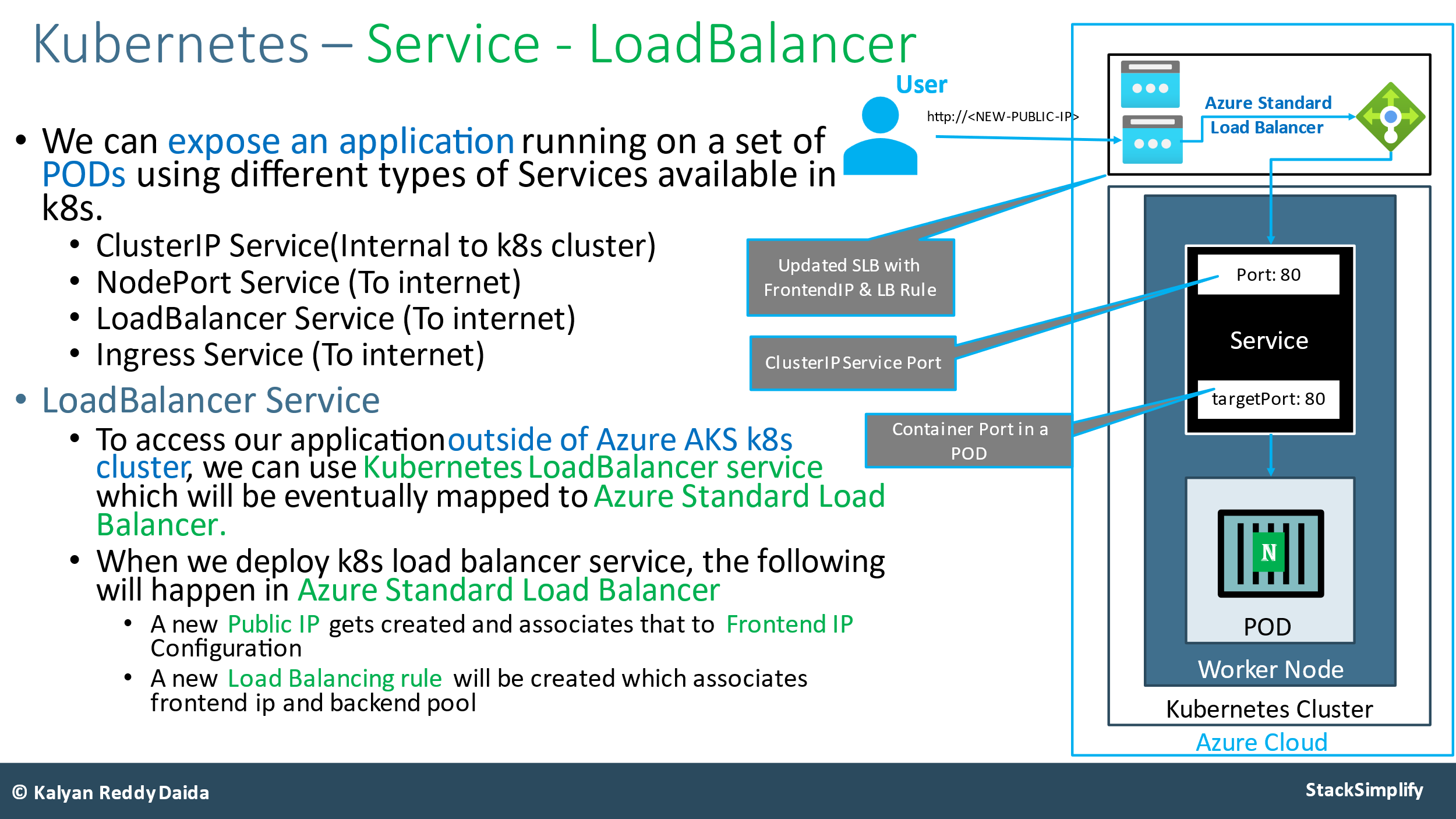
kubectl exec -it my-first-pod -- env

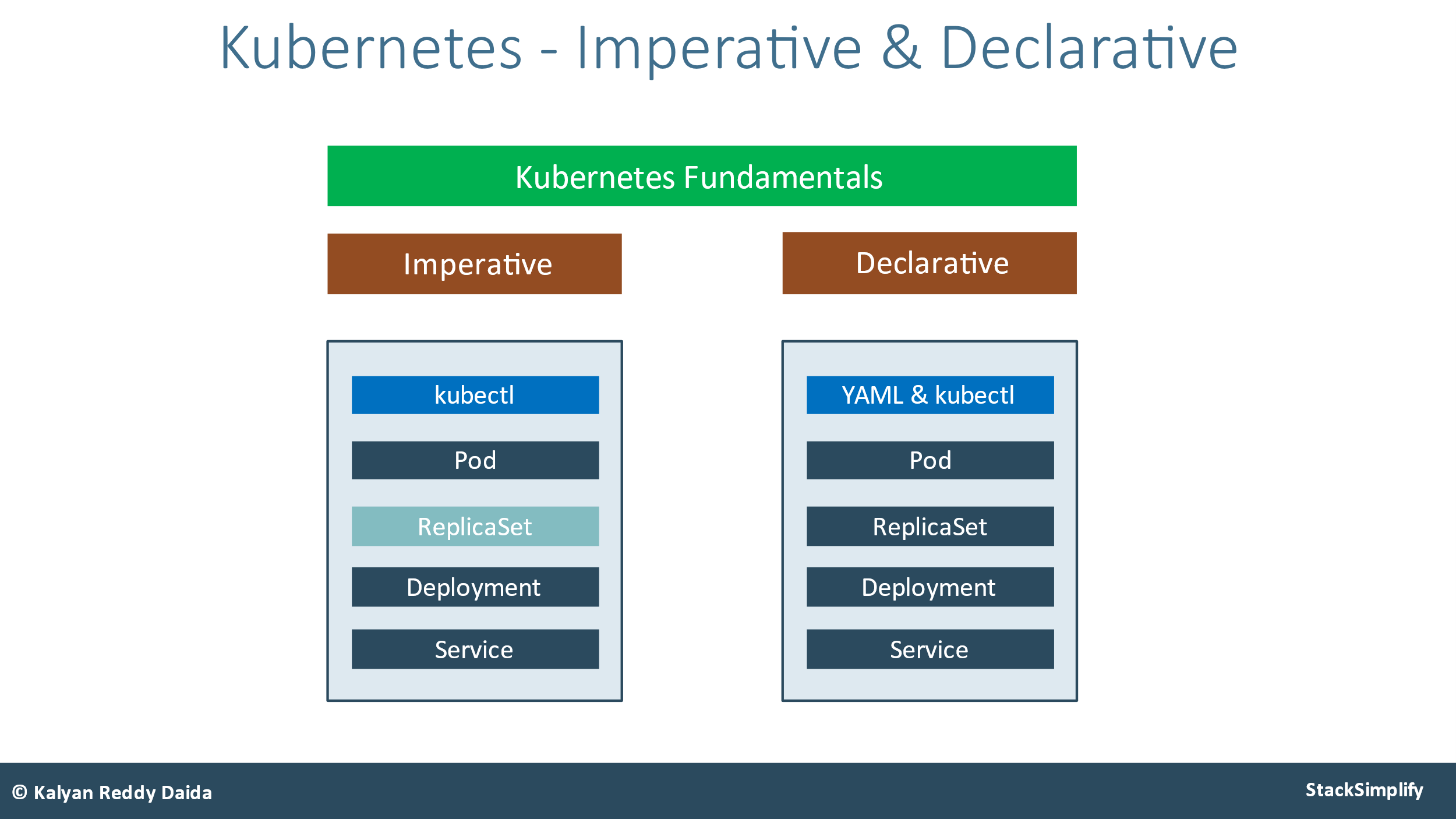
kubectl exec -it my-first-pod -- ls

kubectl exec -it my-first-pod -- cat /usr/share/nginx/html/index.html

# Delete Services

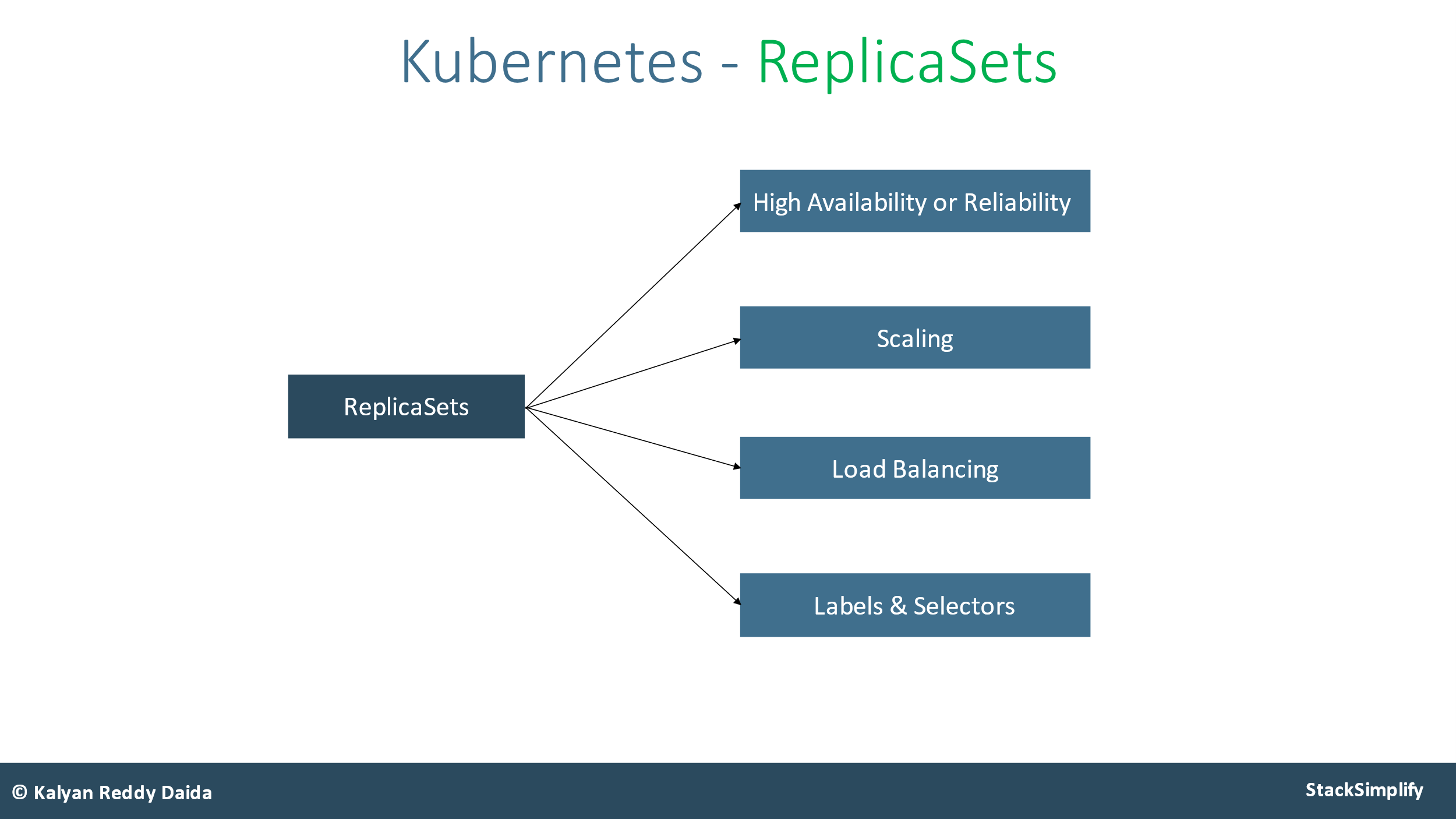
kubectl delete svc my-first-service





**ReplicaSet:**

|  |
| --- |
| kubectl create -f replicaset-demo.yml  kubectl get replicaset  kubectl get rs  kubectl describe rs/<replicaset-name>  kubectl describe pod <pod-name>  kubectl get pods -o wide  kubectl get pods <pod-name> -o yaml -🡪 Verify the owner reference of the pod.  # Expose ReplicaSet as a Service  kubectl expose rs <ReplicaSet-Name> --type=LoadBalancer --port=80 --target-port=8080 --name=<Service-Name-To-Be-Created> |



* A ReplicaSet’s purpose is to maintain a **stable** set of replica Pods running at any given time.
* If our application crashes (any pod dies), replicaset will **recreate** the pod immediately to ensure the configured number of pods running at any given time.
* Reliability
* High Availability

**Deployment**:

|  |
| --- |
| # Create Deployment  kubectl create deployment <Deplyment-Name> --image=<Container-Image>  or  kubectl apply -f 02-deployment-definition.yml  # Verify Deployment  kubectl get deployments  kubectl get deploy  # Describe Deployment  kubectl describe deployment <deployment-name>  # Verify ReplicaSet  kubectl get rs  # Verify Pod  kubectl get po  # Scale Down the Deployment  kubectl scale --replicas=2 deployment/my-first-deployment  Expose Deployment as a Service  kubectl expose deployment <Deployment-Name> --type=LoadBalancer --port=80 --target-port=80 --name=<Service-Name-To-Be-Created>  http://<External-IP-from-get-service-output>  # Update Deployments |

Update Deployments:

# 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

Verify Rollout Status (Deployment Status)

# Verify Rollout Status

kubectl rollout status deployment/my-first-deployment

kubectl rollout history deployment/<Deployment-Name>

kubectl edit deployment/<Deployment-Name> --record=true

# List Deployment History with revision information

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

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

kubectl rollout undo deployment/my-first-deployment

# Rollback Deployment to Specific Revision

kubectl rollout undo deployment/my-first-deployment --to-revision=3

kubectl rollout restart deployment/my-first-deployment

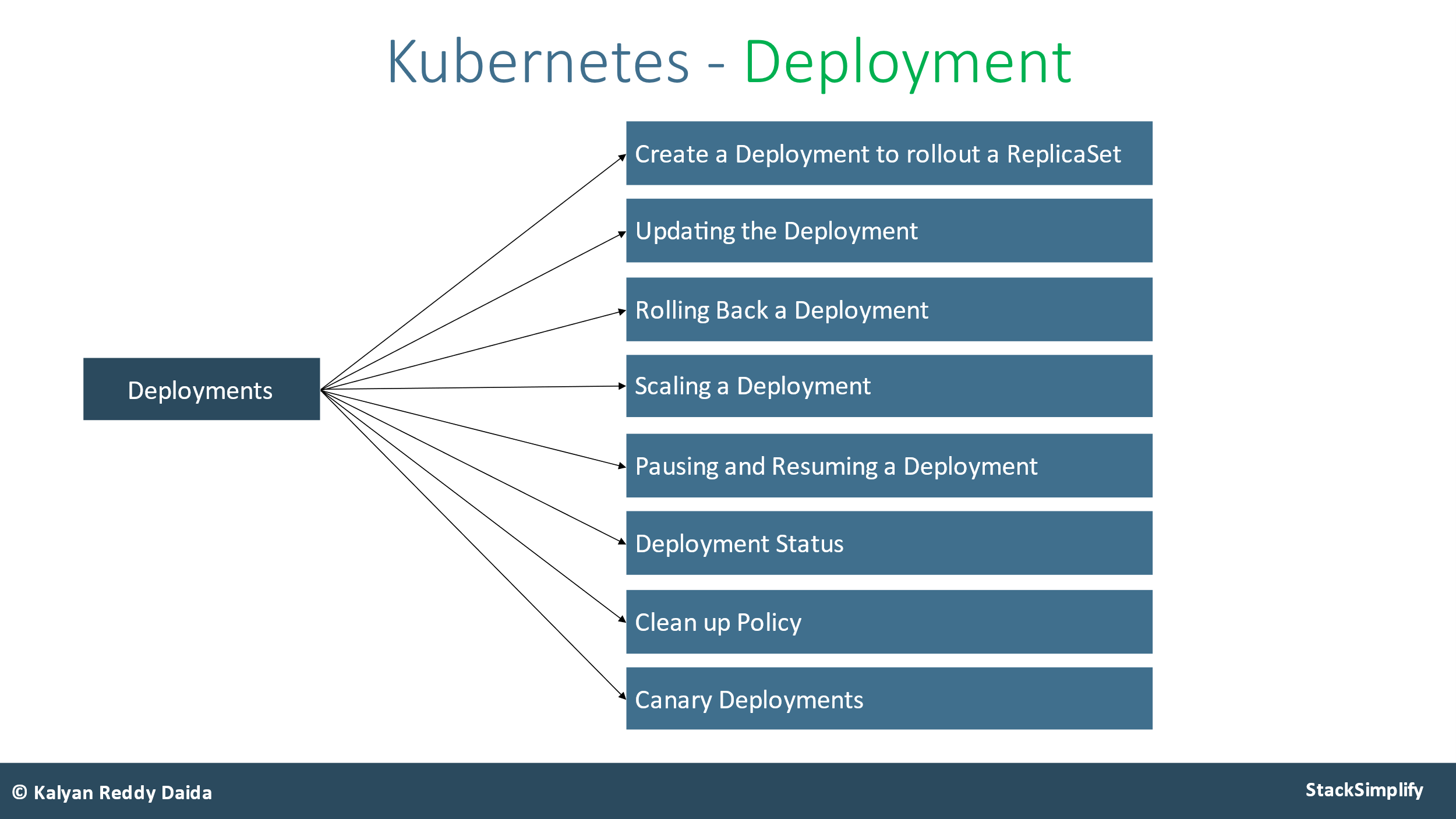
# Pause & resume the Deployment

kubectl rollout pause deployment/<Deployment-Name>

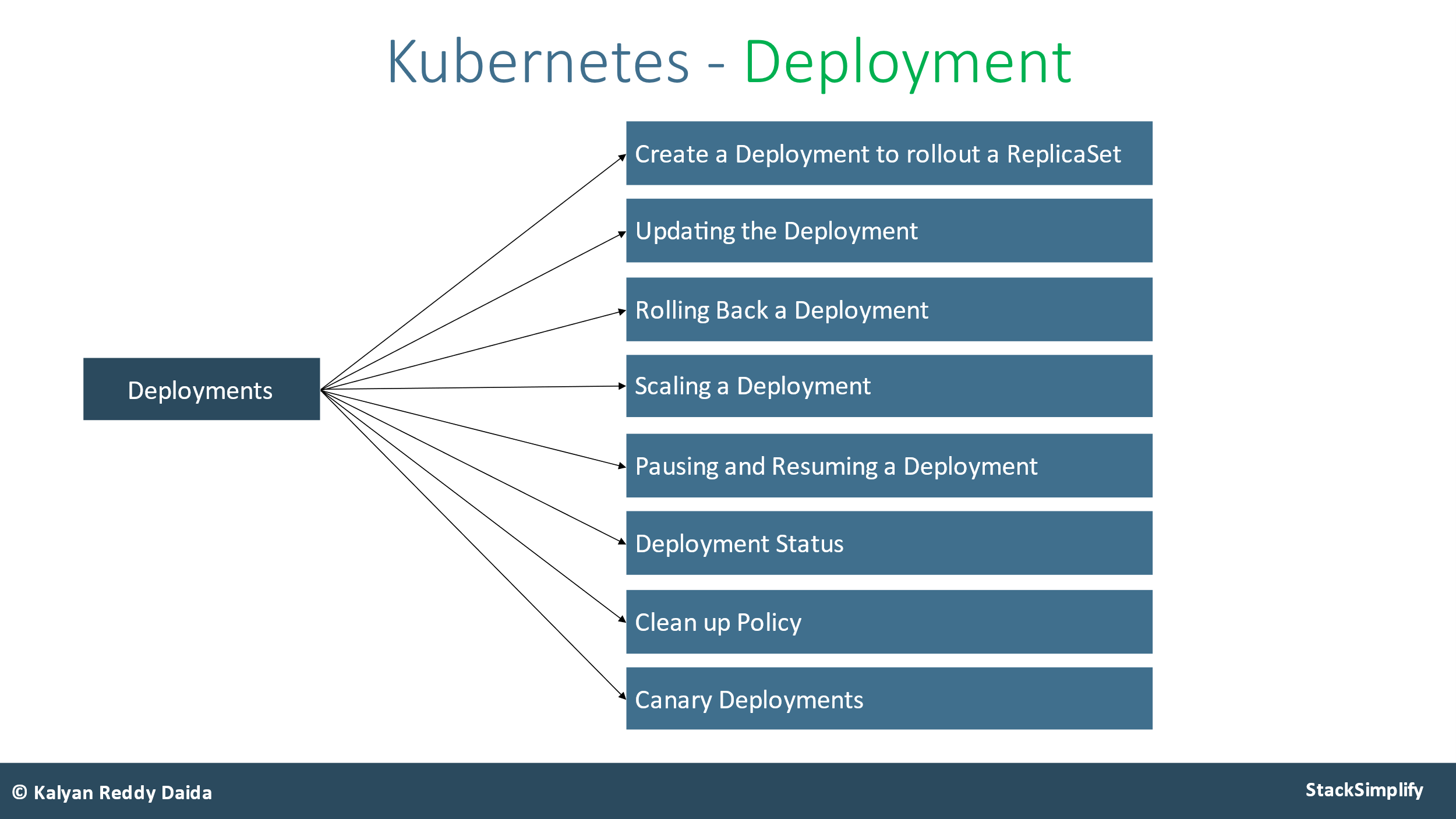
kubectl rollout resume deployment/<Deployment-Name>

# Make one more change: set limits to our container

kubectl set resources deployment/my-first-deployment -c=kubenginx --limits=cpu=20m,memory=30Mi

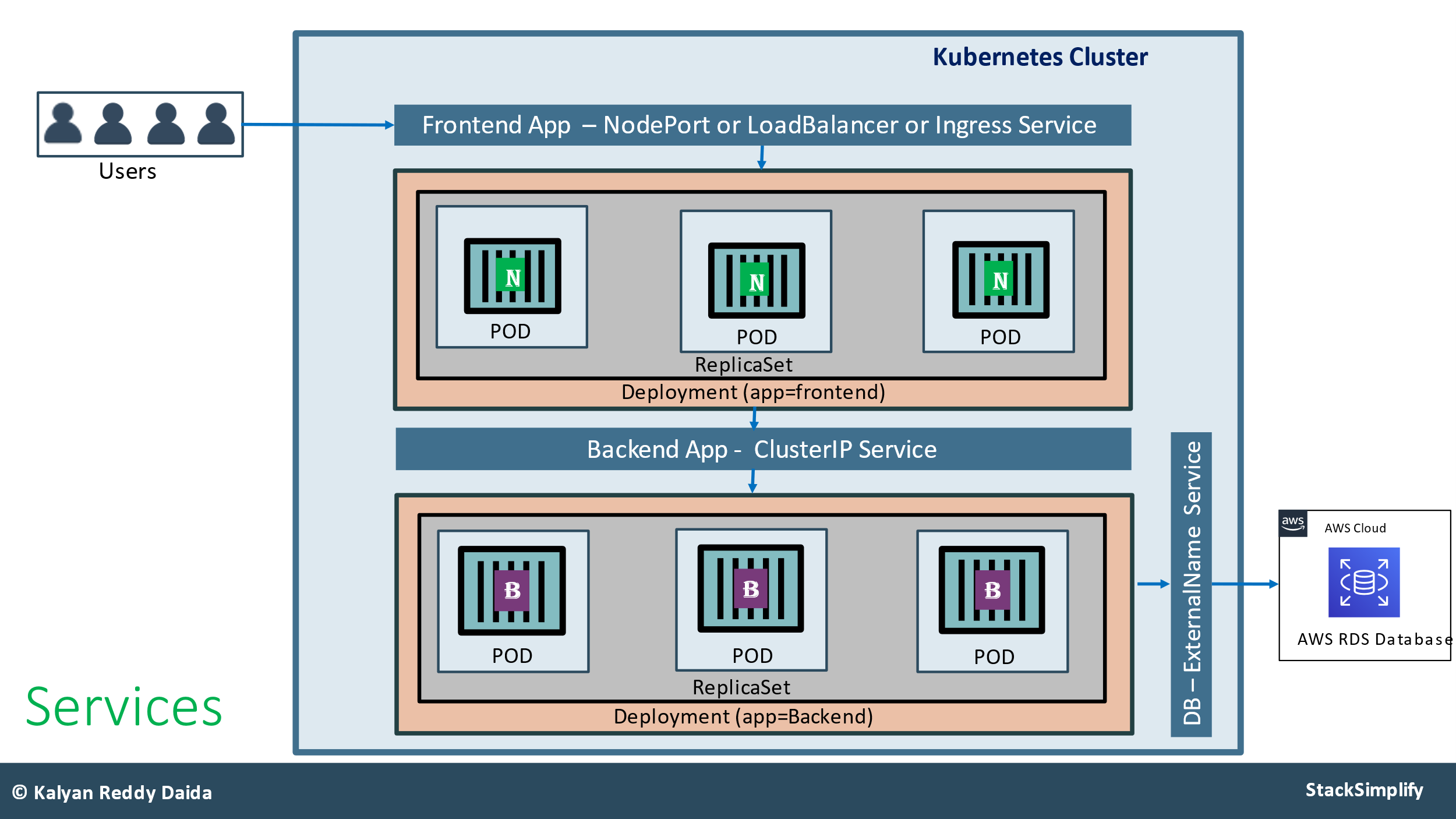






**Service :** In Kubernetes, a Service is a method for exposing a **network application** that is running as one or more Pods in your cluster.

1. **ClusterIp 🡪default**
2. **NodePort**
3. **LoadBalancer**
4. **ExternalName**
5. **Ingress**



**1.Namespaces:**

* It is a mechanism that enables you to organize resources.
* A namespace **isolates** the resources from the resources of other namespaces.

Initial Namespaces

1. kube-**system**: System processes like Master and kubectl processes are deployed in this namespace;
2. kube-**public**: This namespace contains publicly accessible data like a configMap containing cluster information.
3. kube-**node-lease**: This namespace is the heartbeat of nodes.
4. **default**: This is the namespace that you use to create your resources by default.

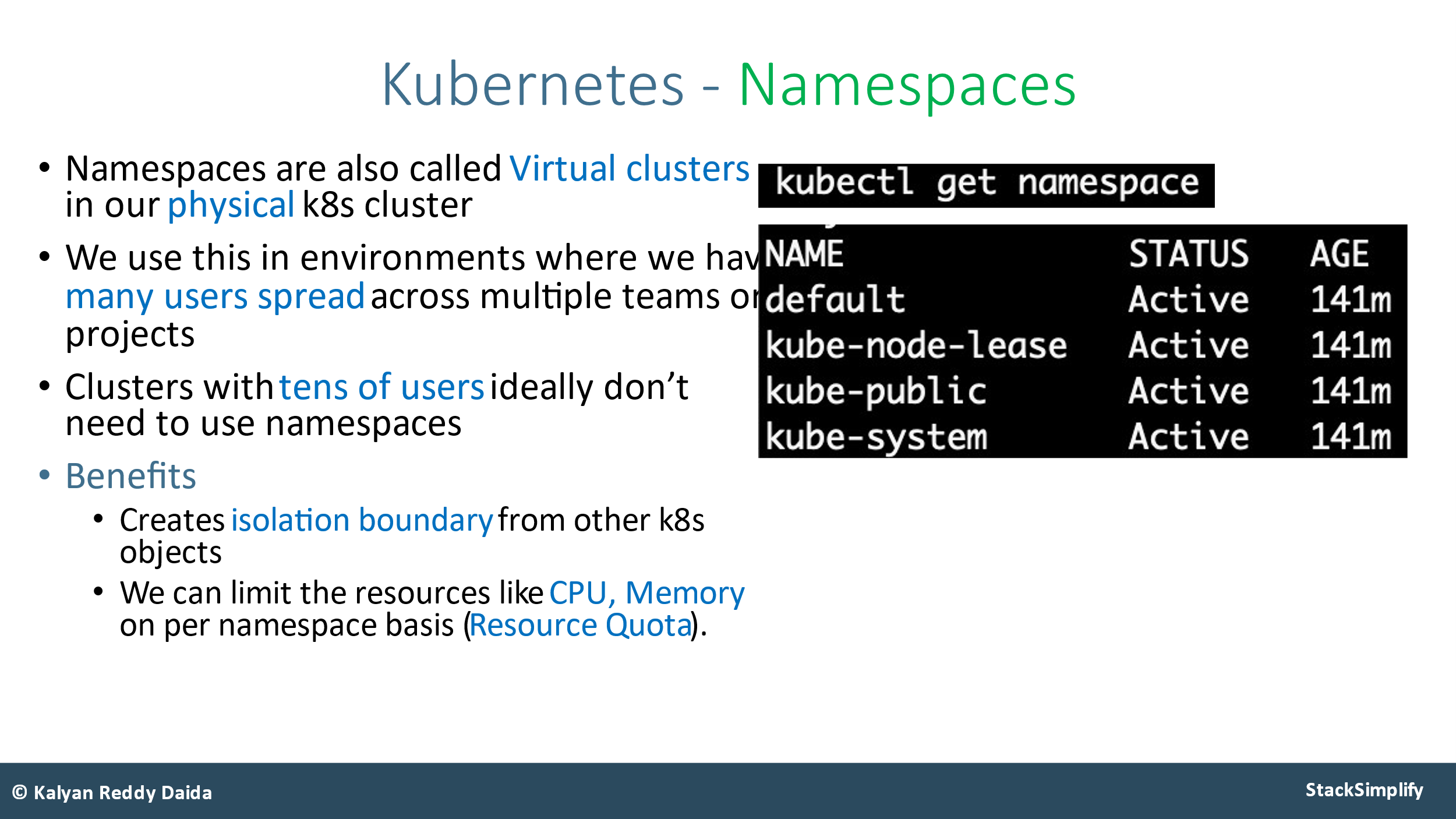
kubectl get ns

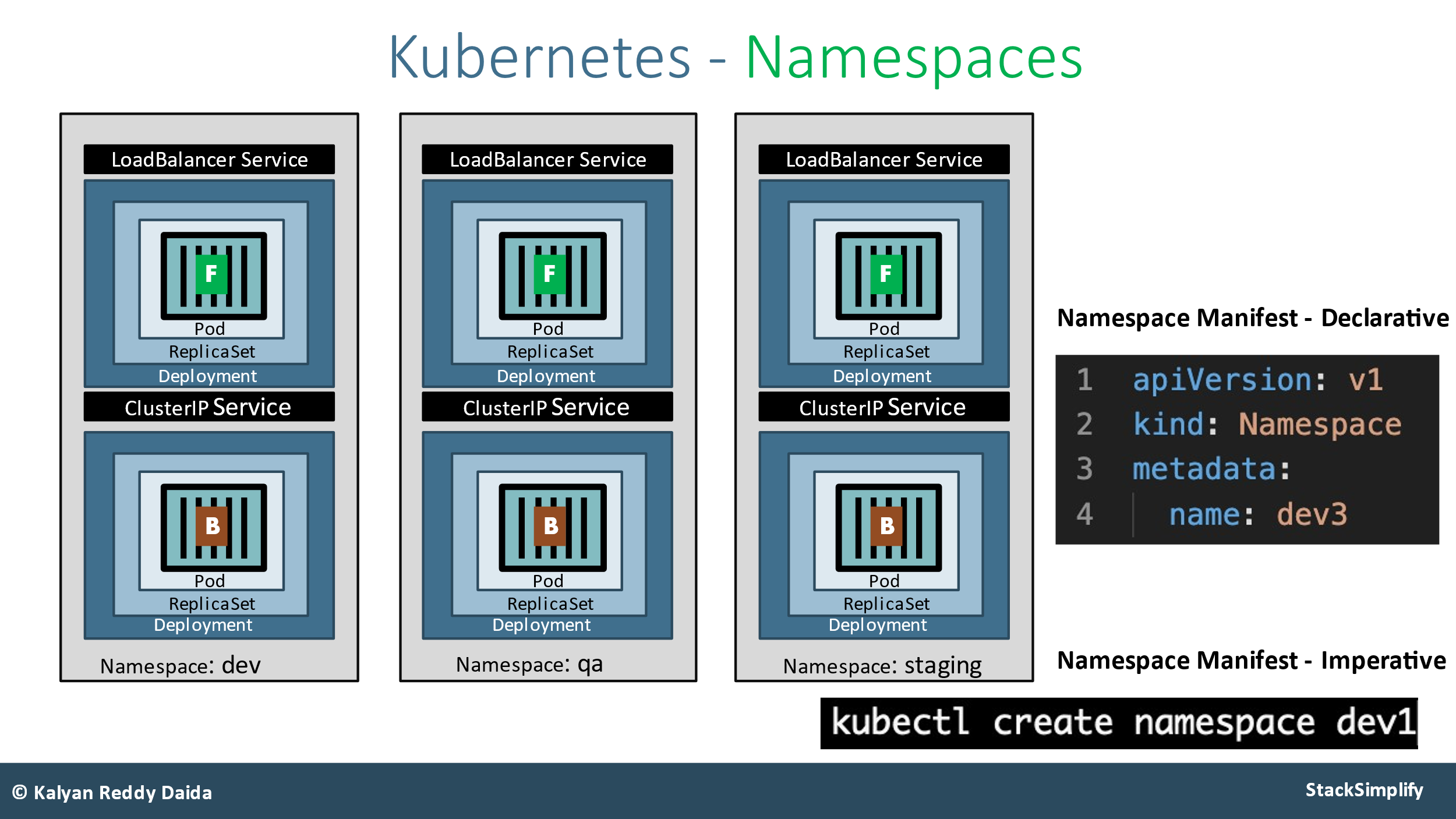
# List Pods from all namespaces

kubectl get pods --all-namespaces

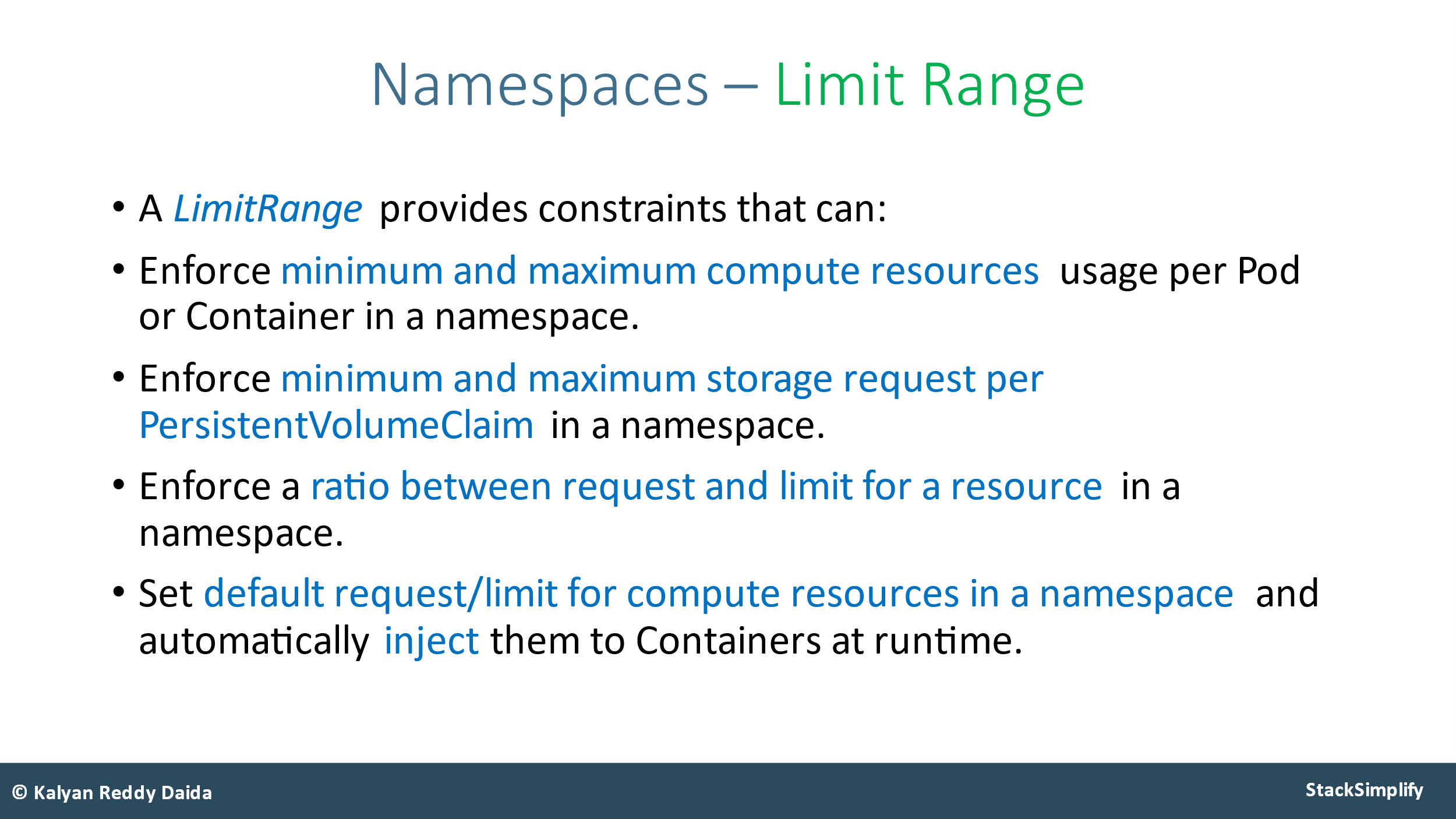
# List all k8s objects from Cluster Control plane

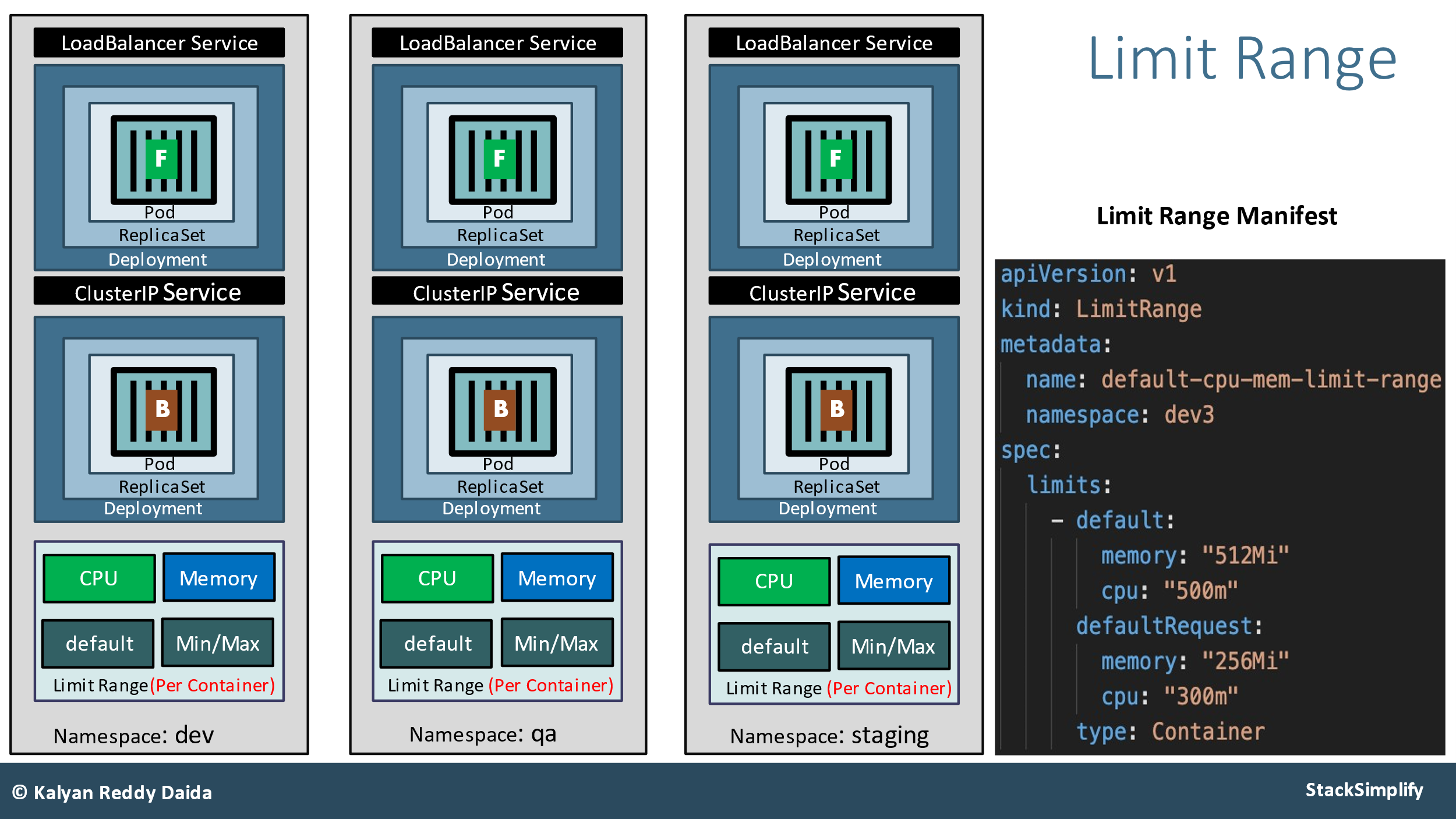
kubectl get all --all-namespaces



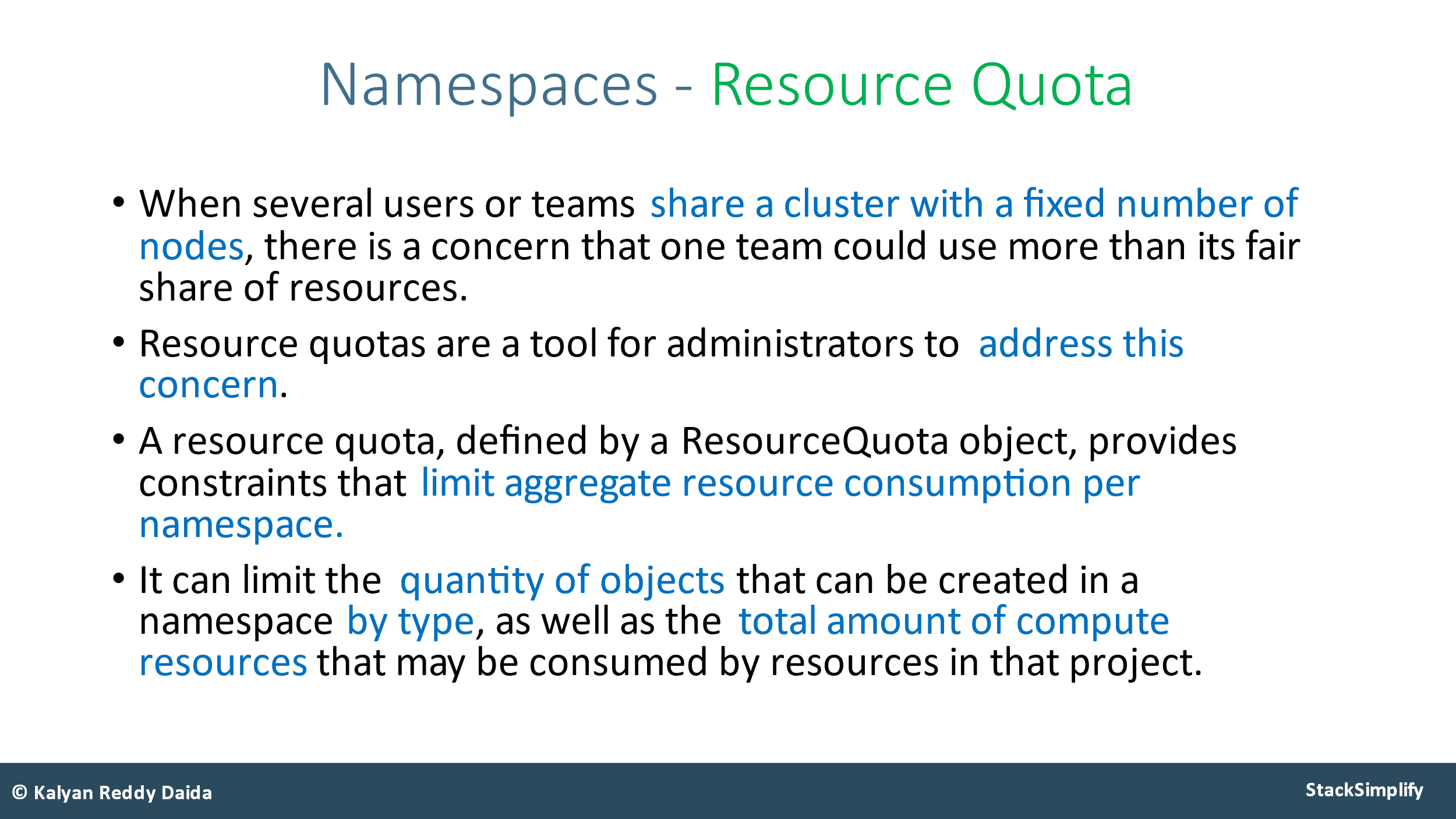


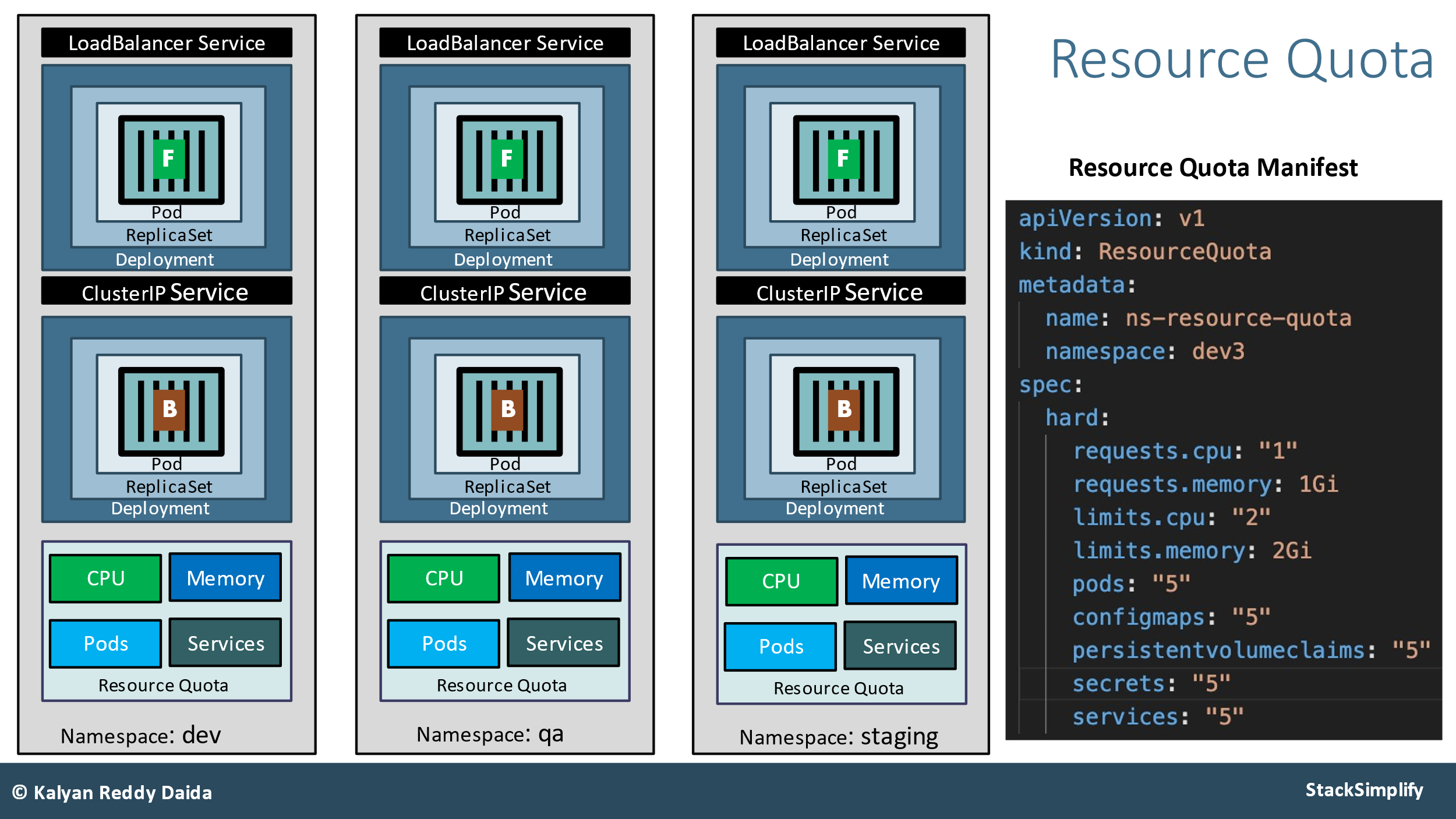
**Namespaces Limit Range:**





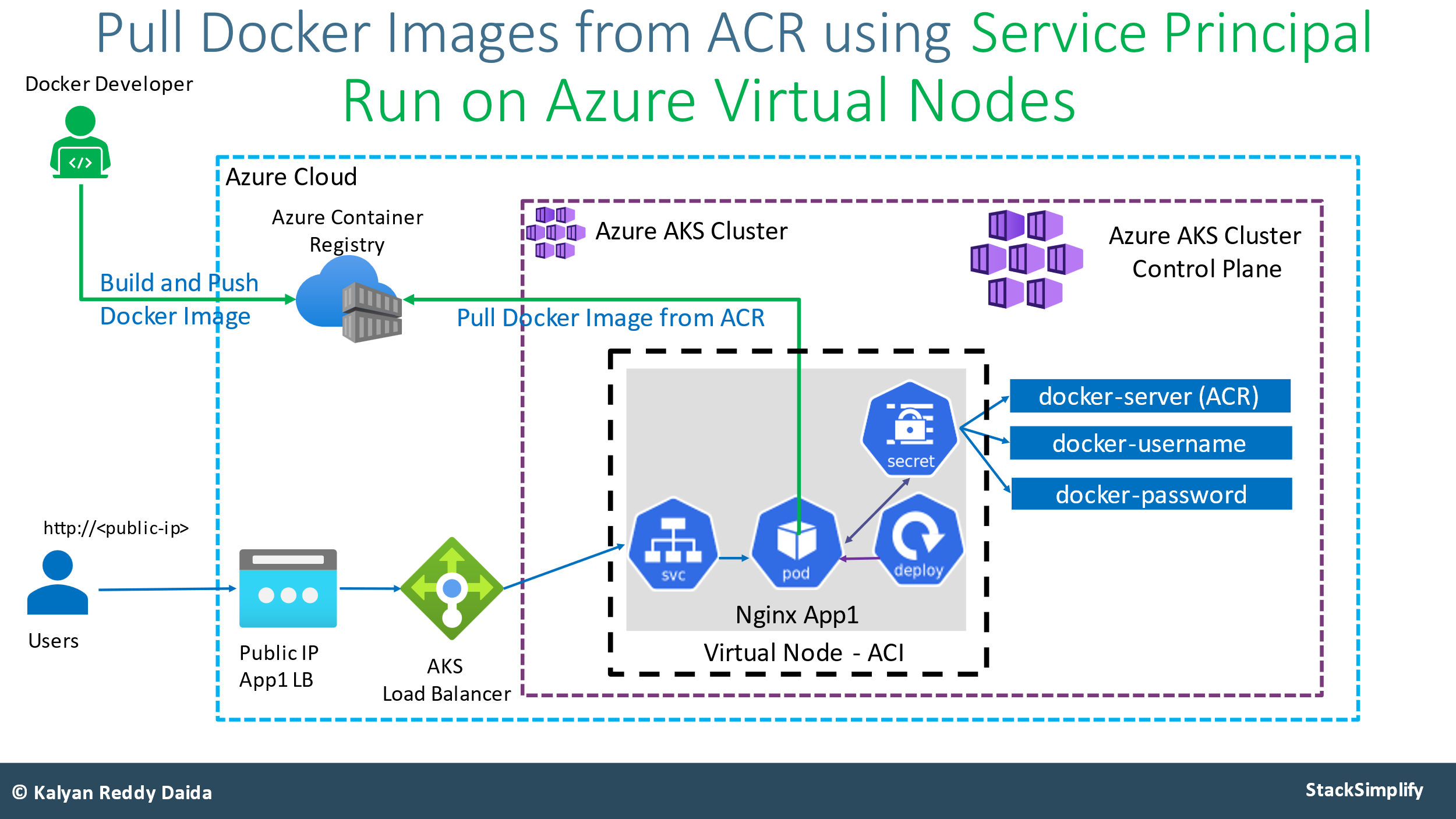
**Namespaces Resource Quota:**



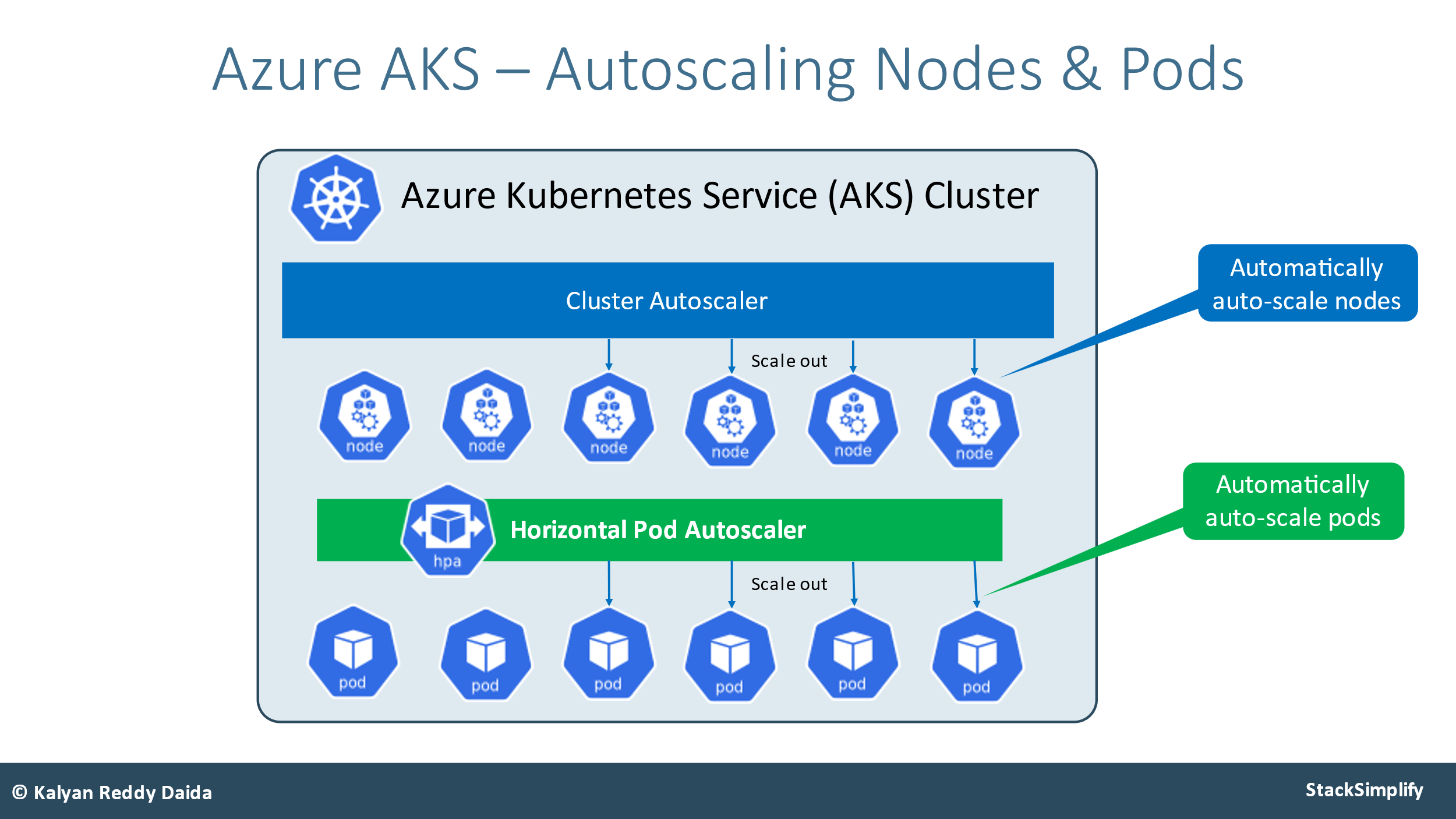


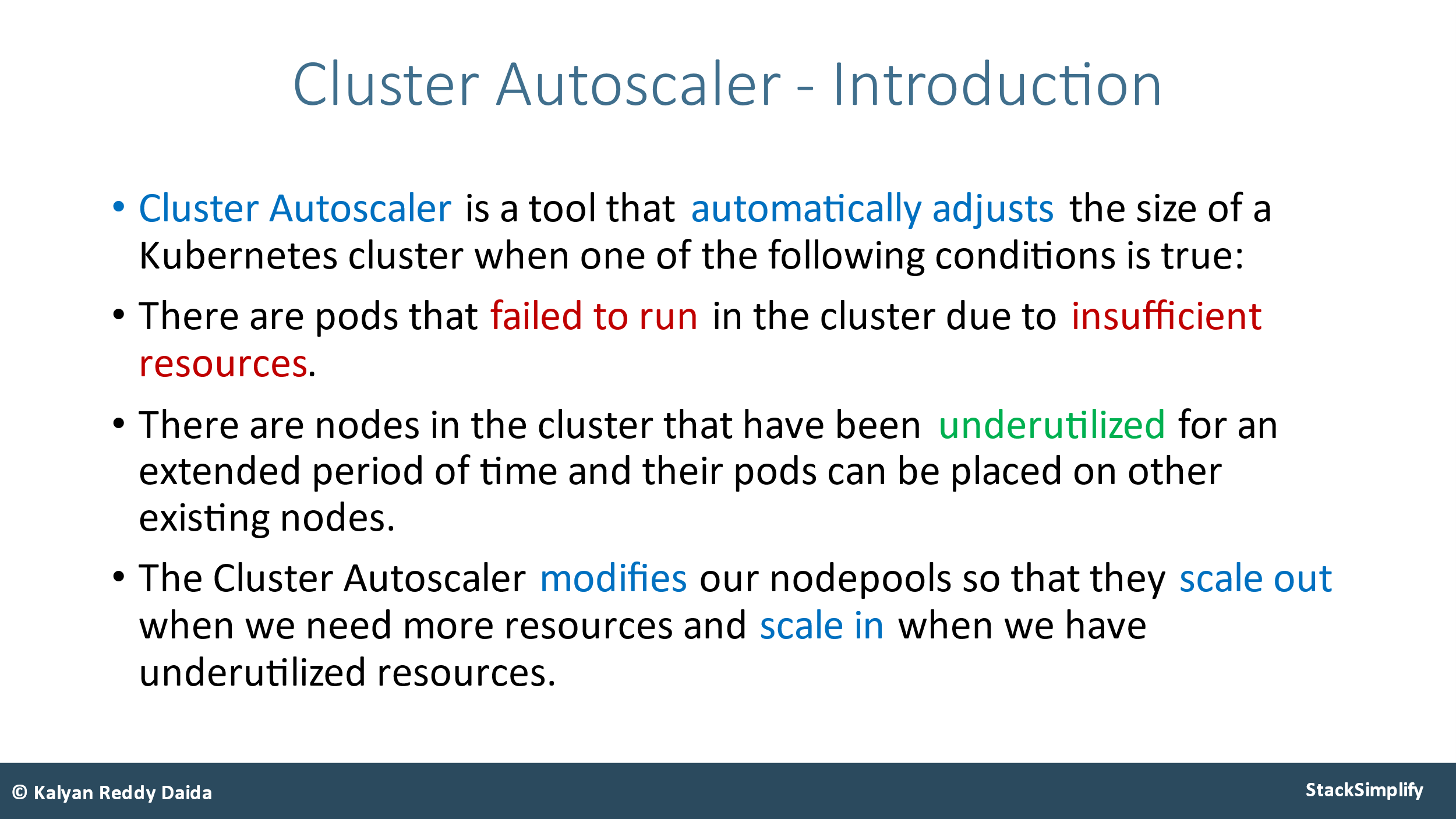
**Azure Container Registry (ACR)**

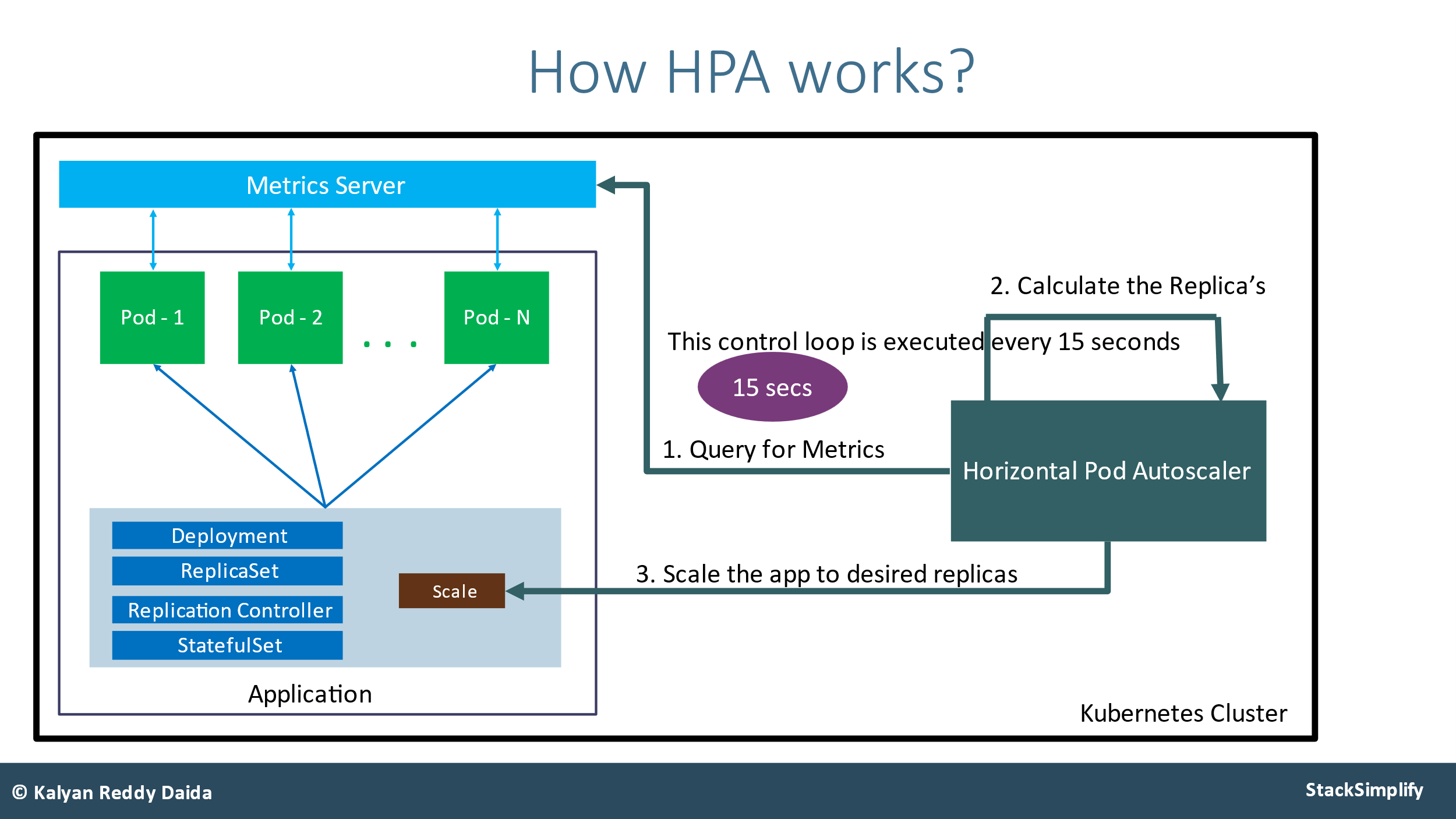
**Pull Images from ACR using Service Principal**

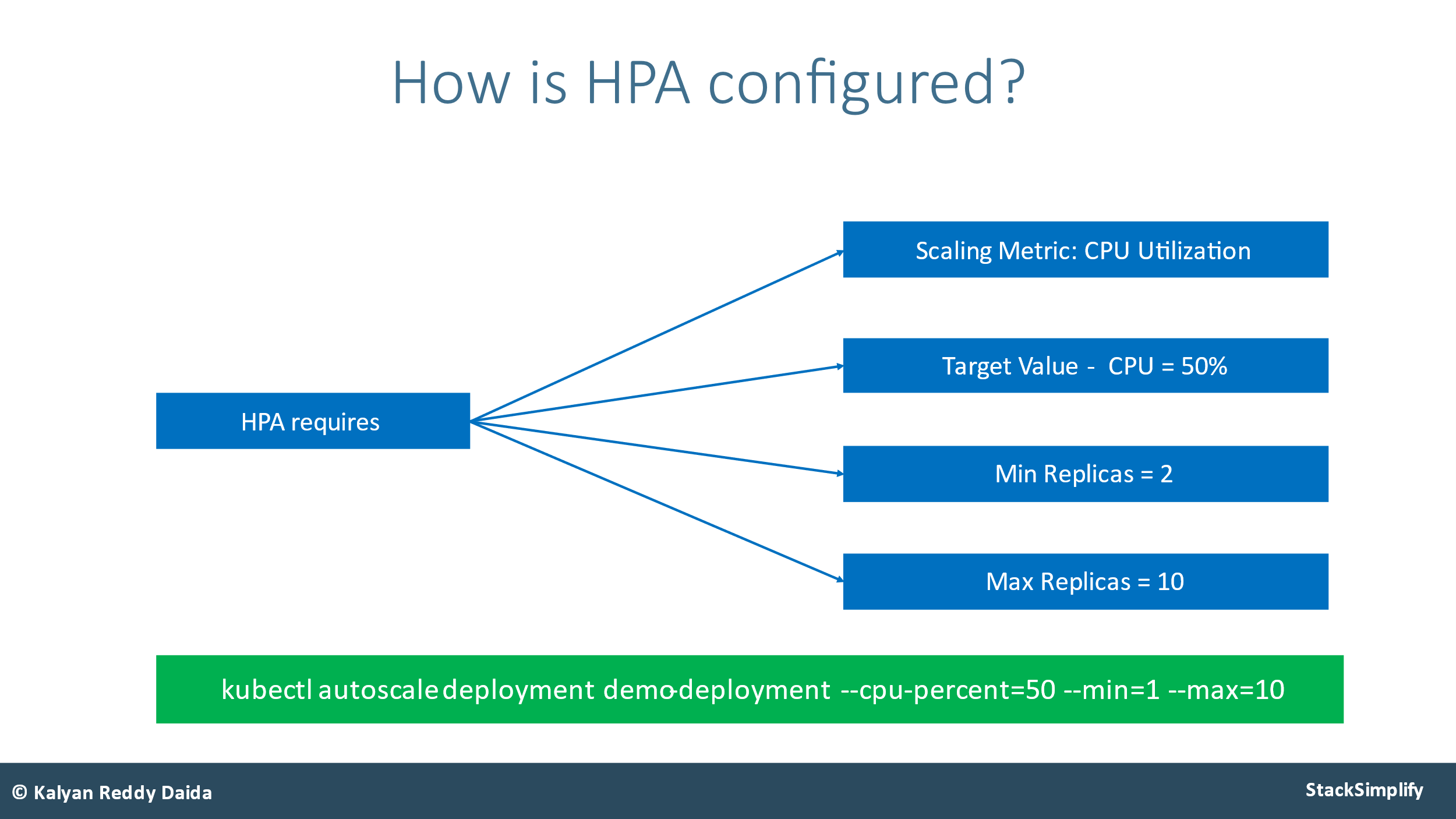


**Autoscaling Nodes & Pods**



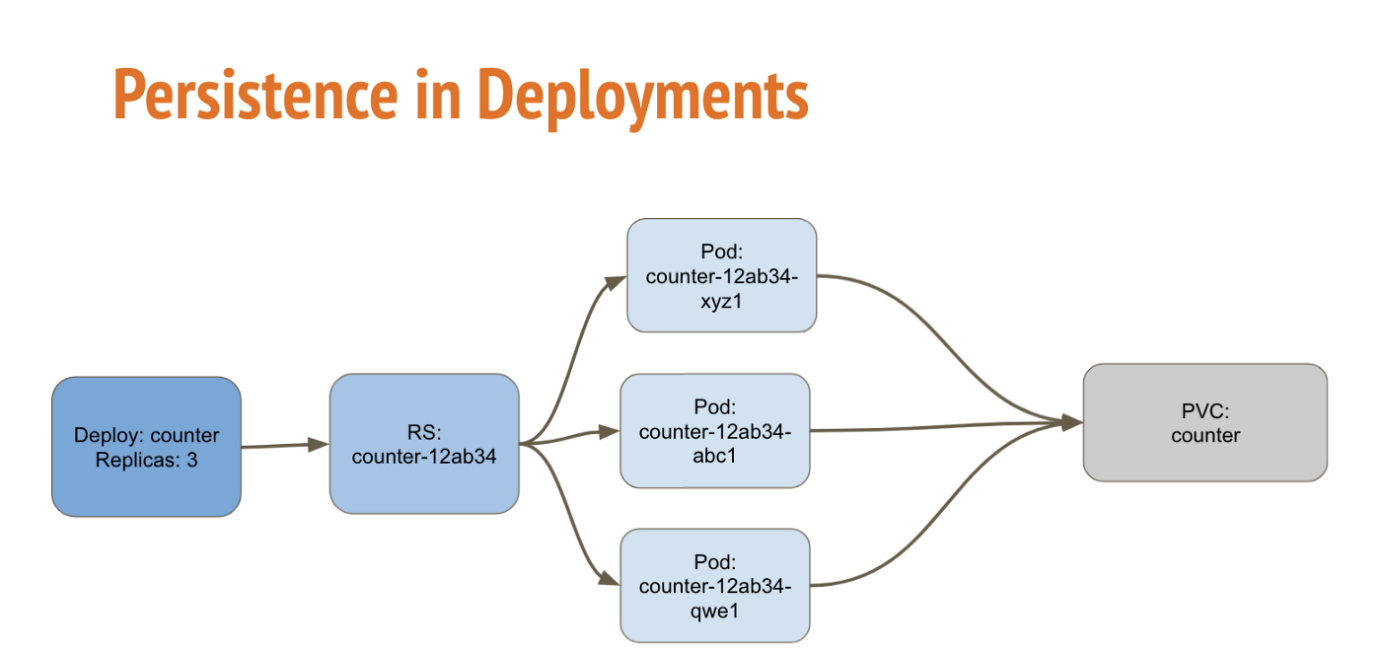






**Deploymets:**

Deployment is a resource to deploy a stateless application, if using a PVC, all replicas will be using the same Volume and none of it will have its own state.

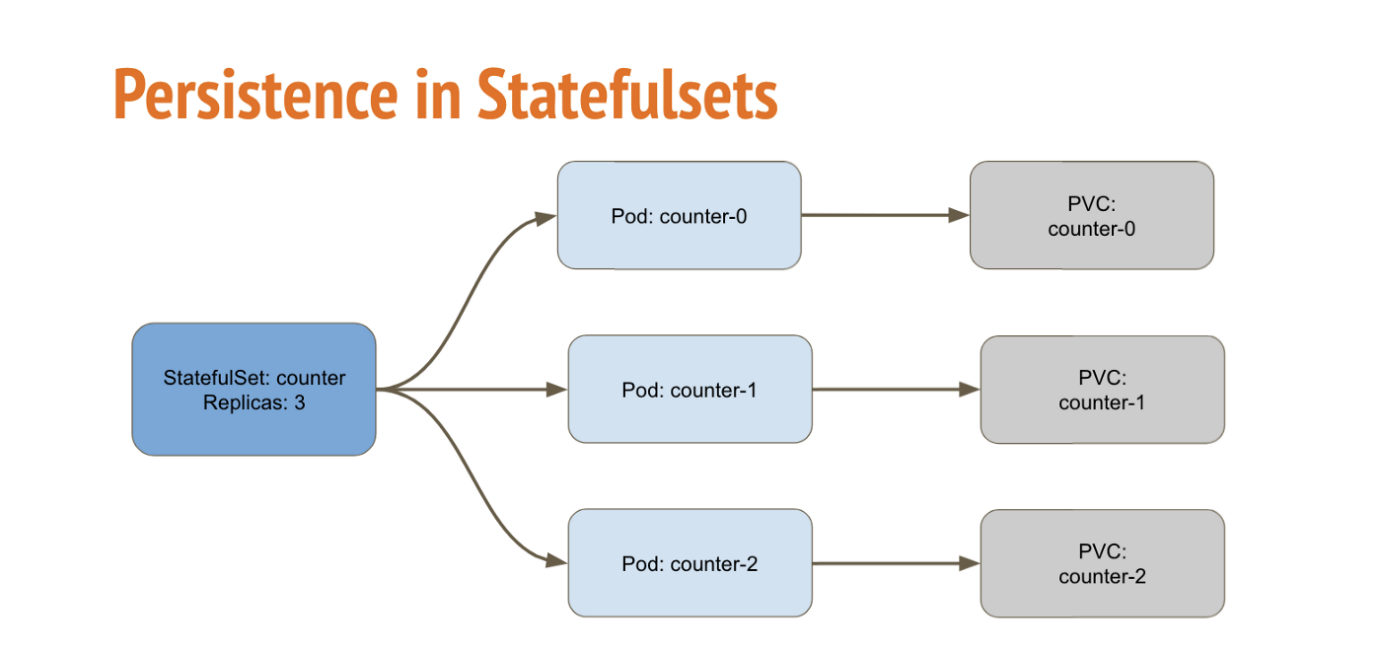
[](https://i.sstatic.net/oGm0m.png)

**Statefulsets** is used for Stateful applications, each replica of the pod will have its own state, and will be using its own Volume.

**stateful** applications require backing storage. Store the data, if pod is delete pv in not delete

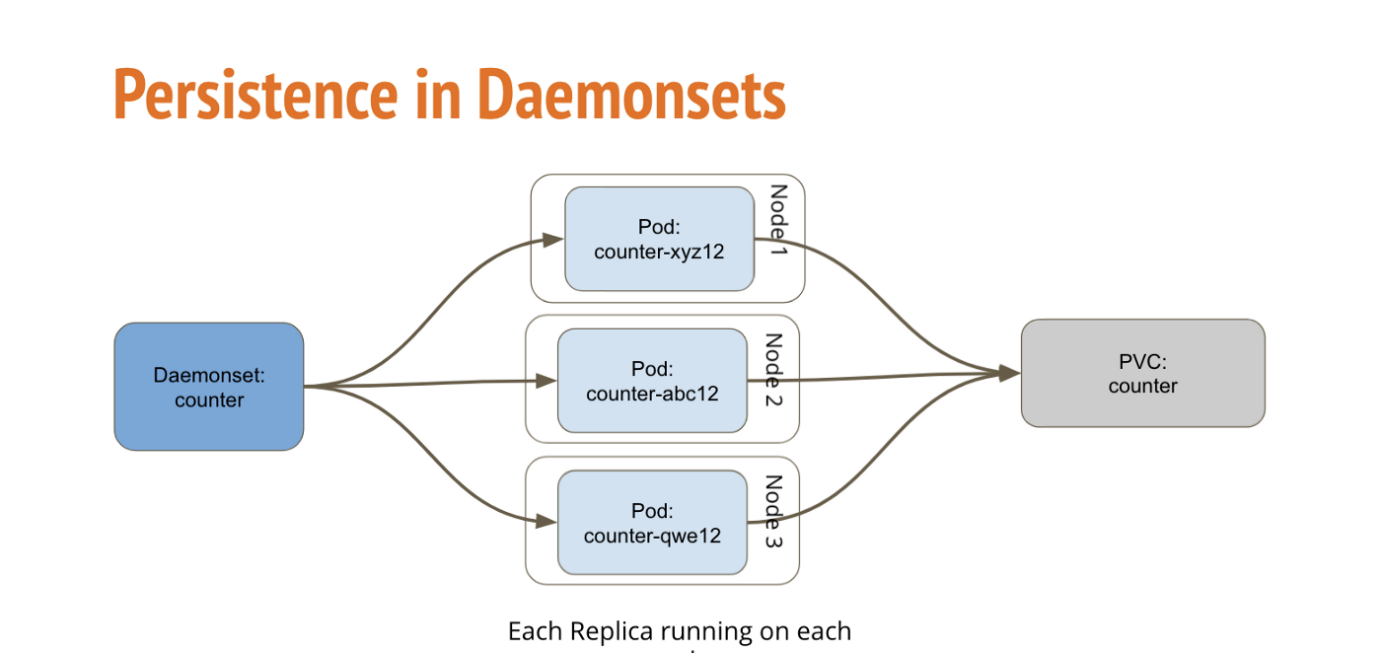
**stateless** applications don’t “store” data. On the other hand, if pod is deleted pv will delete

* Deployed on service 🡪Ip address not present shows like stateful set, is called Headless service
* Pods name shouldn’t be change, if we delete sequentially delete
* Pods have dedicated ip addresses, pos can communicate to one pod to another pod
* This mainly used In database side applications

[](https://i.sstatic.net/5pfeD.png)

**DaemonSet** is a controller similar to **ReplicaSet** that ensures that the pod runs on all the nodes of the cluster. If a **node is added/removed** from a cluster, DaemonSet **automatically** **adds/deletes** the pod.

Widley used in **system service** or **monitoring agents**



**Jobs:** create 1 or more pods, excite the task or command inside the pod status should be completion

**Types of Kubernetes Jobs**

1. **Non-Parallel Jobs**: are designed to run a single task to completion
2. **Multiple Parallel Jobs:** if we need to process **a large batch of data in** single Job to time-consuming multiple parallel jobs to divide the **workload** among several **pods**
3. **Parallel Jobs with Fixed Number of Completions:** create a parallel job with a fixed number of completions, and configure it to run the **test suite** and all tests are run the **required** number of times.

**Ex: command: ["echo", "Hello, World!"]**

**How to Use Kubernetes Jobs with Code Examples**

1. **Creating and Configuring a Job**
2. **Schedule a Job**
3. **Running a Job**
4. **Clean Up Finished Jobs Automatically**