Kubernetes – Overview

KOENIG
step forward

Kubernetes Architecture

KOENIG
step forward

Cloud-Native Principles

Cloud-native principles are as follows:

- Container packaged: Isolated unit of work that does not require OS dependencies
- Dynamically managed: Actively scheduled and managed by an orchestration process
- Microservice oriented: Loosely coupled from other dependencies

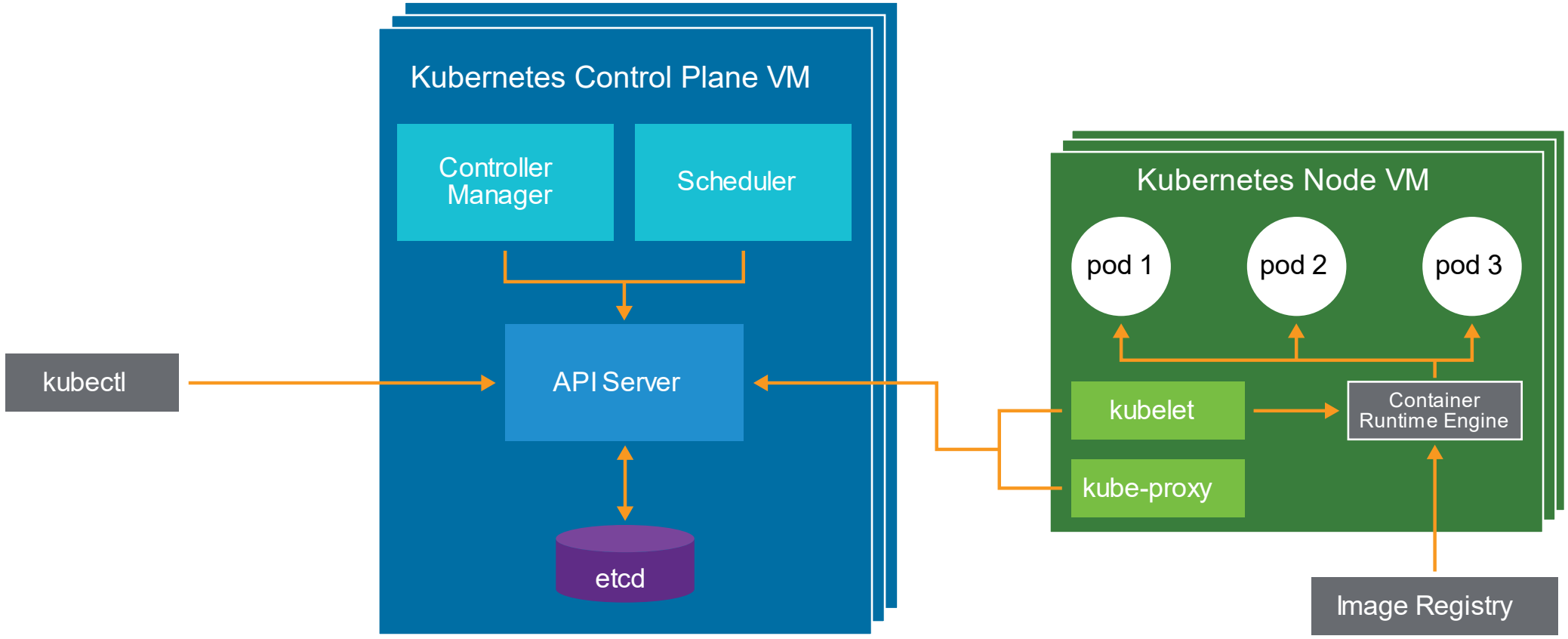
About Kubernetes

Kubernetes is an open-source cluster management tool:

- Automates the deployment, management, and scaling of applications
- Managed by Cloud Native Computing Foundation (CNCF)
- Actively developed by a well-supported community, including former Borg engineers
- Lessons learned from Borg in production for more than a decade
- Can run on bare metal, hypervisors, or on various cloud providers

Kubernetes Architecture

The Kubernetes architecture includes several components.



Problems Solved by Kubernetes

With Docker, containers are managed on a single container host. Managing multiple containers across multiple container hosts creates issues in the following areas:

- Managing large numbers of containers
- Restarting failed containers
- Scaling containers to meet capacity
- Networking and load balancing

Kubernetes provides an orchestration layer to solve these issues.

About Manifests

In Kubernetes, manifest files declare the desired state of objects.

Manifests have the following properties:

- YAML format
- Declarative configuration
- Desired API primitives

Kubernetes manages the creation of the requested primitives.

About Pods

A pod is a set of one or more tightly coupled containers.

It is the smallest unit of work in Kubernetes.

Containers in a pod live and die together.

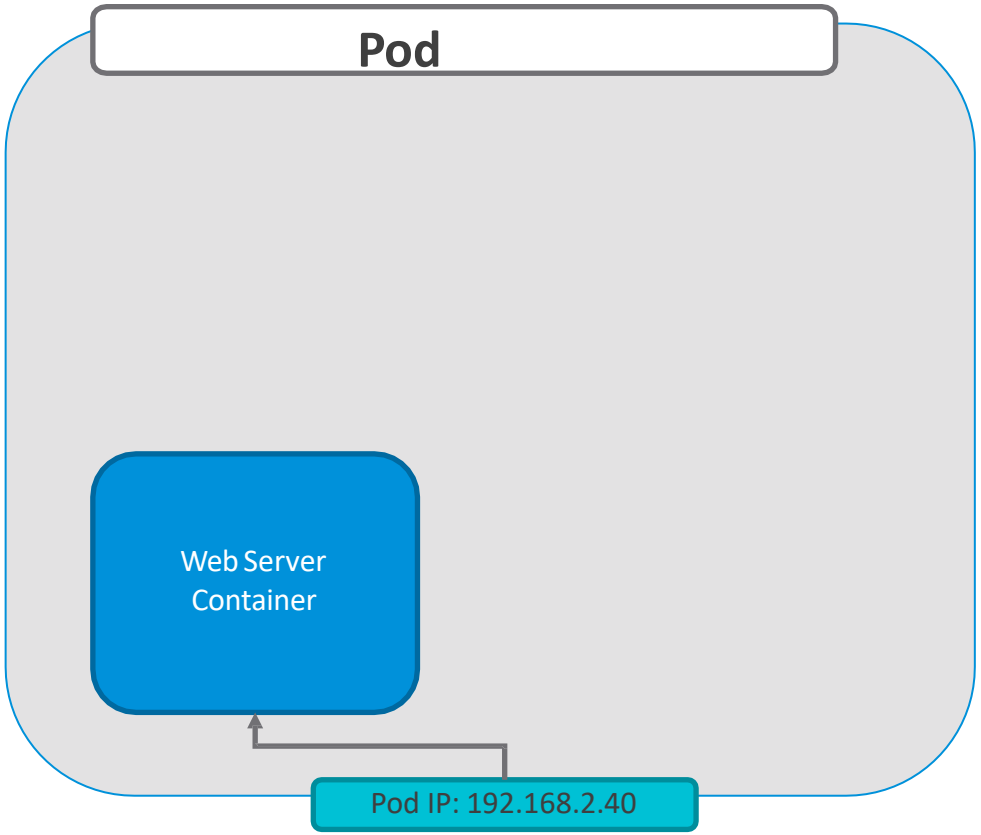
Create a pod example:

- The pod has a label app with the value nginx.
- The pod runs the image nginx with version 1.7.9.
- The pod listens on port 80.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-6dd86-nlrhx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
      - containerPort: 80
```

Pod-to-Container Relationship

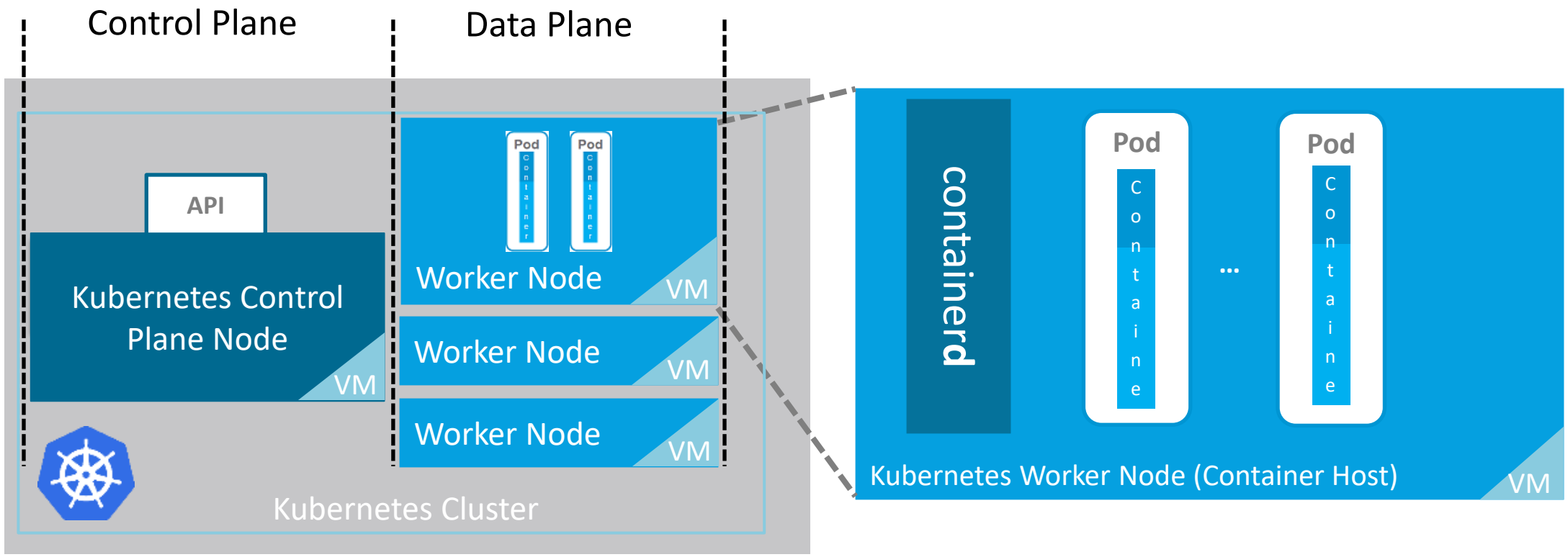
A Kubernetes pod is created to run a container. Additional containers can be defined.



Kubernetes Building Blocks

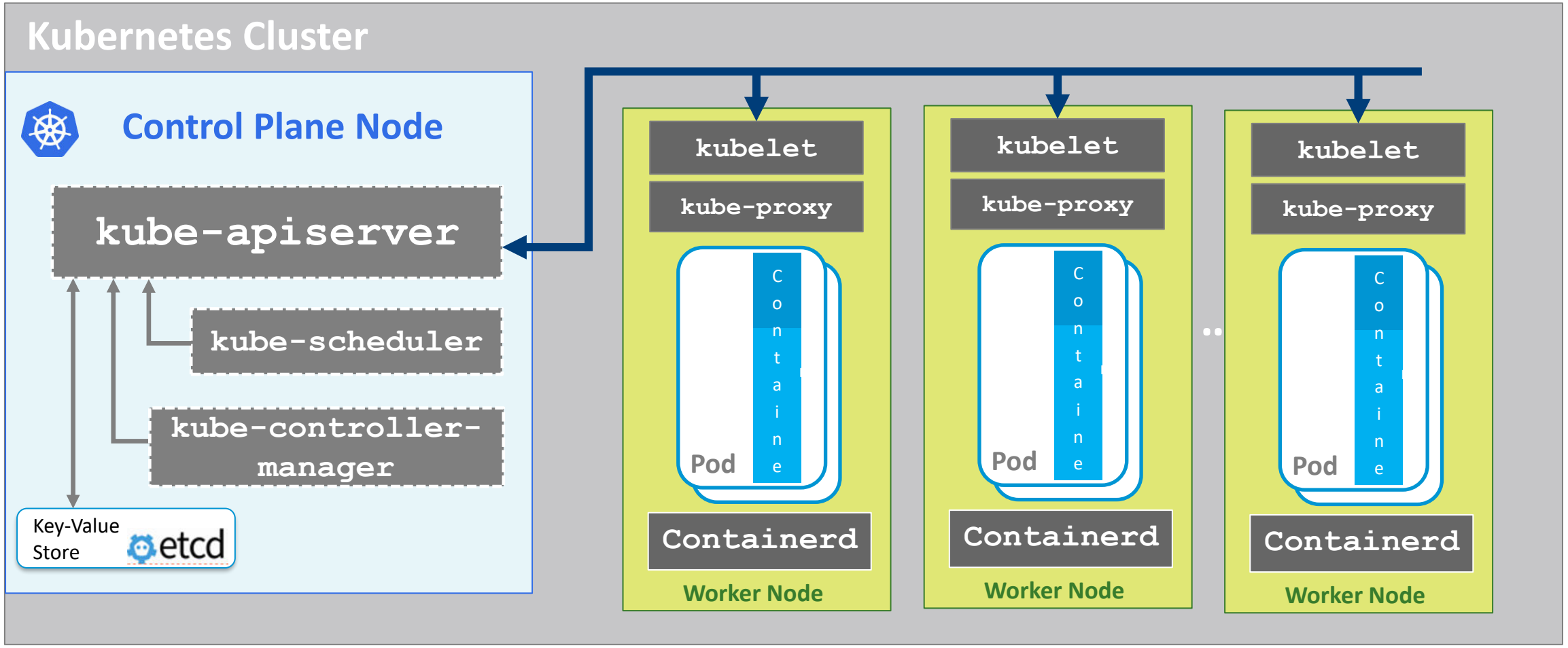
The building blocks of Kubernetes are pods, nodes, and clusters:

- Pod: Containers are encapsulated in pods.
- Node: A container host, for example, a virtual machine running Docker engine.
- Cluster: A set of worker nodes (data plane) that are managed by a control plane node.



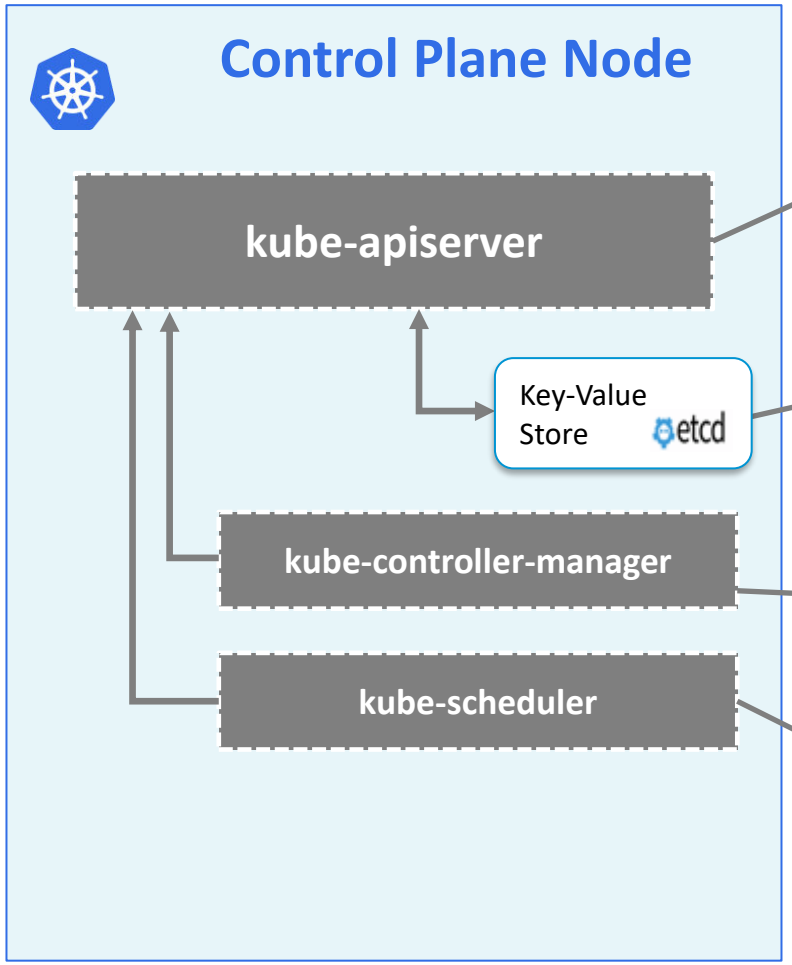
Kubernetes Cluster Architecture

A Kubernetes cluster is a set of worker nodes managed by a control plane node. Typically, a Kubernetes node is a virtual machine.



Kubernetes Control Plane Node

The control plane's components make global decisions about the cluster. They also detect and respond to cluster events.



- Kubernetes front-end control plane
- Provides RESTful interface
- Returns state objects as JSON

- Key-value store
- Leader-based clustering
- Can be clustered across master nodes
- Contains all state known about cluster

- Provides core control loops for platform
- Watches shared state through apiserver
- Makes changes from current to desired

- Policy-based workload scheduler
- Topology aware
- Assists with availability, performance, and capacity
- Affinity and anti-affinity capable

Kubernetes etcd

The key-value store for Kubernetes is called etcd. It is considered the one source of truth for Kubernetes clusters.

It is a distributed key-value store that represents the state of the cluster that components can reference to configure or reconfigure themselves.

It performs the following functions:

- Observes the current cluster state by using the Kubernetes API
- Finds the differences between the desired state and current state
- Stores configuration data that can be used by each of the nodes in the cluster
- Can be distributed across multiple nodes
- Can be used for service discovery

Kubernetes Worker Node

A Kubernetes worker node is typically a virtual machine.

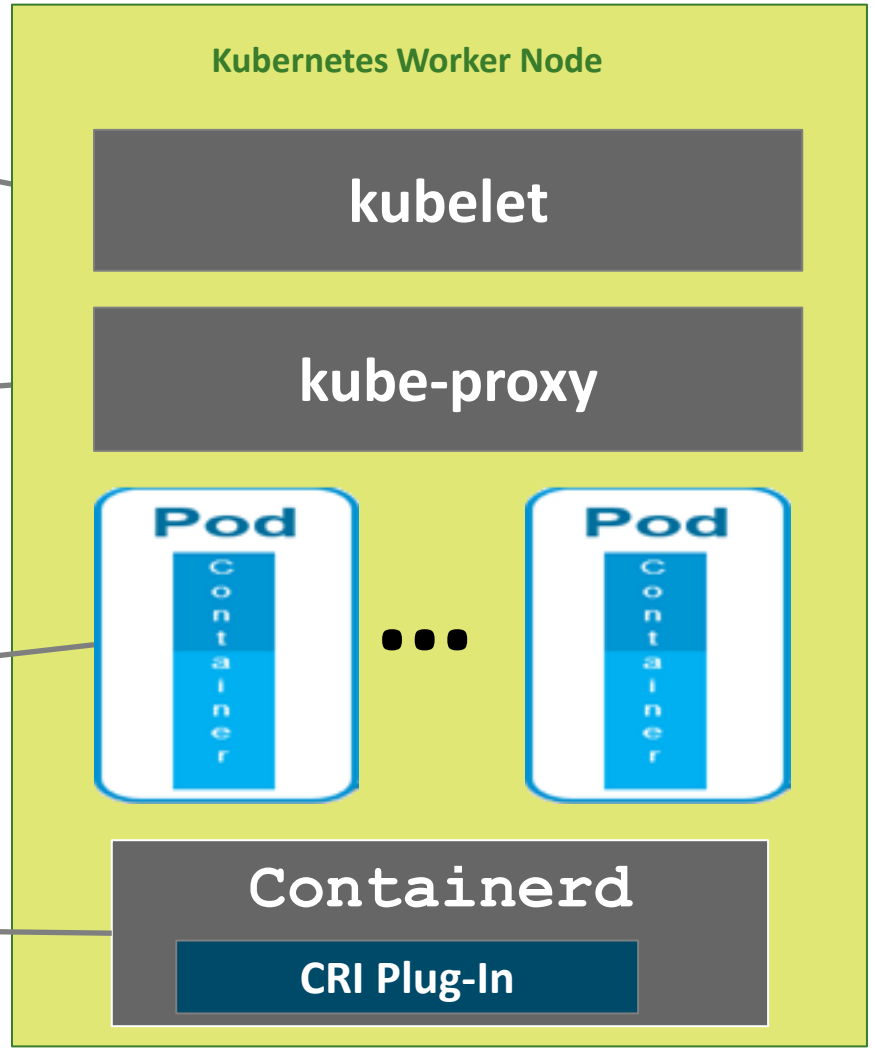
The Kubernetes node agent kubelet ensures that containers described in pod specifications are running and healthy.

kube-proxy performs these functions:

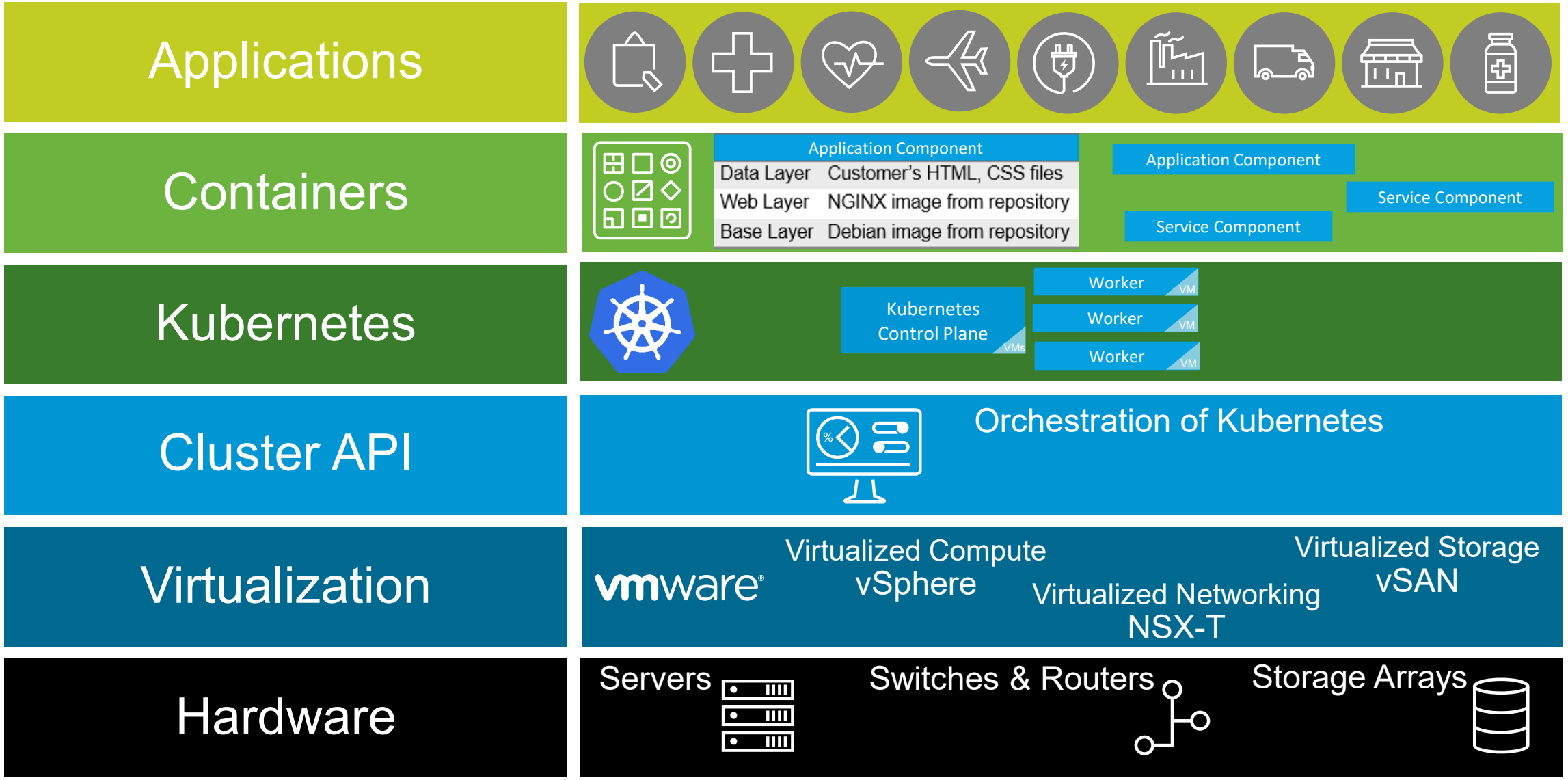
- Load balances interface for pods
- Creates a virtual IP address for external access
- Dynamically updates the node's local iptables to map references to a service port to a pod endpoint

A pod is the smallest deployable unit of computing that can be created and managed in Kubernetes.

- Reimplementation of Docker's containerd
- Container Runtime Interface (CRI) plug-in replaces `dockershim`



Application Layer of a Kubernetes Environment



Kubernetes Plug-In Architecture

During design, you can select different plug-ins:

- Container Runtime Interface (CRI): Interface between Kubernetes and the container runtime engine (docker, containerd)
- Container Networking Interface (CNI): Interface between Kubernetes and the networking implementation (NSX-T Data Center, Project Calico, Antrea)
- Container Storage Interface (CSI): Interface between Kubernetes and the storage implementation (cloud storage, vendor-specific storage, vSphere Storage Policy Based Management [SPBM]).

Kubernetes Tools

Several tools are available to build a Kubernetes cluster. Each has different goals (development or enterprise):

- kubectl
- minikube
- kind
- kubeadm
- Cluster API

About kubeadm

Bootstrap minimum viable cluster:

- Control plane creation and setup
- Joining nodes
- Certificate creation and management
- Initial cluster-admin account setup

Lesson 2: Kubernetes kubectl CLI

KOENIG
step forward

API: REST

REST JSON-based API works in the following ways:

- All internal and external components communicate through this API.
- It is explicitly versioned so that future changes can be made.
- It provides an abstraction layer on top of the storage (etcd).

```
curl http://localhost:8001/api/v1/pods


{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/pods",
    "resourceVersion": "3606680"
  },
  "items": [
```

API: kubectl

The command-line interface for the API is called kubectl:

- Calls one or more REST API calls for each command-line invocation
- Contains some business logic not in the API

```
$ kubectl get pods

NAME                    READY     STATUS    RESTARTS   AGE
blog-56d5c9dbf7-6j67j   1/1       Running   0          19s
blog-56d5c9dbf7-dh7vb   1/1       Running   0          19s
blog-56d5c9dbf7-zv7d7   1/1       Running   0          19s
```

About kubectl Command-Line Interface

You can use the command-line tool kubectl to control Kubernetes clusters:

- This tool is used to authenticate to a Kubernetes cluster.
- The kubectl client version must be compatible with the Kubernetes API service version.
- The kubectl CLI is available for Linux, macOS, and Windows operating systems.

Interacting with Kubernetes Objects Using kubectl (1)

Common `kubectl` commands include the `apply`, `get`, `describe`, and `delete` commands:

- The `kubectl apply` command applies the contents of a YAML file.

  Typically, this command is used to create a pod or deployment:

  ```
  kubectl apply -f /path/to/my.yaml
  ```

- The `kubectl get` command returns basic information about an object:

  ```
  kubectl get pod <pod_name_name>
  ```

Interacting with Kubernetes Objects Using kubectl (2)

- The `kubectl describe` command returns verbose information about an object:

```
kubectl describe pod <pod_name>
```

- The `kubectl delete` command deletes an object:

```
kubectl delete pod <pod_name>
```

Kubectl Commands (1)

```
vmware@ubuntu-01a:~/files$ kubectl --help
kubectl controls the Kubernetes cluster manager.

Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
create         Create a resource from a file or from stdin.
expose         Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
run            Run a particular image on the cluster
set            Set specific features on objects

Basic Commands (Intermediate):
explain        Documentation of resources
get            Display one or many resources
edit           Edit a resource on the server
delete         Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:
rollout        Manage the rollout of a resource
scale          Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
autoscale      Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:
certificate    Modify certificate resources.
cluster-info   Display cluster info
top            Display Resource (CPU/Memory/Storage) usage.
cordon         Mark node as unschedulable
uncordon       Mark node as schedulable
drain          Drain node in preparation for maintenance
taint          Update the taints on one or more nodes
```

Kubectl Commands (2)

```
Troubleshooting and Debugging Commands:
describe        Show details of a specific resource or group of resources
logs            Print the logs for a container in a pod
attach          Attach to a running container
exec            Execute a command in a container
port-forward    Forward one or more local ports to a pod
proxy           Run a proxy to the Kubernetes API server
cp              Copy files and directories to and from containers.
auth            Inspect authorization

Advanced Commands:
diff            Diff live version against would-be applied version
apply           Apply a configuration to a resource by filename or stdin
patch           Update field(s) of a resource using strategic merge patch
replace         Replace a resource by filename or stdin
wait            Experimental: Wait for a specific condition on one or many resources.
convert         Convert config files between different API versions
kustomize       Build a kustomization target from a directory or a remote url.

Settings Commands:
label           Update the labels on a resource
annotate        Update the annotations on a resource
completion      Output shell completion code for the specified shell (bash or zsh)

Other Commands:
api-resources   Print the supported API resources on the server
api-versions    Print the supported API versions on the server, in the form of "group/version"
config          Modify kubeconfig files
plugin          Provides utilities for interacting with plugins.
version         Print the client and server version information
```

Commands: kubectl get nodes

To list all the nodes in the selected Kubernetes cluster, you run the following command:

```
kubectl get nodes
```

```
vmware@ubuntu-01a:~$ kubectl get nodes -o wide
NAME                                   STATUS   ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
f43363ff-03d6-43bf-8066-2bf49a8dce73   Ready    <none>   8d    v1.14.5   10.20.0.3     10.20.0.3     Ubuntu 16.04.6 LTS   4.15.0-55-generic   docker://18.9.8
```

Commands: kubectl get pods

To list all pods in the selected namespace, you run the following command:

```
kubectl get pods
```

```
vmware@ubuntu-01a:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                    READY   STATUS    RESTARTS   AGE
kube-system   coredns-95489c5c9-v2sbh                 1/1     Running   0          8d
kube-system   coredns-95489c5c9-vbs8n                 1/1     Running   0          8d
kube-system   coredns-95489c5c9-zghhf                 1/1     Running   0          8d
kube-system   kubernetes-dashboard-558689fc66-plk2d   1/1     Running   0          8d
kube-system   metrics-server-867b8fdb7d-64hnn          1/1     Running   0          8d
pks-system    event-controller-646d78b9b8-nvvmv       2/2     Running   0          8d
pks-system    fluent-bit-szttw                        2/2     Running   0          8d
pks-system    metric-controller-c998cb5bf-ndpcd       1/1     Running   0          8d
pks-system    observability-manager-64f749cd4c-bjbqs  1/1     Running   0          8d
pks-system    sink-controller-6774fd95f7-f4k45        1/1     Running   0          8d
pks-system    telegraf-gcg52                          1/1     Running   0          8d
pks-system    telemetry-agent-858446f4ff-n77mv        2/2     Running   0          8d
pks-system    validator-6b677f49d4-khps2              1/1     Running   0          8d
pks-system    vrops-cadvisor-6tdth                    1/1     Running   0          8d
```

Navigating Namespaces Using kubectl

A user can have permissions on multiple namespaces. The `kubectl` commands are typically actioned against the current active namespace.

View the list of available namespaces:

```
kubectl config get-contexts
```

Change the current active namespace:

```
kubectl config use-context <namespace>
```

Kubernetes Troubleshooting: Event Log

The event log records events on the cluster:

- Contains helpful troubleshooting information
- By default, stores an hour of history

```
$ kubectl get events

LAST SEEN    TYPE      REASON                    KIND    MESSAGE
2m14s        Normal    NodeHasSufficientMemory   Node    Node master  status is now: NodeHasSufficientMemory
2m14s        Normal    NodeHasNoDiskPressure     Node    Node master  status is now: NodeHasNoDiskPressure
2m14s        Normal    NodeHasSufficientPID      Node    Node master  status is now: NodeHasSufficientPID
2m14s        Normal    NodeAllocatableEnforced   Node    Updated Node Allocatable limit across pods
51s          Normal    RegisteredNode            Node    Node master event: Registered Node master in Controller
38s          Normal    Starting                  Node    Starting kube-proxy.
```

Getting a List of Objects

You run `kubectl get` to retrieve a list of objects.

You add `-o wide` for more details about each object.

```
$ kubectl get pods -o wide


NAME                              READY    STATUS     RESTARTS    AGE      IP              NODE
gowebapp-54d74586cc-jqlmx         1/1      Running    0           3m5s     192.168.0.7     master
gowebapp-54d74586cc-sqpfk         1/1      Running    0           3m5s     192.168.0.6     master
gowebapp-mysql-9746475d-gdhtx     1/1      Running    0           3m35s    192.168.0.5     master


# When using -o wide with `kubectl get pods` the output will include the Pod's IP address, and which  #
node it's running on.
```

Getting More Information About an Object

You run `kubectl describe` to view details about an object.

The output includes event log entries that are generated by that object.

```
$ kubectl describe pod gowebapp-54d74586cc-jqlmx


Name:               gowebapp-54d74586cc-jqlmx
Namespace:          default
Node:               master/172.31.30.8
Start Time:         Wed, 08 May 2021 15:17:46 +0000
Labels:             app=gowebapp
                    tier=frontend
Status:             Running
IP:                 192.168.0.7
Controlled By:      ReplicaSet/gowebapp-54d74586cc
```

Example: kubectl describe

```
Containers:
  gowebapp:
    Container ID:    docker://ae0321def949ed866fb0c303207eaa0d3188dd9830f648d52196418637bda13d
    Image:           localhost:5000/gowebapp:v1
    Image ID:        docker-pullable://localhost:5000/gowebapp@sha256:8227…
Events:
  Type    Reason                 Age   From                Message
  ----    ------                 ----  ----                -------
  Normal  Scheduled              11m   default-scheduler   Successfully assigned default/gowebapp-54d7458… to master
  Normal  Pulled                 11m   kubelet, master     Container image "localhost:5000/gowebapp:v1" already present…
  Normal  Created                11m   kubelet, master     Created container
  Normal  Started                11m   kubelet, master     Started container
```

Creating Resources

Declarative approach (preferred):

Creates new or updates existing resources from files

```
kubectl apply -f [ <file> | <directory> | <url> ]
```

Imperative approach:

`kubectl create`: Creates new resource from a file

`kubectl replace`: Updates an existing resource from a file

`kubectl edit`: Updates existing resource using your default editor

`kubectl patch`: Updates existing resource by merging a code snippet

Getting Pod Logs

You run `kubectl logs` to retrieve log output from a container.

You can use `-c <container_name>` if the pod contains more than one container.

```
$ kubectl logs gowebapp-54d74586cc-jqlmx


2021-05-08  03:17:48  PM  Running HTTP :80
2021-05-08  03:41:09  PM  172.31.30.8:56701  GET /
2021-05-08  03:41:09  PM  172.31.30.8:56701  GET /static/css/bootstrap.min.css?1544384023
2021-05-08  03:41:09  PM  172.31.30.8:56703  GET /static/css/global.css?1544384023
2021-05-08  03:41:09  PM  172.31.30.8:56705  GET /static/js/underscore-min.js?1544384023
2021-05-08  03:41:09  PM  172.31.30.8:56707  GET /static/js/global.js?1544384023
2021-05-08  03:41:09  PM  172.31.30.8:56701  GET /static/favicons/favicon-196x196.png
2021-05-08  03:41:11  PM  172.31.30.8:56701  GET /login
2021-05-08  03:41:17  PM  172.31.30.8:56701  GET /register
```

Kubernetes Pod Phases

Pending:

- The pod was accepted by the system, but one or more of the container images are not created. Includes the time before being scheduled and the time spent downloading images over the network.

Running:

- The pod is bound to a node, and all containers are created.
- At least one container is still running or is in the process of starting or restarting.

Succeeded:

- All containers in the pod have terminated in success and are not restarted.

Failed:

- All containers in the pod have terminated. At least one container has terminated in failure (exited with nonzero exit status or was terminated by the system).

Unknown:

- The state of the pod cannot be obtained, typically because of an error in communicating with the host of the pod.

Kubernetes – Beyond Basics

KOENIG
step forward

Kubernetes Objects

KOENIG
step forward

Basic Kubernetes Objects

You use the following objects in Kubernetes:

- Pods
- Replicasets
- Deployments
- Services
- Namespaces
- Labels

About Manifests

In Kubernetes, manifest files declare the desired state of objects.

Manifests have the following properties:

- YAML format
- Declarative configuration
- Desired API primitives

Kubernetes manages the creation of the requested primitives.

About Pods

A pod is a set of one or more tightly coupled containers.

It is the smallest unit of work in Kubernetes.

Containers in a pod live and die together.

Create a pod example:

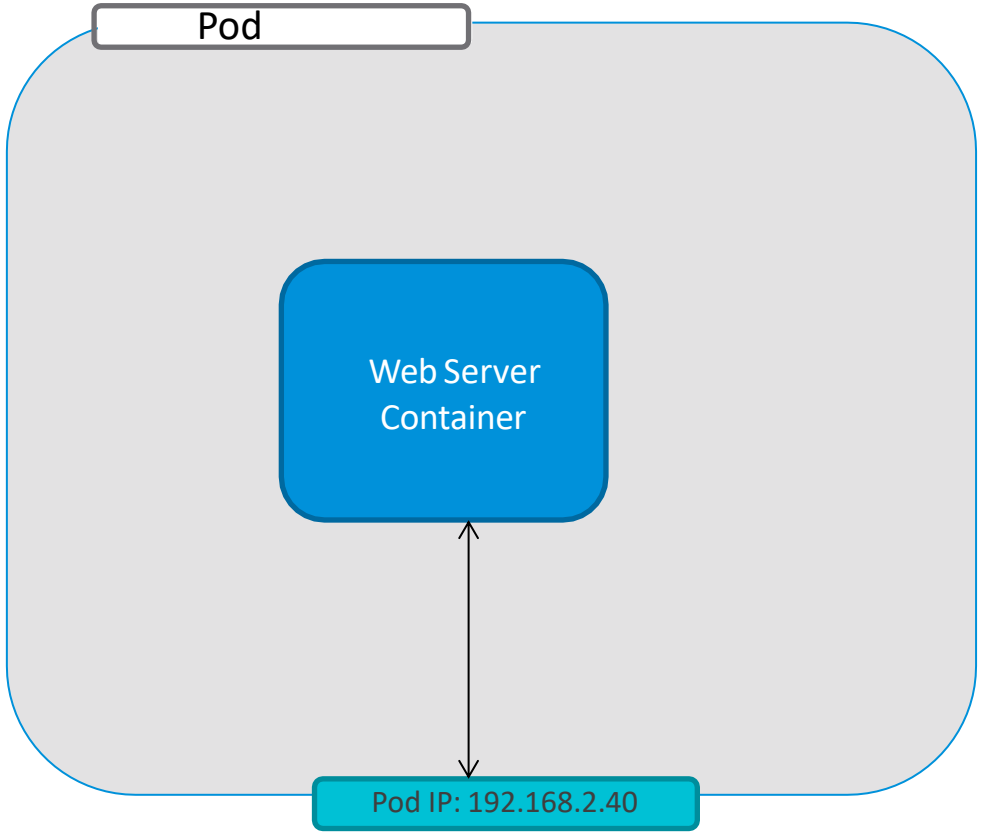
- The pod has a label app with the value nginx.
- The pod runs the image nginx with version 1.7.9.
- The pod listens on port 80.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-6dd86-nlrhx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
      - containerPort: 80
```

Pod-to-Container Relationship

The Kubernetes pod is created to run the container.

The container is the reason that a pod exists.



Multiple Containers in a Pod

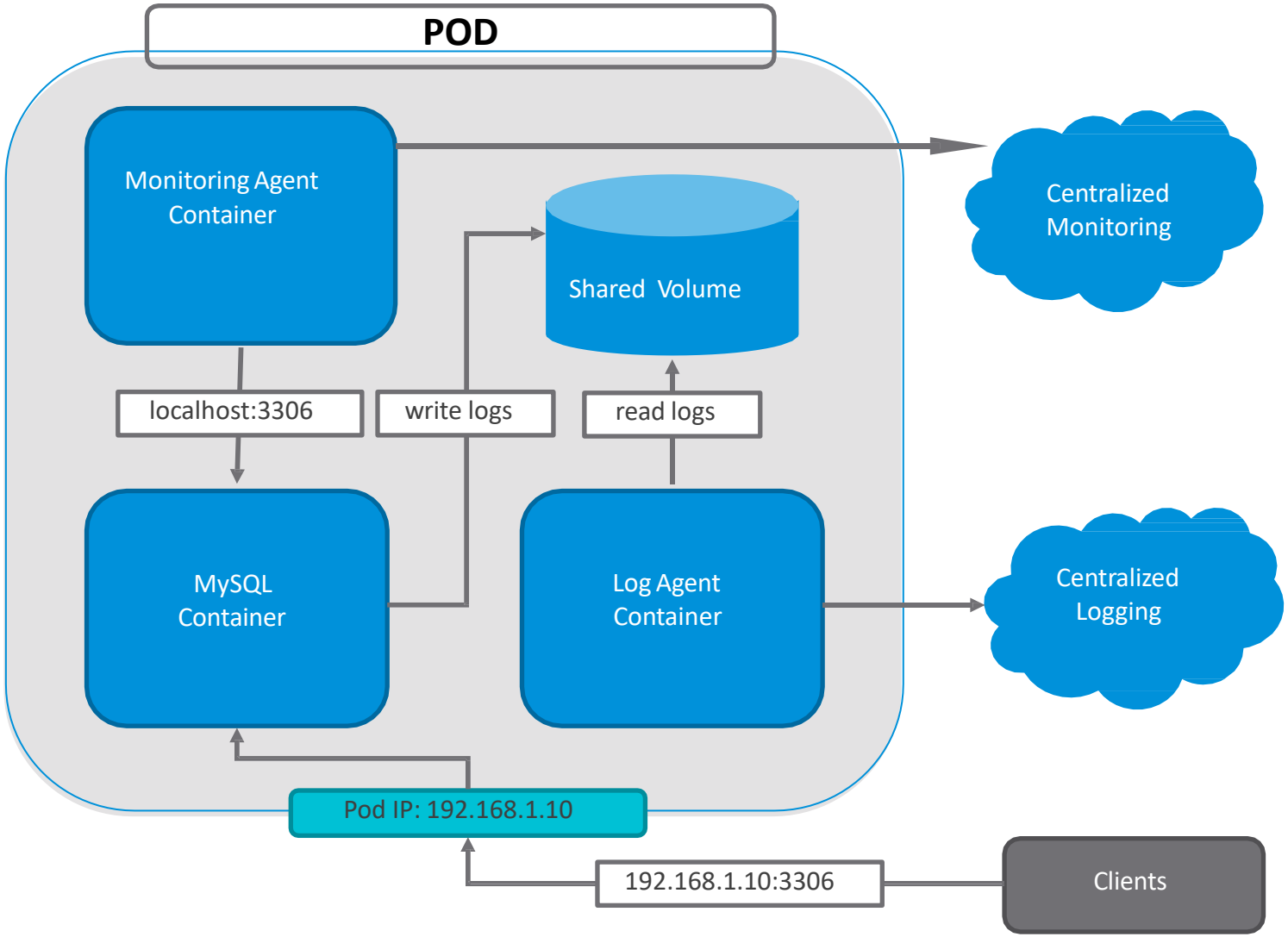
A pod typically exists for one container (1:1 relationship)

One or more containers can be included in a pod.

All containers in a pod run on the same node.

Containers in a pod can talk to each other over the localhost.

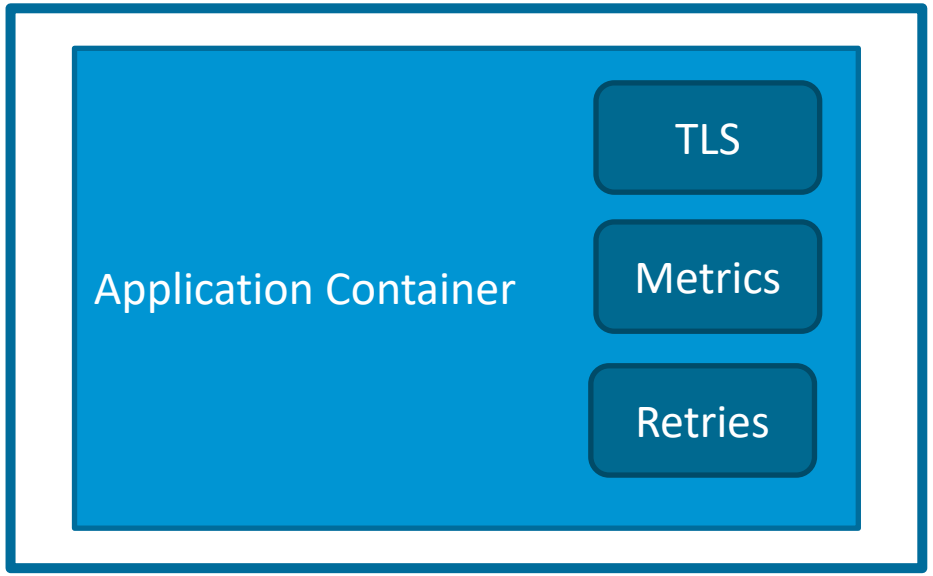
Containers can share volume resources.



Sidecar Containers

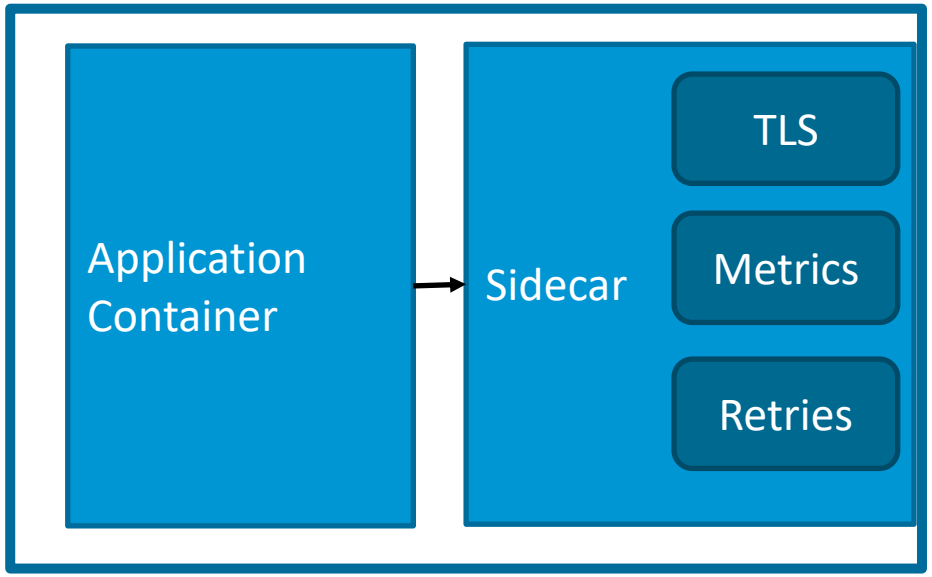
The term sidecar is commonly used to refer to logging or metric pods that are used to forward information into another system such as logging or monitoring.

Before



Kubernetes Pod

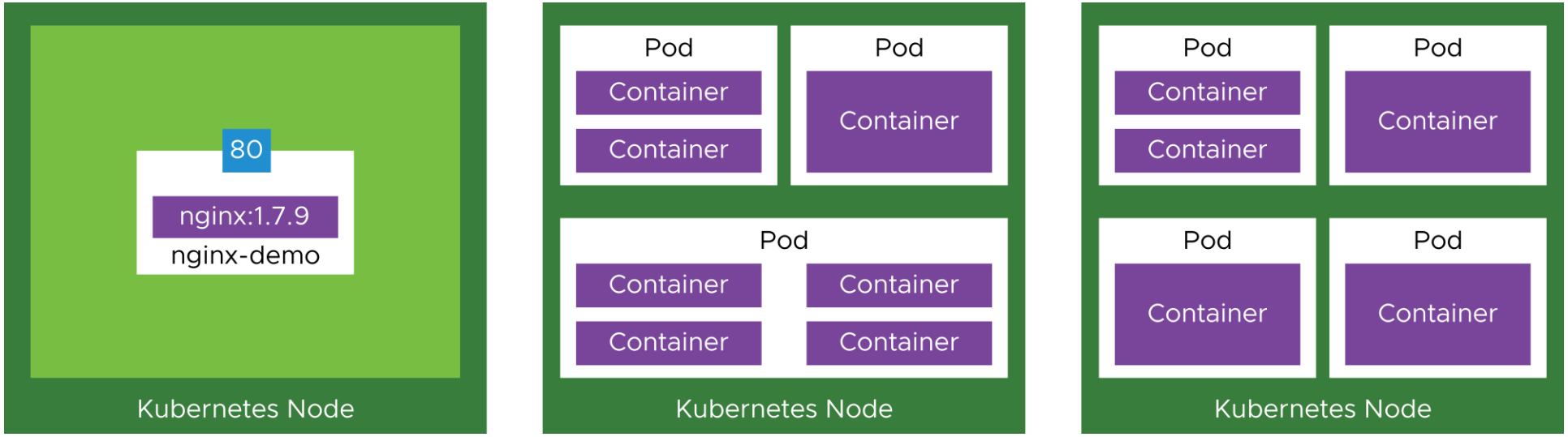
After



Kubernetes Pod

Pod to Worker Node Relationship

Pods run inside Kubernetes worker nodes and can run one or more container processes.



About ReplicaSets (1)

A ReplicaSet declares how the functionality of a pod is made scalable and resilient through redundancy.

With the ReplicaSet, a specified number of pods is kept running.

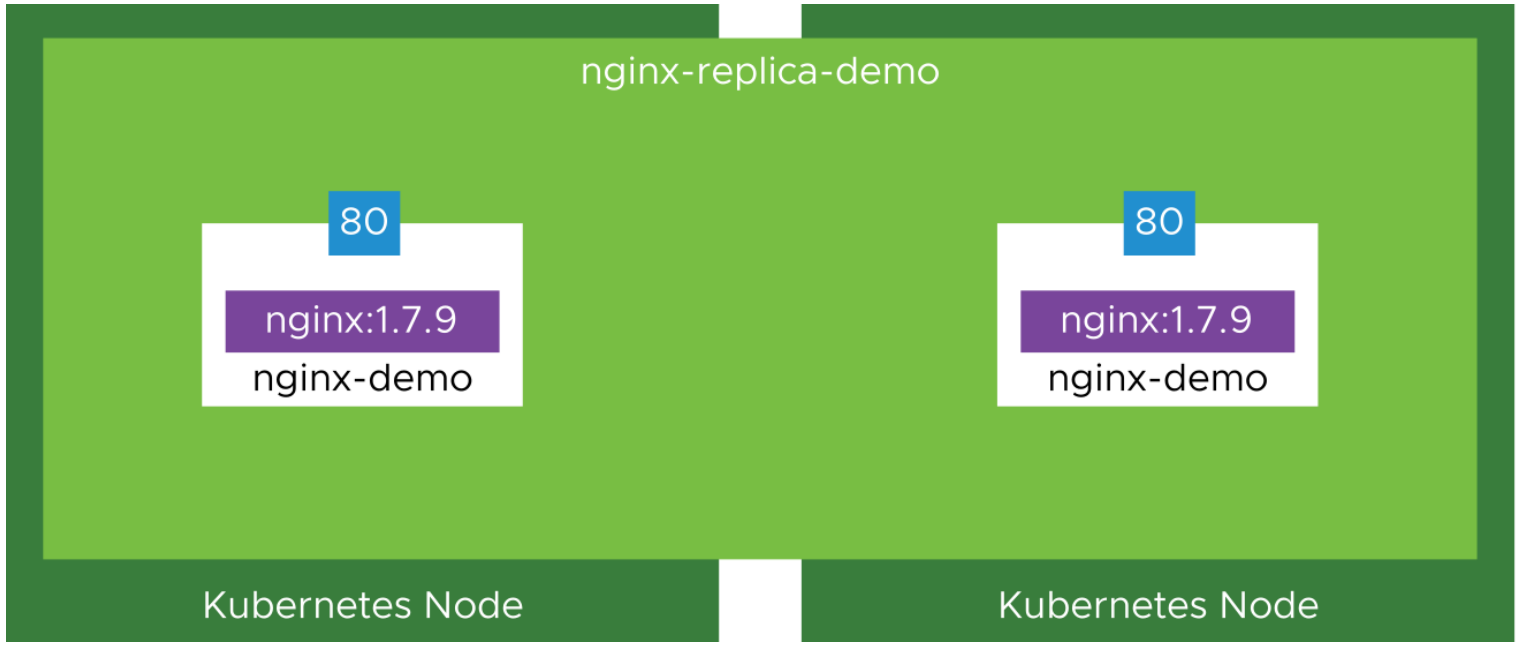
Create a ReplicaSet example:

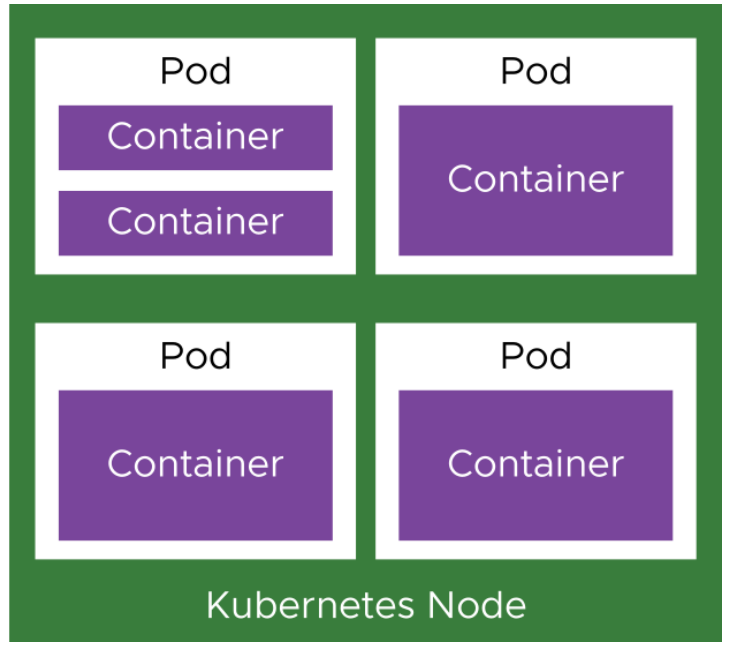
- The ReplicaSet creates two pods running nginx.
- The pods have an app label with the value nginx.
- The ReplicaSet ensures that the two pods are always running.

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: my-nginx-6dd86
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

About ReplicaSets (2)

ReplicaSets create and maintain copies of a pod. Replica pods can run on the same Kubernetes node or across nodes.





Activity: Pod Creation and Deployments

Situation: You have a website that needs 20 NGINX servers.

Challenge: How many pods do you deploy?

Solution: Pod Creation and Deployments

For a website that requires 20 NGINX servers, how many pods do you deploy?

Solution: Twenty pod objects are sent to Kubernetes apiserver.

Twenty pod manifest files are required.

A ReplicaSet can be used.

But what if updates are required?

A higher-level construct with automation of updates is required.

About Deployments (1)

A deployment is the most commonly used object:

- Provides rolling updates by creating ReplicaSets and destroying old ReplicaSets automatically
- Enables a new version of an image to be deployed without downtime

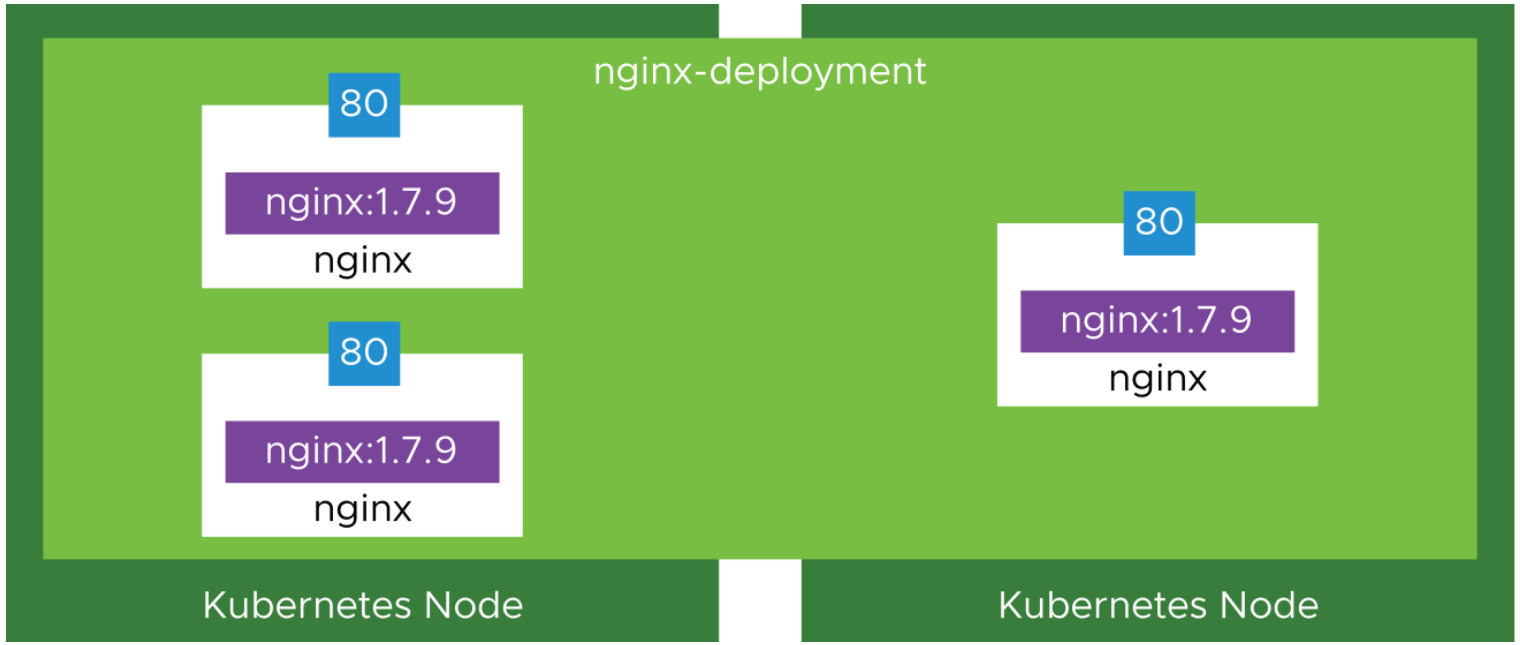
Create a Deployment example:

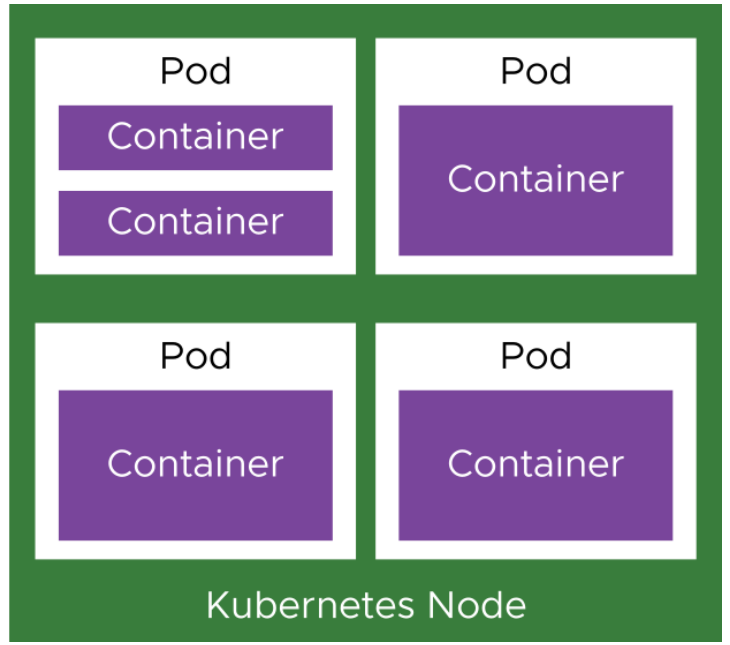
- The deployment creates a ReplicaSet with two replicas.
- The ReplicaSet creates two pods running the nginx image.
- When the spec is updated with a new image version, a rolling update is performed all by a single object of the kind Deployment.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: my-nginx
spec:
  strategy:
    type: RollingUpdate
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

About Deployments (2)

Deployments are commonly used to combine pod and ReplicaSet declarations in a single manifest.





About Services (1)

A service describes how pods discover and communicate with each other and external networks.

A service exposes a deployment as a single IP address.

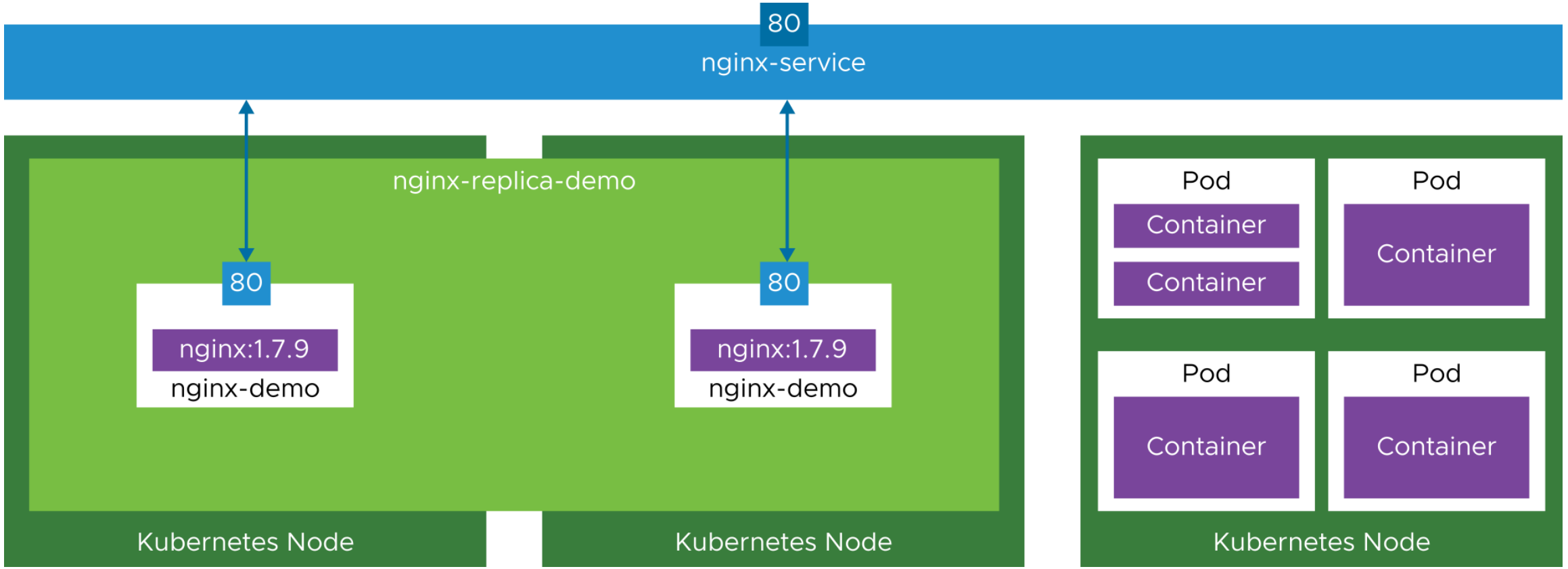
Create a service example:

- The service type is LoadBalancer.
- The load balancer listens on port 80.
- Traffic is balanced across pods with the app label nginx.
- Traffic is sent to pods on target port 80.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-service
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

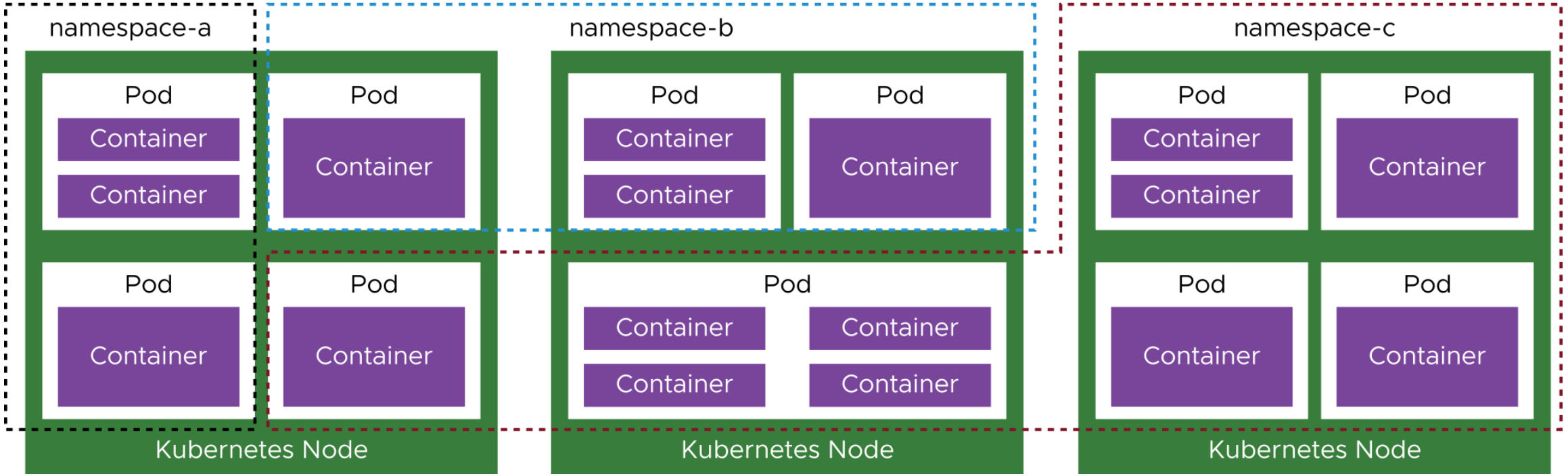
About Services (2)

Services, such as load-balancer services, can expose pods with a static external IP address.



About Namespaces (1)

Namespaces provide a resource and authorization boundary for Kubernetes objects. Namespaces can span across Kubernetes nodes.



About Namespaces (2)

A namespace is a Kubernetes construct that is used to divide cluster resources:

- Namespaces support multiuser access.
- Resources are controlled using resource quotas.

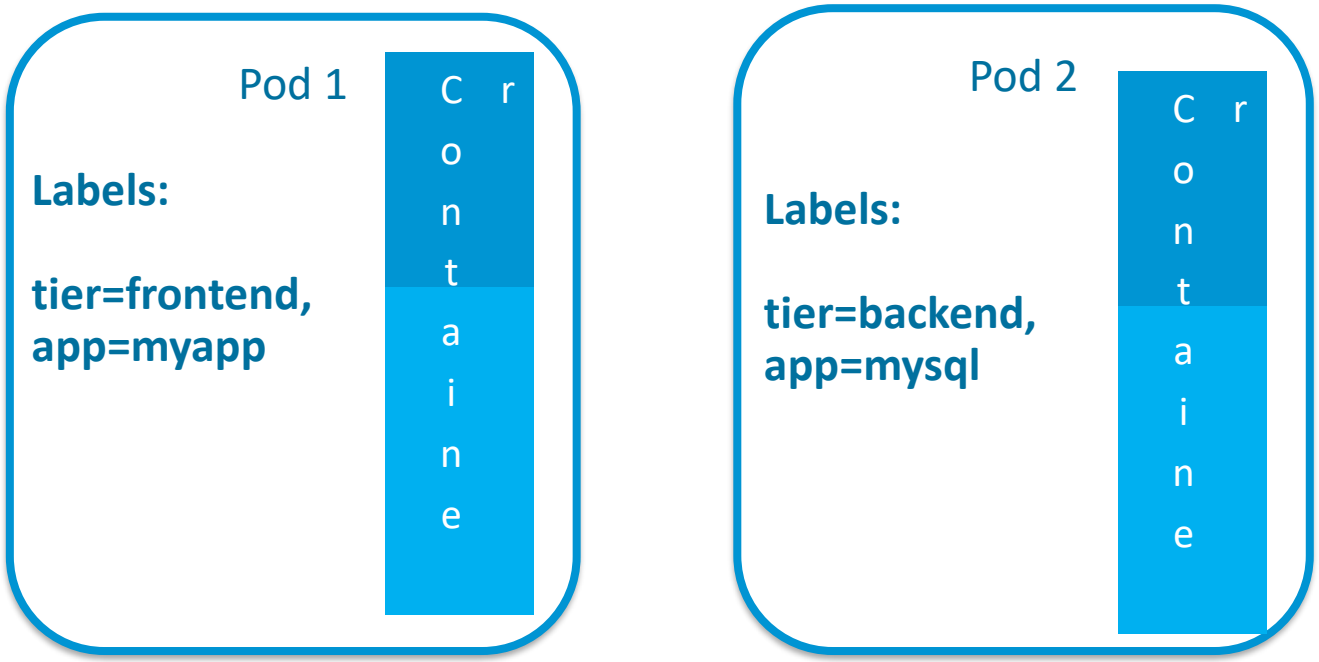
About Labels

A label is a key-value pair attached to pods that conveys user-defined attributes. You aim to standardize key-value pairs across a cluster.

You can use label selectors to select pods with specific labels and apply services or replication controllers to them (they can be functional or organizational).

Labels can be attached to objects during creation and subsequently added and modified at any time.

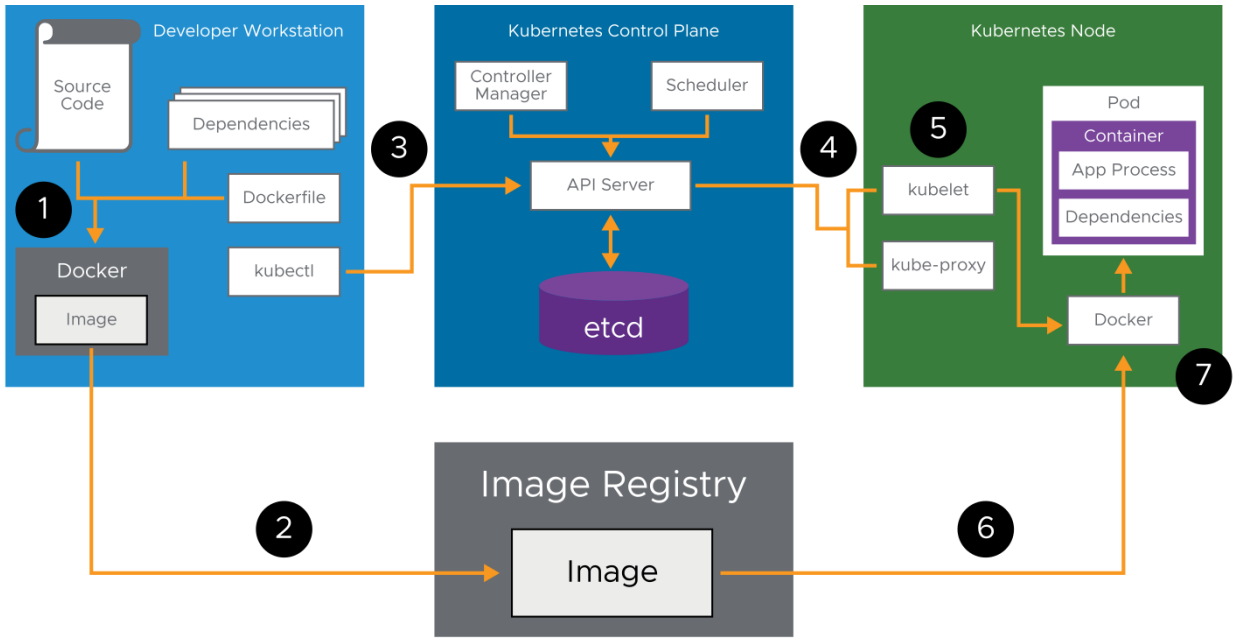
Labels are indexed, searchable, and best used consistently as a key-value pair across the cluster.



Typical Kubernetes Workflow Example

Example workflow steps:

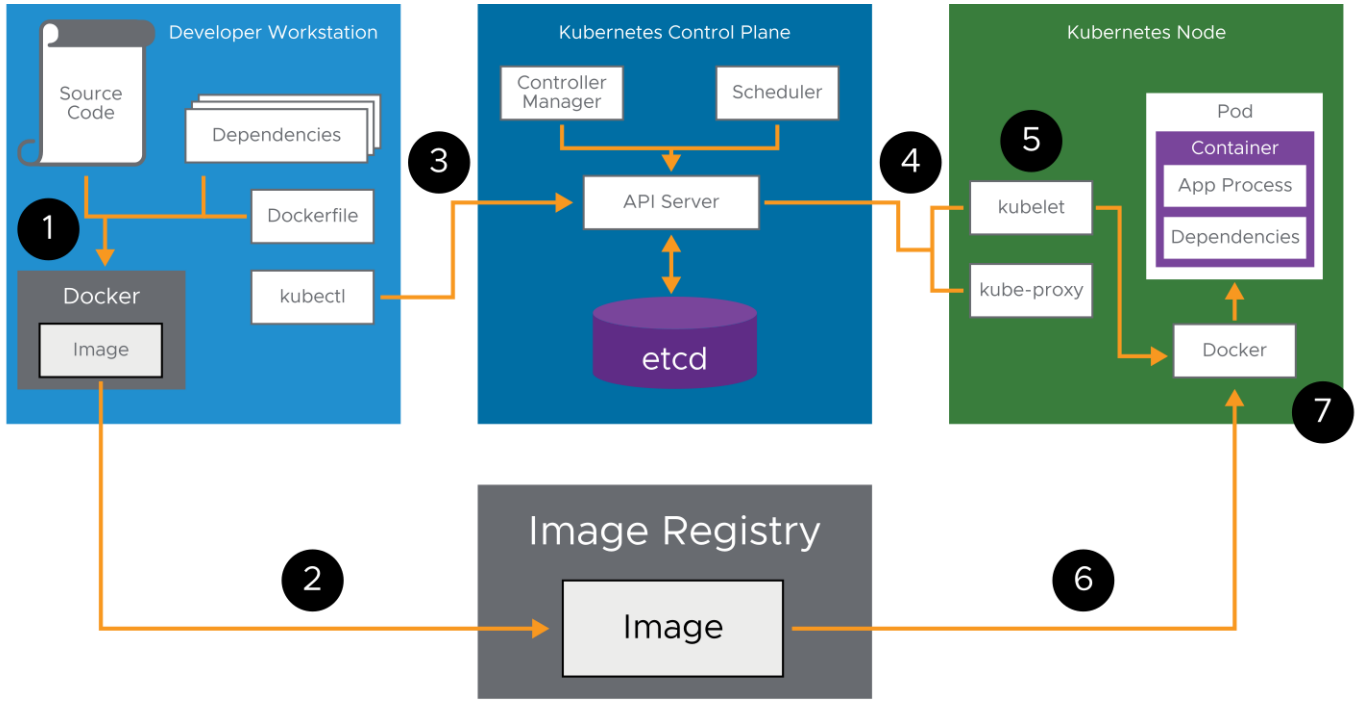
1. Build an image from source code and dependencies.
2. Send the image to the image registry.
3. Tell Kubernetes to use the image to run a pod.
4. Scheduler assigns the pod to a node.
5. Kubelet accepts the pod.
6. Docker takes the image from the image registry.
7. Docker starts the container process inside a pod.



Typical Kubernetes Workflow: Example Commands

Example commands corresponding to workflow steps:

1. `docker build`
2. `docker tag` and `docker push`
3. `kubectl apply -f deployment.yaml`
4. Remaining steps happen as part of Kubernetes orchestration.



Deployment Management

KOENIG
step forward

Deployments: ReplicaSets

Deployments handle the creation and modification of ReplicaSets.

Pods for a deployment span all valid ReplicaSets.

You do not manually manage ReplicaSets that are owned by a deployment.

```
NAME     DESIRED CURRENT     UP-TO-DATE     AVAILABLE     AGE    CONTAINER(S)      IMAGE(S)
nginx    3         3         3              3             23m    nginx             nginx:1.13.1
```

Built-In Deployment Strategies

RollingUpdate deployment:

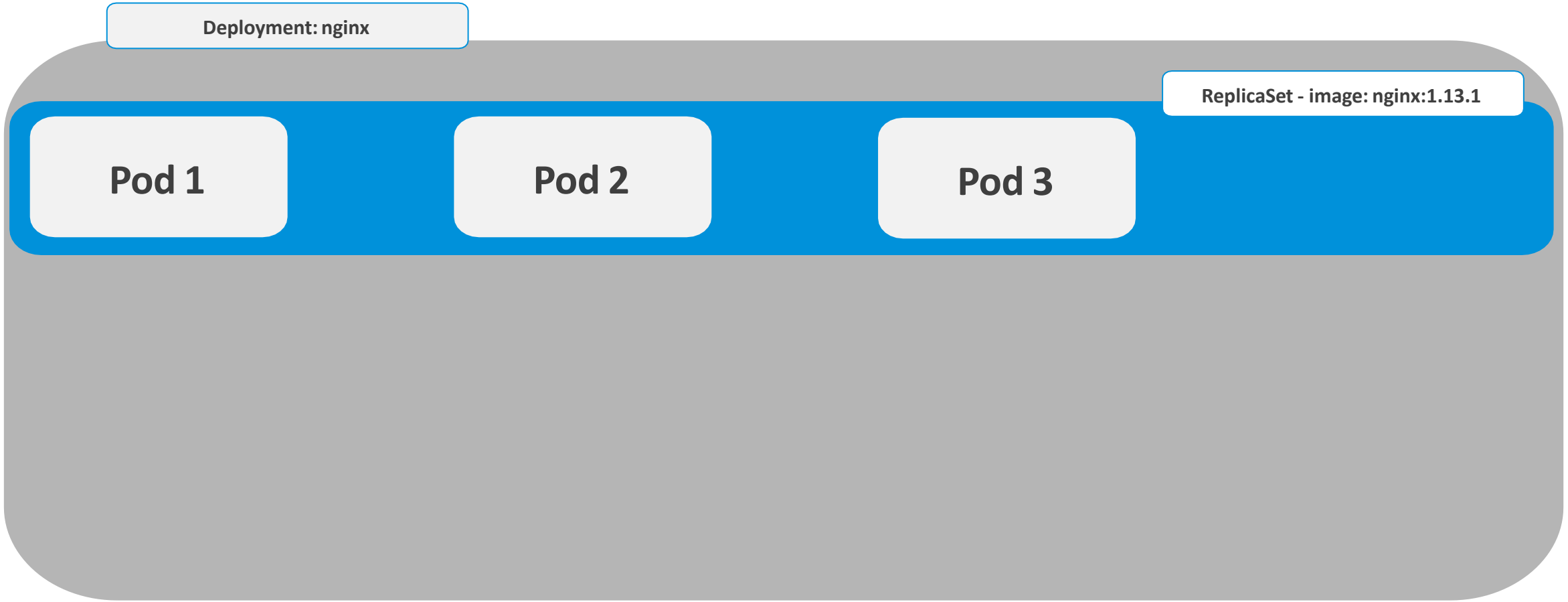
- This type is the default if a deployment is not defined.
- A new ReplicaSet is created and then scaled up as the old ReplicaSet is scaled down.

Recreate deployment:

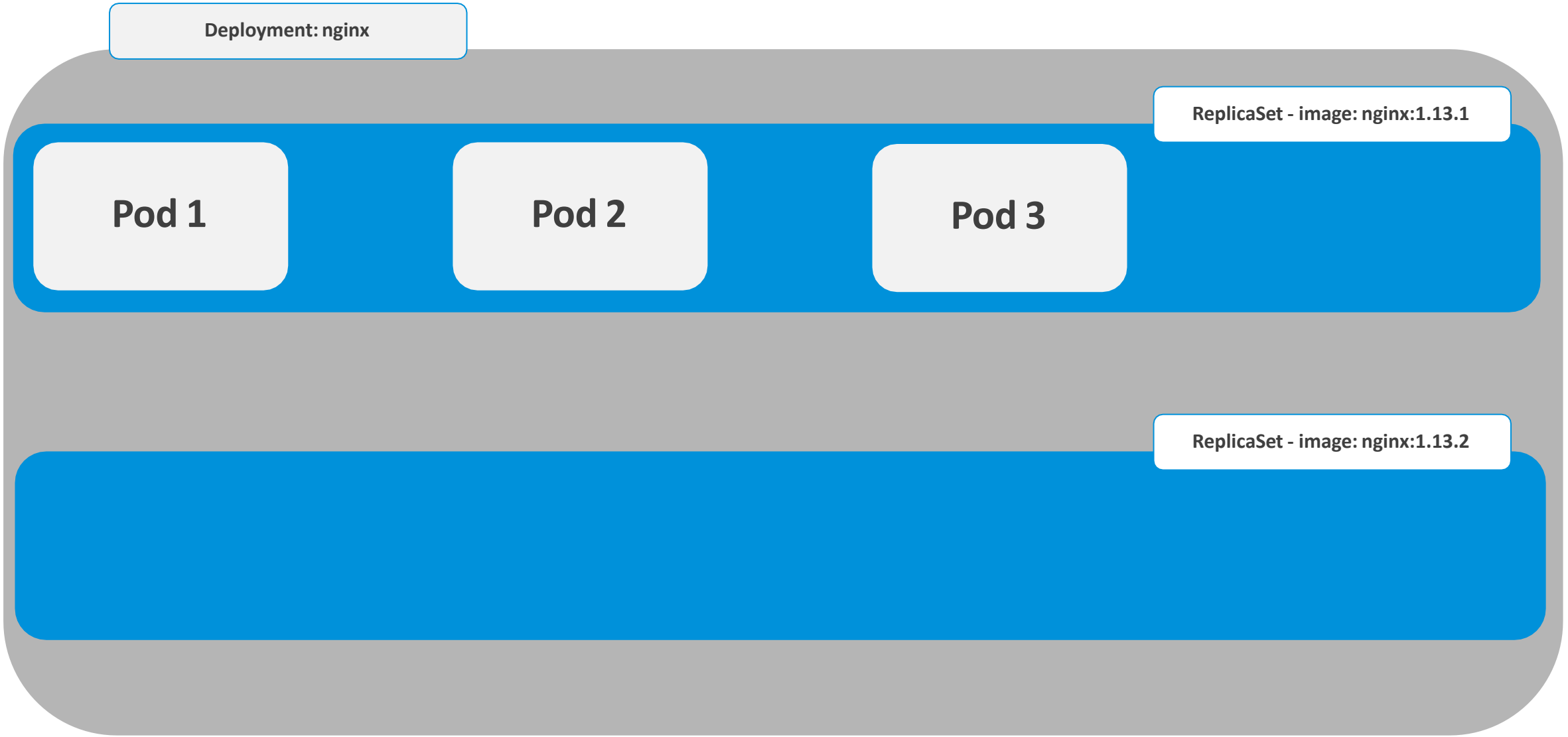
- Removes all existing pods in the existing ReplicaSet first
- Creates new pods in the new ReplicaSet

Deployment Strategies: RollingUpdate (1)

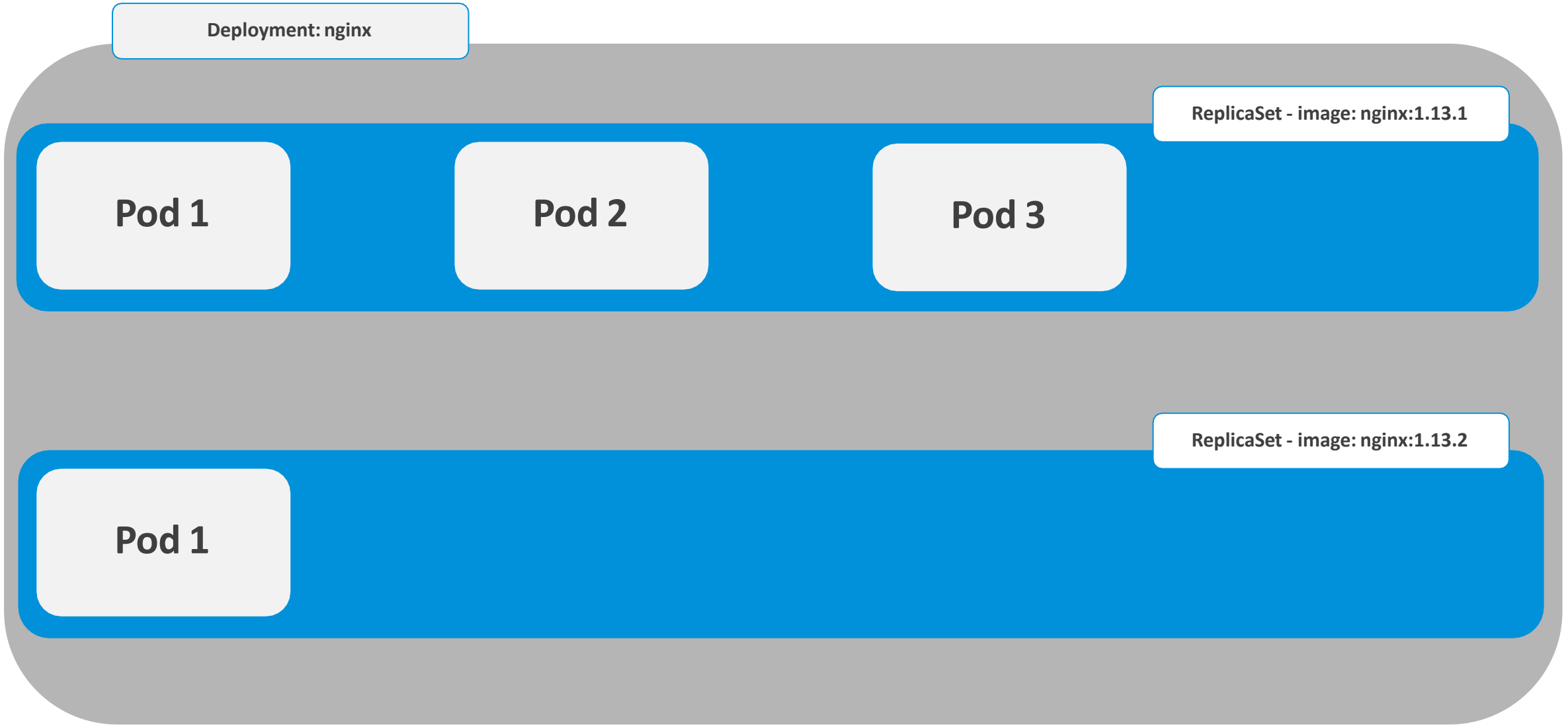
Applications are constantly being improved and updated. An easy and efficient way to deploy a new version is available through the Deployment object.



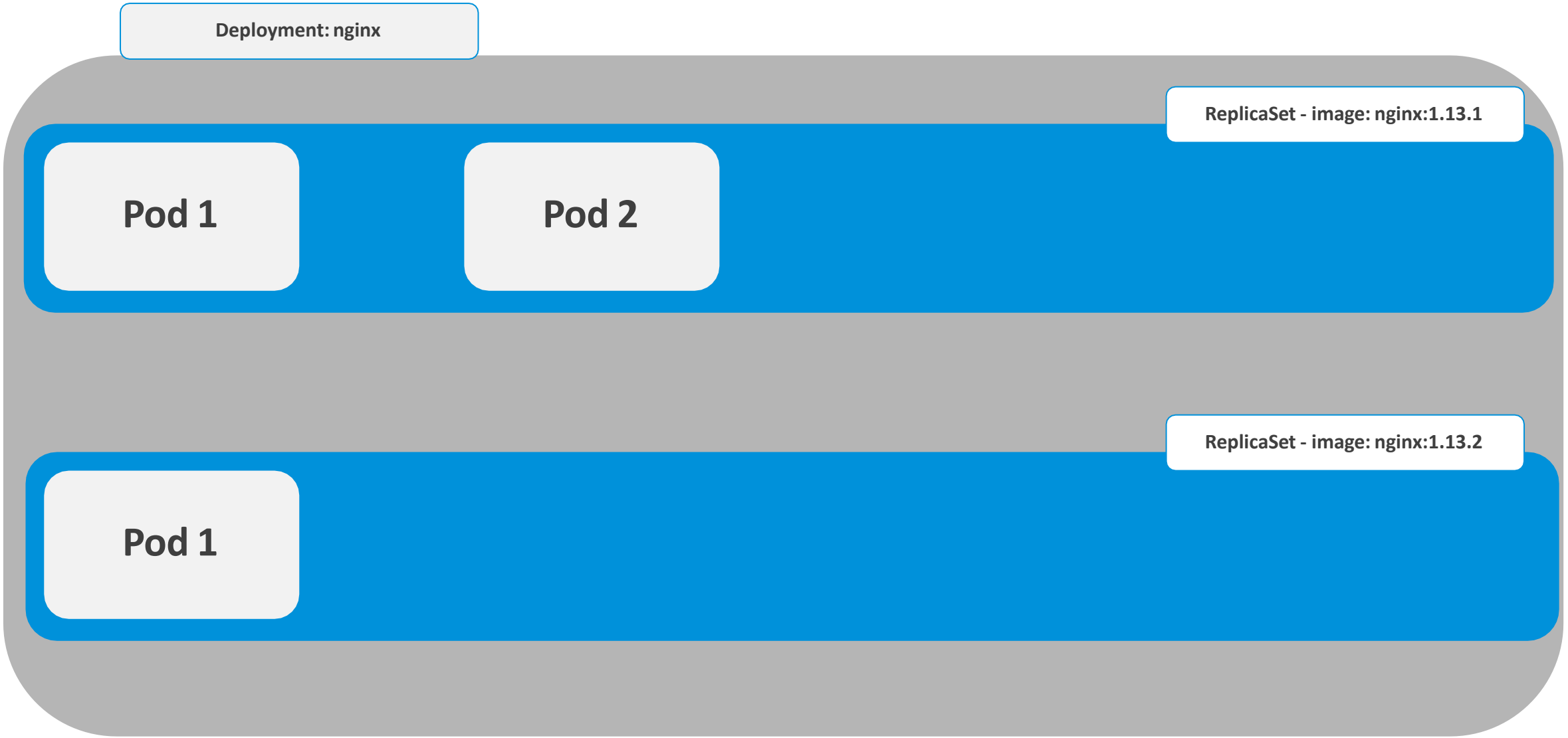
Deployment Strategies: RollingUpdate (2)



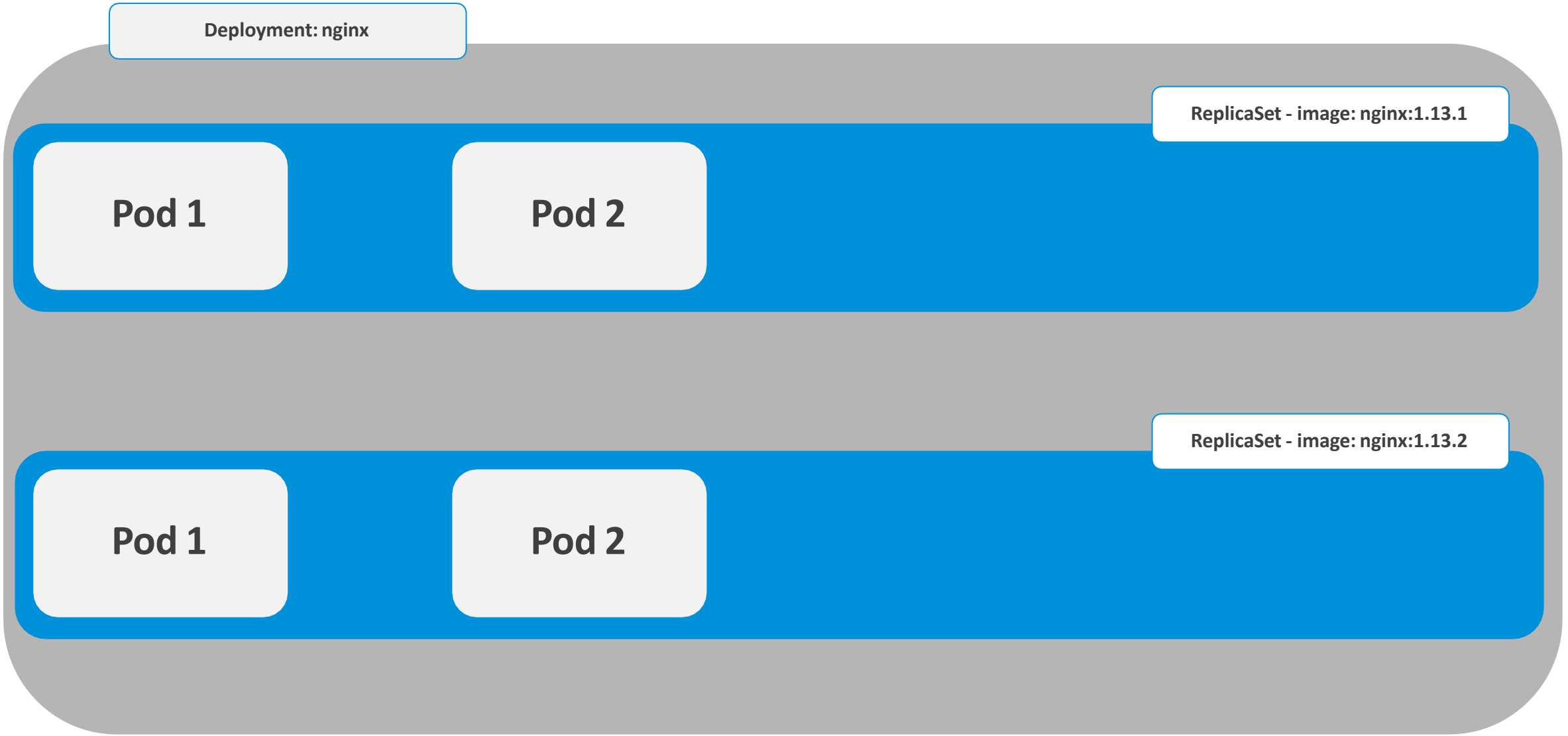
Deployment Strategies: RollingUpdate (3)



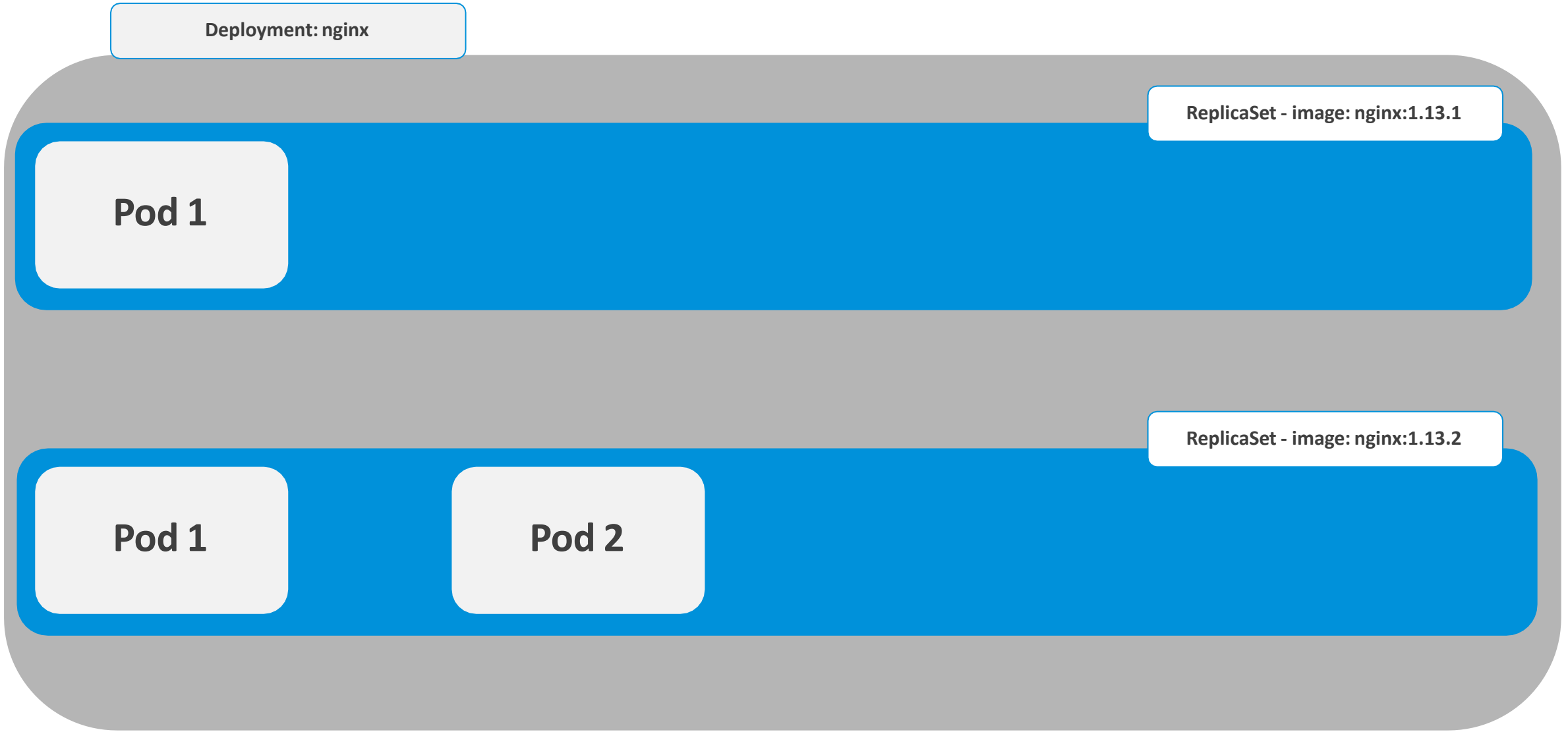
Deployment Strategies: RollingUpdate (4)



Deployment Strategies: RollingUpdate (5)



Deployment Strategies: RollingUpdate (6)

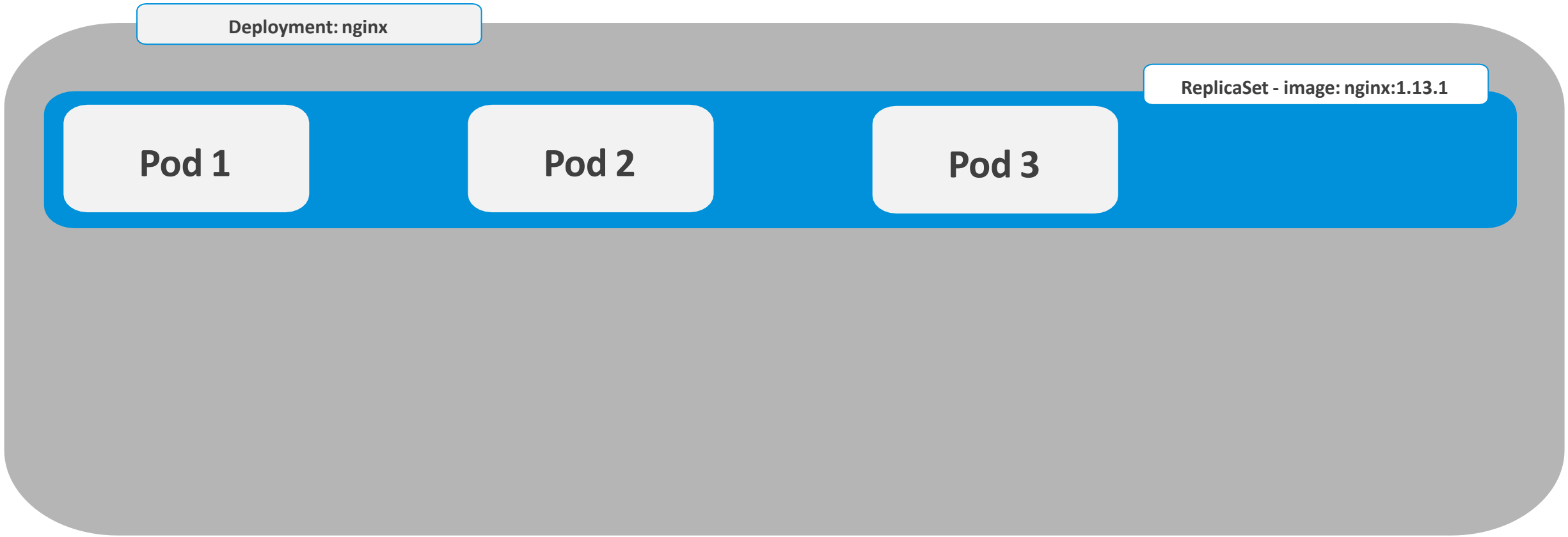


Deployment Strategies: RollingUpdate (7)

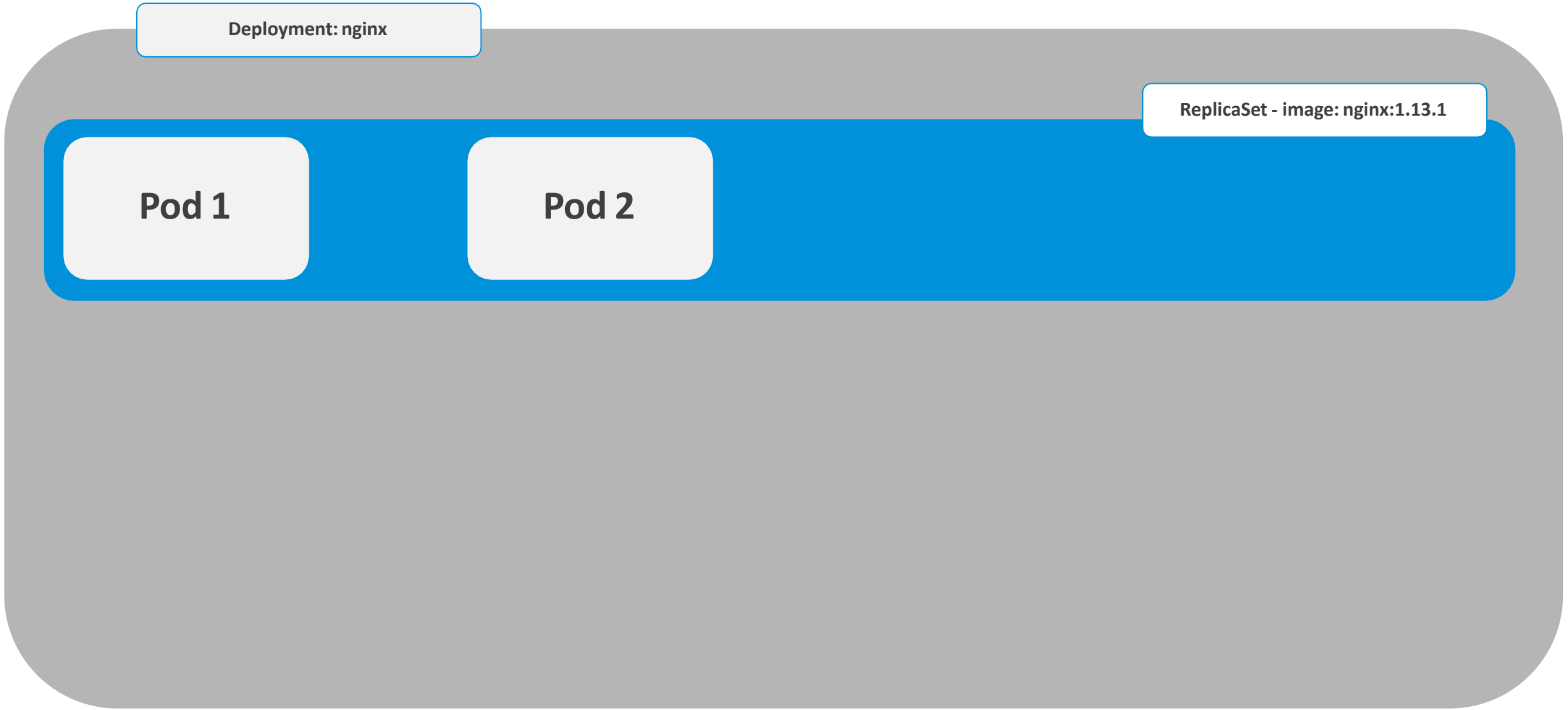
Deployment Strategies: RollingUpdate (8)

Deployment Strategies: Recreate (1)

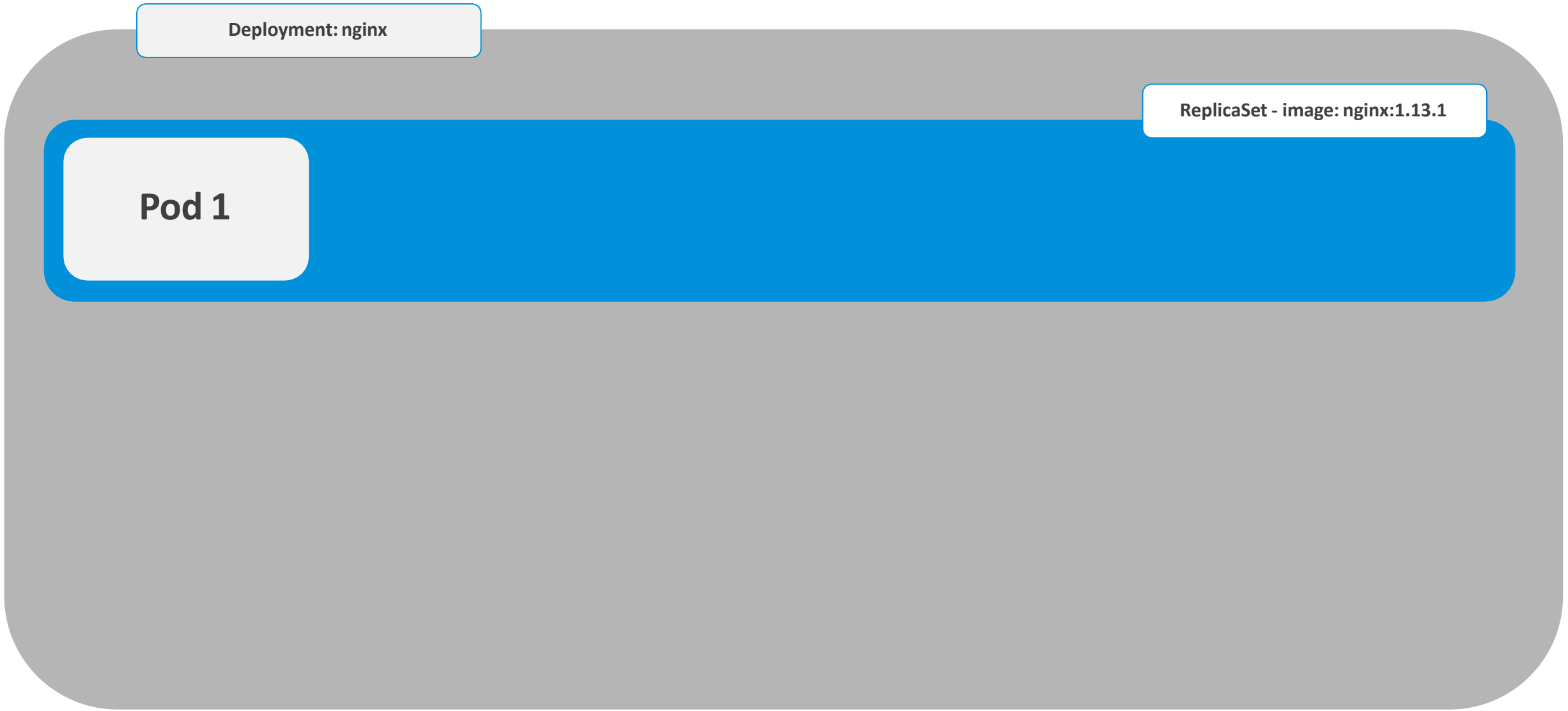
Using the recreate update causes the container to become unavailable for a short duration.



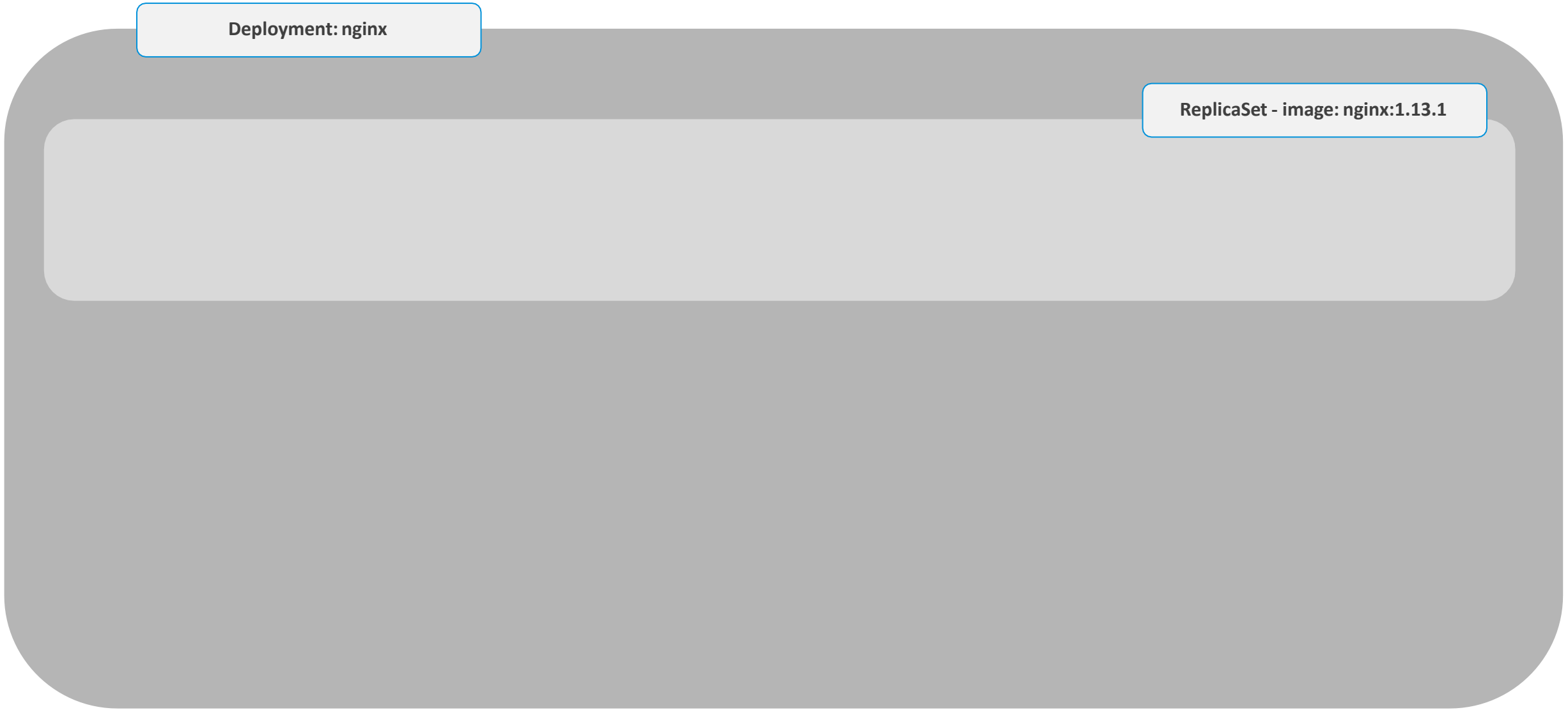
Deployment Strategies: Recreate (2)



Deployment Strategies: Recreate (3)

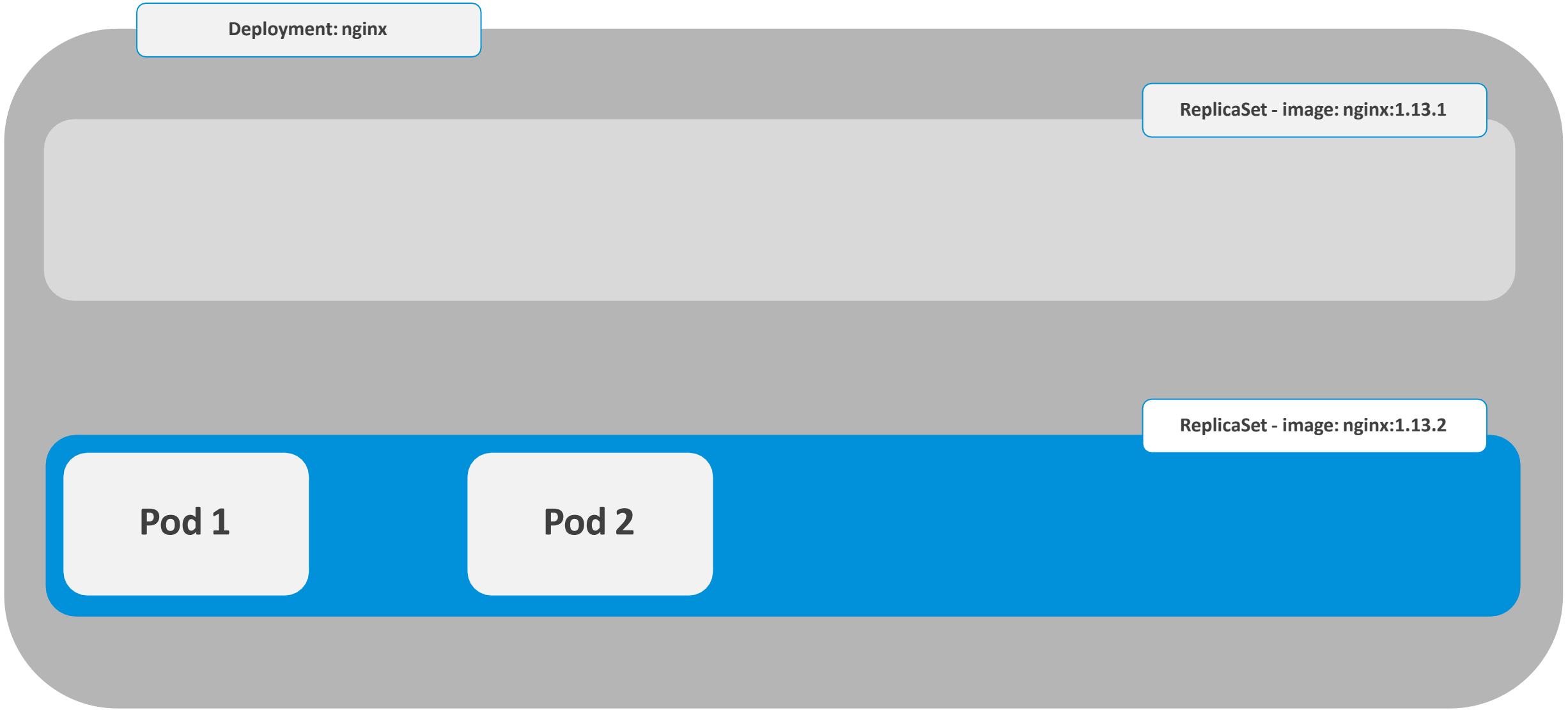


Deployment Strategies: Recreate (4)



Deployment Strategies: Recreate (5)

Deployment Strategies: Recreate (6)



Deployment Strategies: Recreate (7)

Deployment Strategies: Canary and Blue/Green

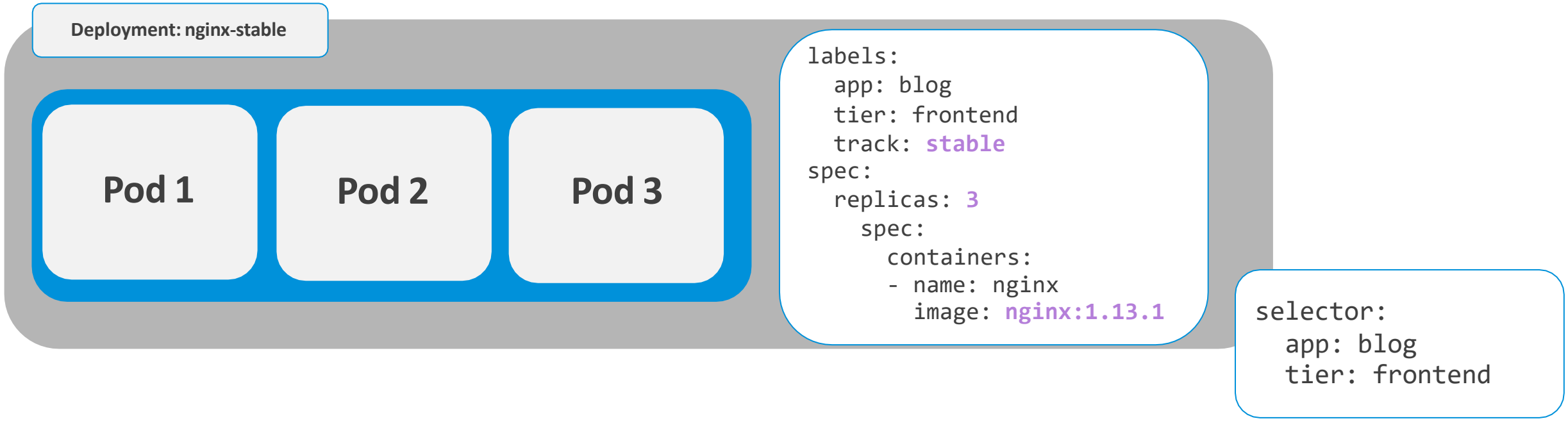
Canary deployment:

- Deploy new container to a subset of traffic
- Involves two deployments and label management between them

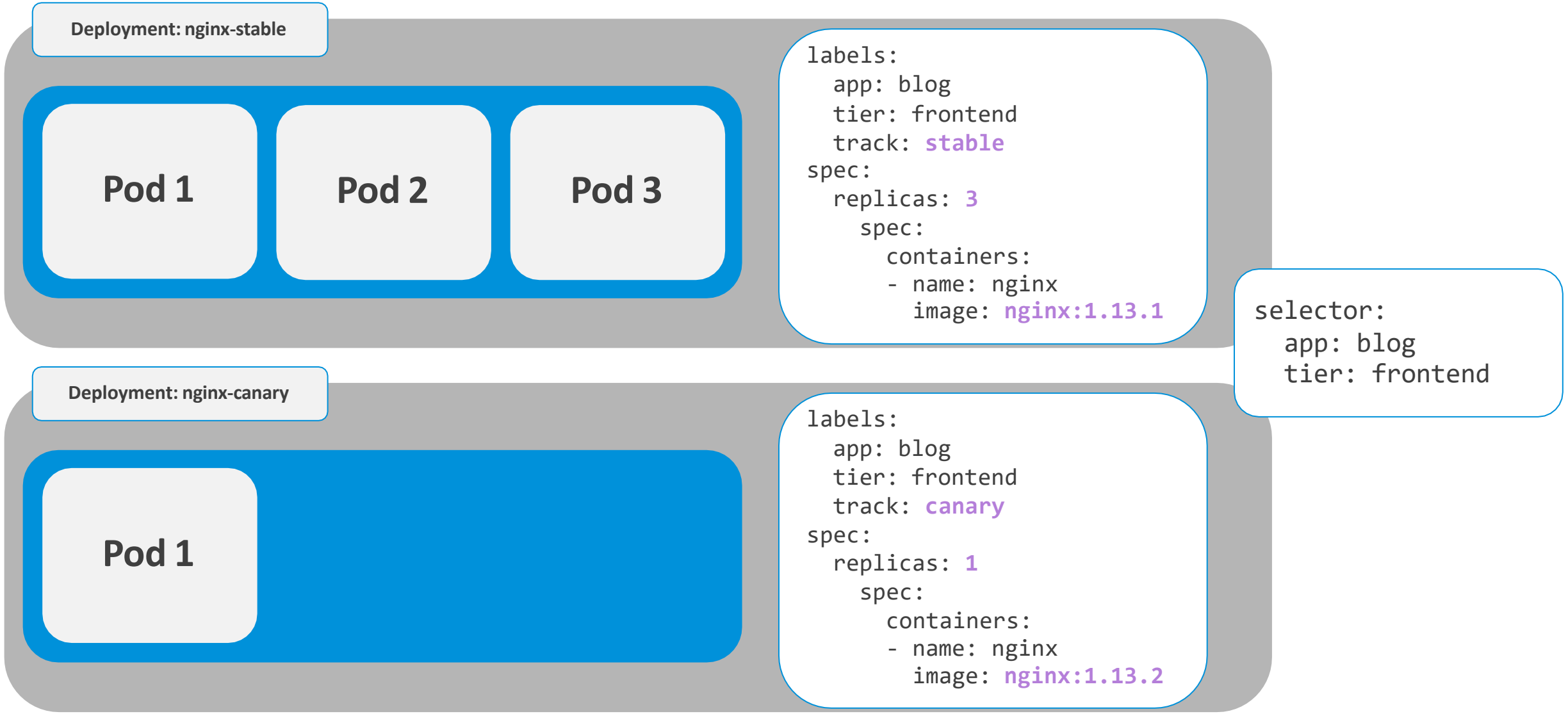
Blue/Green deployment:

- Deploy all new containers, test, and flip or switch traffic to new containers
- Involves two deployments and label management between them

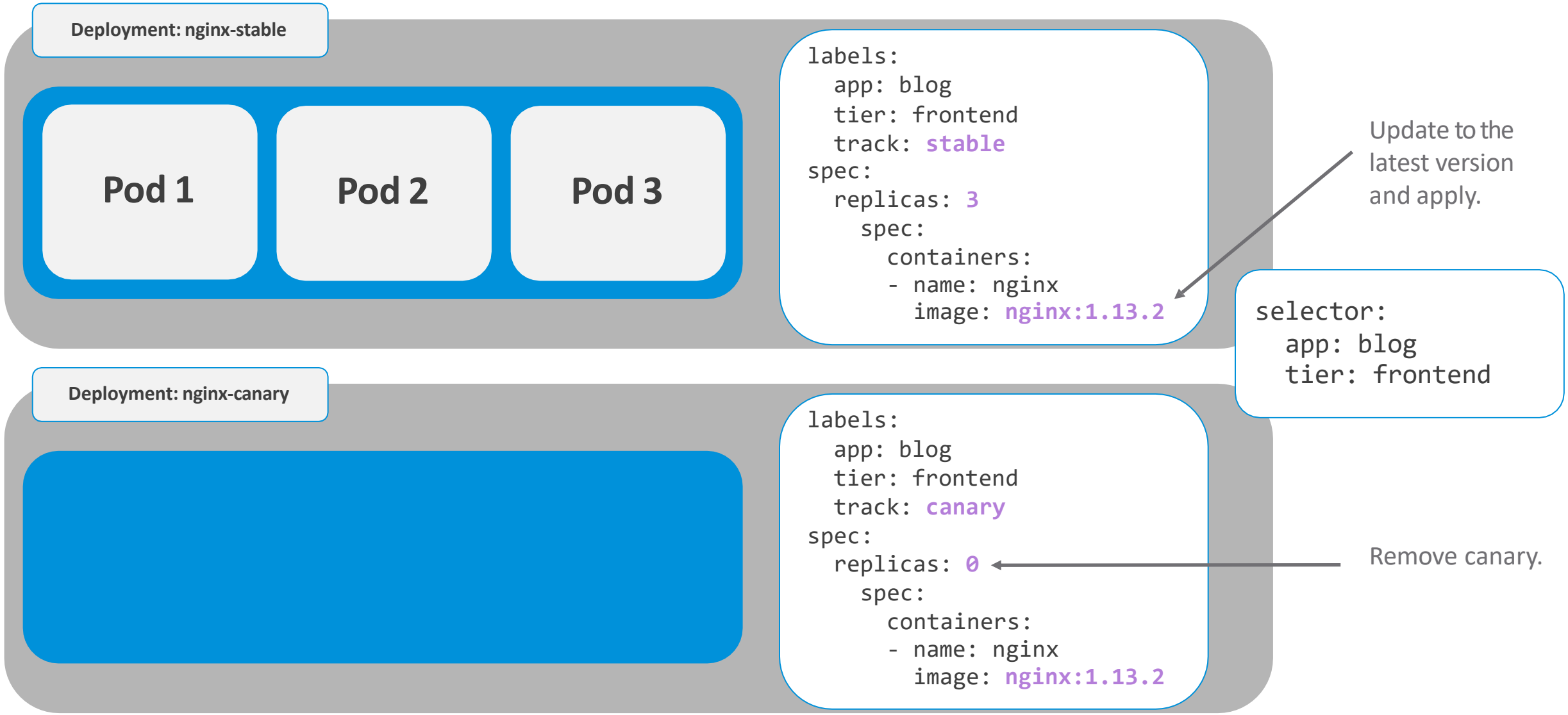
Deployment Strategies: Canary (1)



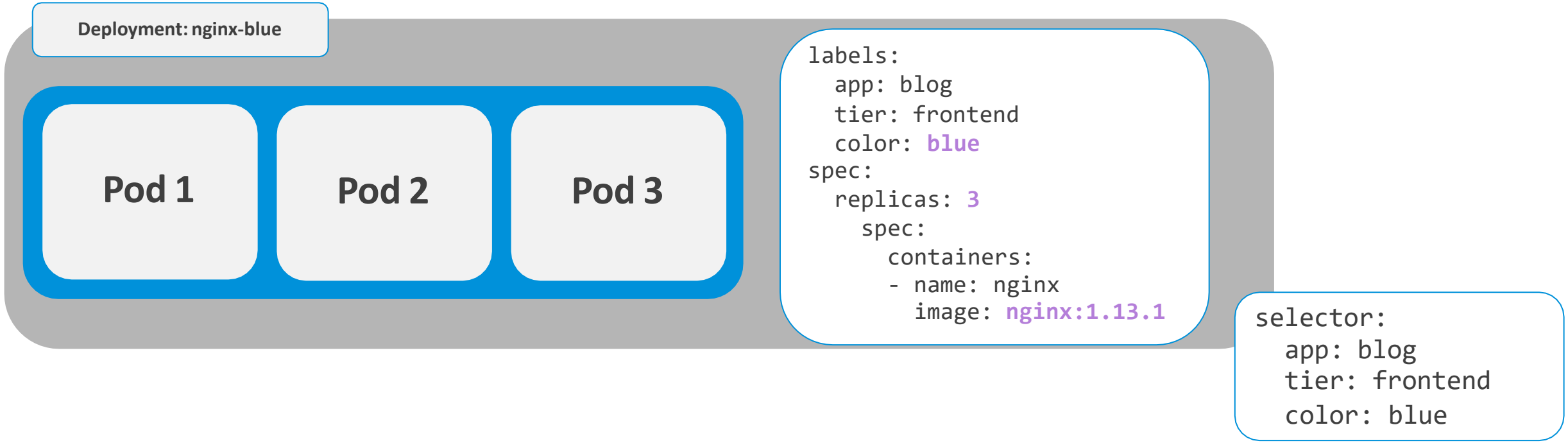
Deployment Strategies: Canary (2)



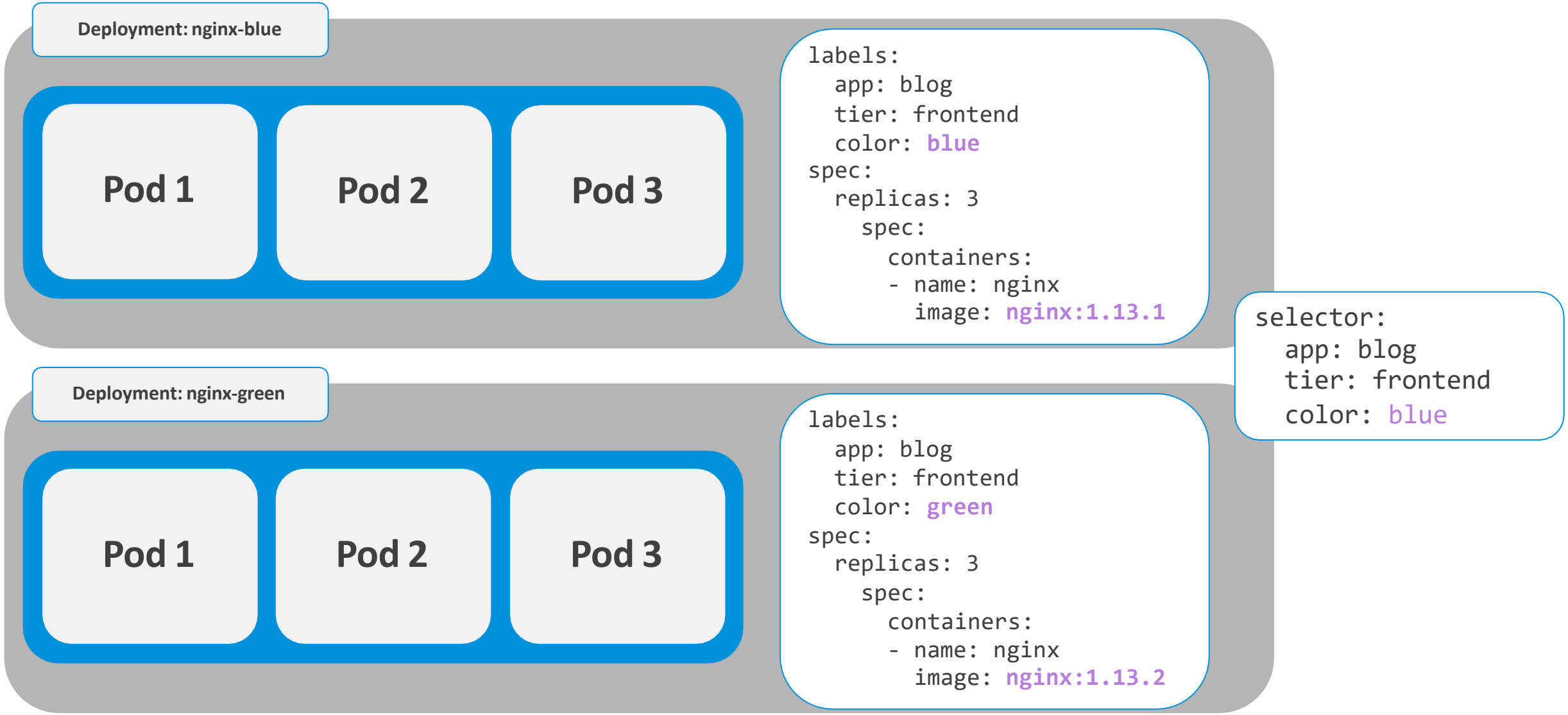
Deployment Strategies: Canary (3)



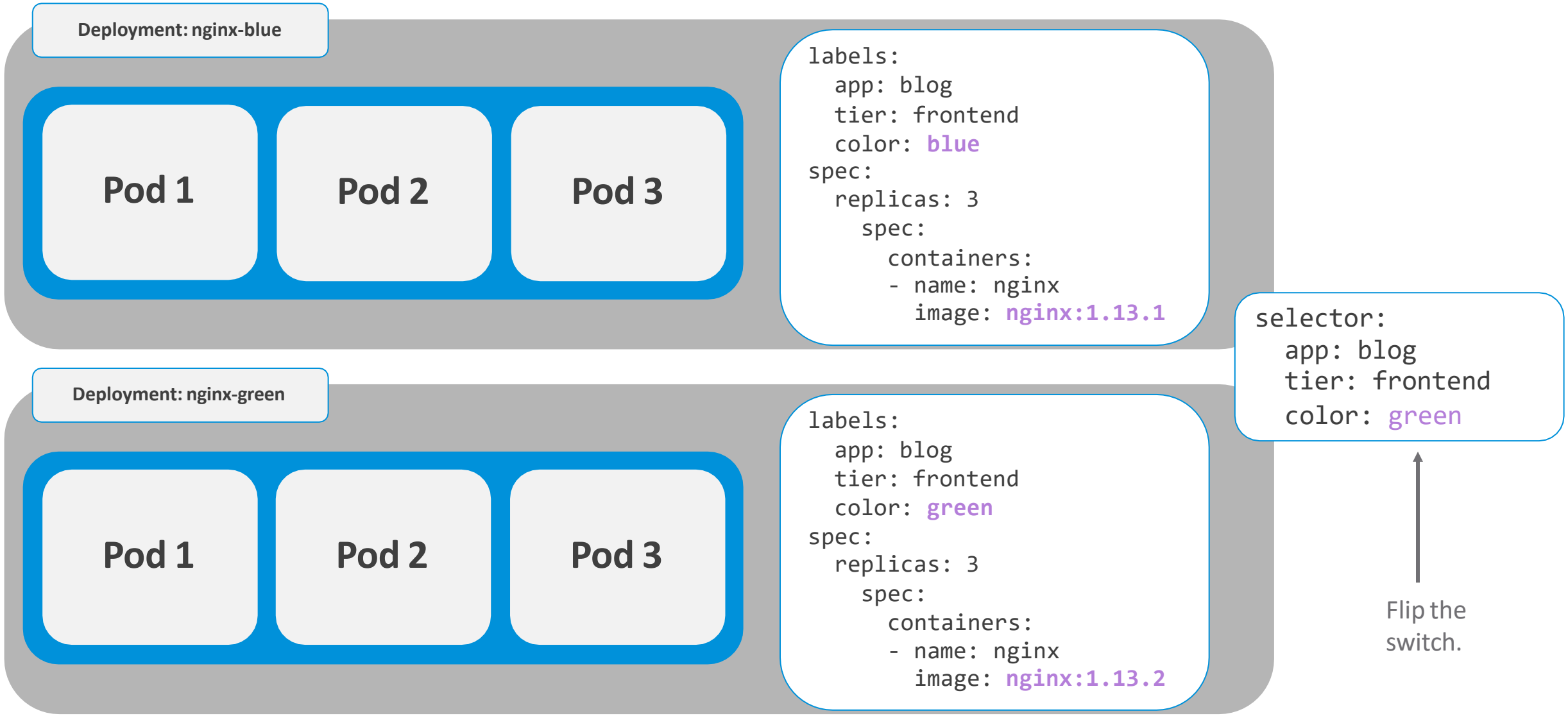
Deployment Strategies: Blue/Green (1)



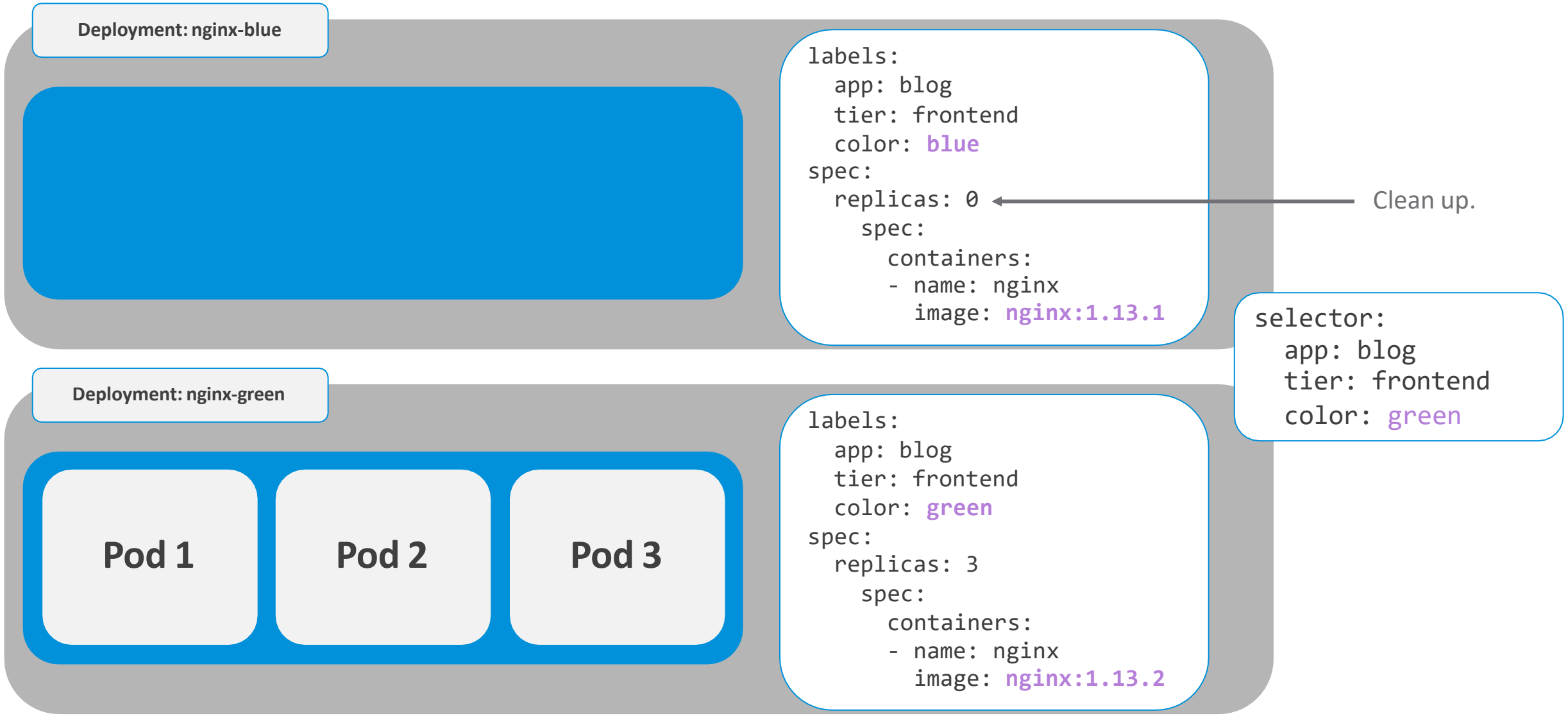
Deployment Strategies: Blue/Green (2)



Deployment Strategies: Blue/Green (3)



Deployment Strategies: Blue/Green (4)



Controlling Deployments

Pausing:

```
kubectl rollout [ pause | resume ] deployment <name>
```

Rolling back to a previous version:

- Declarative (preferred):

  Modify the deployment's YAML file and re-apply

  ```
  kubectl apply -f deployment.yaml
  ```

- Imperative:

  ```
  kubectl rollout undo deployment <name>
  ```

Pod and Container Configurations

KOENIG
step forward

About Probes

A probe is a diagnostic performed periodically by the kubelet on a container. You can configure liveness and readiness probes for containers:

- Liveness:
  - Identifies whether the container is running.
  - On failure, the container is restarted according to policy.
- Readiness
  - Identifies whether the container is ready to service requests.
  - On failure, the pod is removed from the service, so no requests are sent to it.

Probe Handlers

To perform a diagnostic, the kubelet calls a handler implemented by the container. The following types of handlers can be used:

- Exec:
  - Run a command inside the container.
  - Success: Return code from command is 0.
- TCPSocket
  - TCP check.
  - Success: Port is open and accepting connection.
- HTTPGet
  - Invoke HTTP GET against a URL.
  - Success: Any 2xx or 3xx HTTP response.

Use Cases

Cache initialization:

- Specify readiness probe to confirm caches are hot.

Ensure JVM starts successfully:

- Specify a liveness probe against URL or looks for particular line in logs.

Container maintenance:

- Container can take itself out of service by failing either probe.

Probe Options

initialDelaySeconds:

- How long to delay probing after container is started.

periodSeconds:

- How often to perform the probe. Default = 10.

timeoutSeconds:

- Default = 1.

successThreshold:

- Minimum consecutive successes for the probe to be considered successful. Default = 1.

failureThreshold:

- Minimum consecutive failures for the probe to be considered failed. Default = 3.

Resource Requests

A resource request can be thought of as minimum resource required specification:

- Resource requests help Kubernetes schedule pods efficiently.
- Pods are scheduled if the sum of the resource requests of the scheduled containers is less than the capacity of the node.

```
containers:
 - name: db
   image: mysql
   resources:
     requests:
       memory: "512Mi"
       cpu: "0.5"
```

Resource Limits

Resource limits protect against a runaway application.

If a container exceeds its memory limit, it might be terminated.

If a container exceeds its memory request, it is likely that its pod will be evicted whenever the node runs out of memory.

```
containers:
 - name: db
   image: mysql
   resources:
     limits:
       memory: "1Gi"
       cpu: "1"
```

Resource Management: Defining CPU Values

CPU units:

- 1 AWS vCPU
- 1 GCP Core
- 1 Azure vCore
- 1 hyperthread on a bare-metal Intel processor with hyperthreading

CPU values:

- 0.1 = 100 m
- One hundred millicpu / 10% of a CPU unit

Resources: Defining Memory Values

Memory units:

- T, G, M, K: 10 based
- Ti, Gi, Mi, Ki: Power of 2 based

Memory values:

- 512M != 512 Mi
- Easier to just specify the true/proper i suffix style

Container Networking

KOENIG
step forward

CNI Providers

Common Kubernetes Container Network Interface (CNI) implementations that affect networking include:

- Kubenet
- Flannel/Calico
- Weave Net
- Common VMware Tanzu choices:
  - Contour
  - NSX-T Data Center
  - NSX Advanced Load Balancer (Avi)
  - Antrea
  - Project Calico
  - HAProxy

Networking: Within a Pod

The containers within a pod have the following capabilities:

- Can connect to each other using localhost
- Share an IP address accessible throughout the cluster
- Share a common port space (but beware of conflicts)

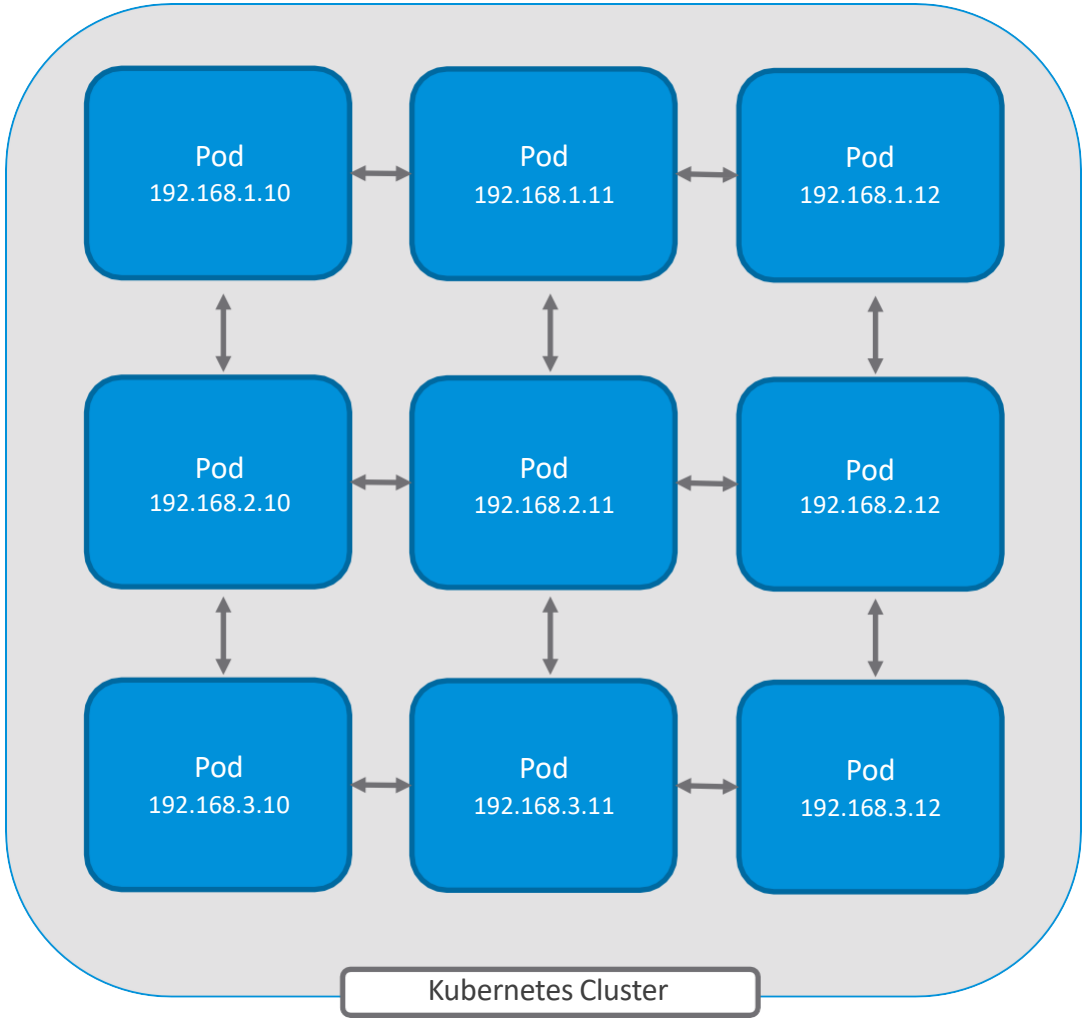
These capabilities closely mimic those of multiple processes running on the same virtual machine.

Networking: Pod to Pod

Every pod is assigned an IP address.

This IP address is routable anywhere within the cluster.

The implementation of connectivity between all pods in the cluster is the responsibility of the CNI that was implemented.



Networking: Services to Pods

A service is a Kubernetes resource:

- Layer 4 load balancing for a group of pods
- Service discovery using the cluster's internal DNS

Several types of services are available:

- ClusterIP
- NodePort
- LoadBalancer

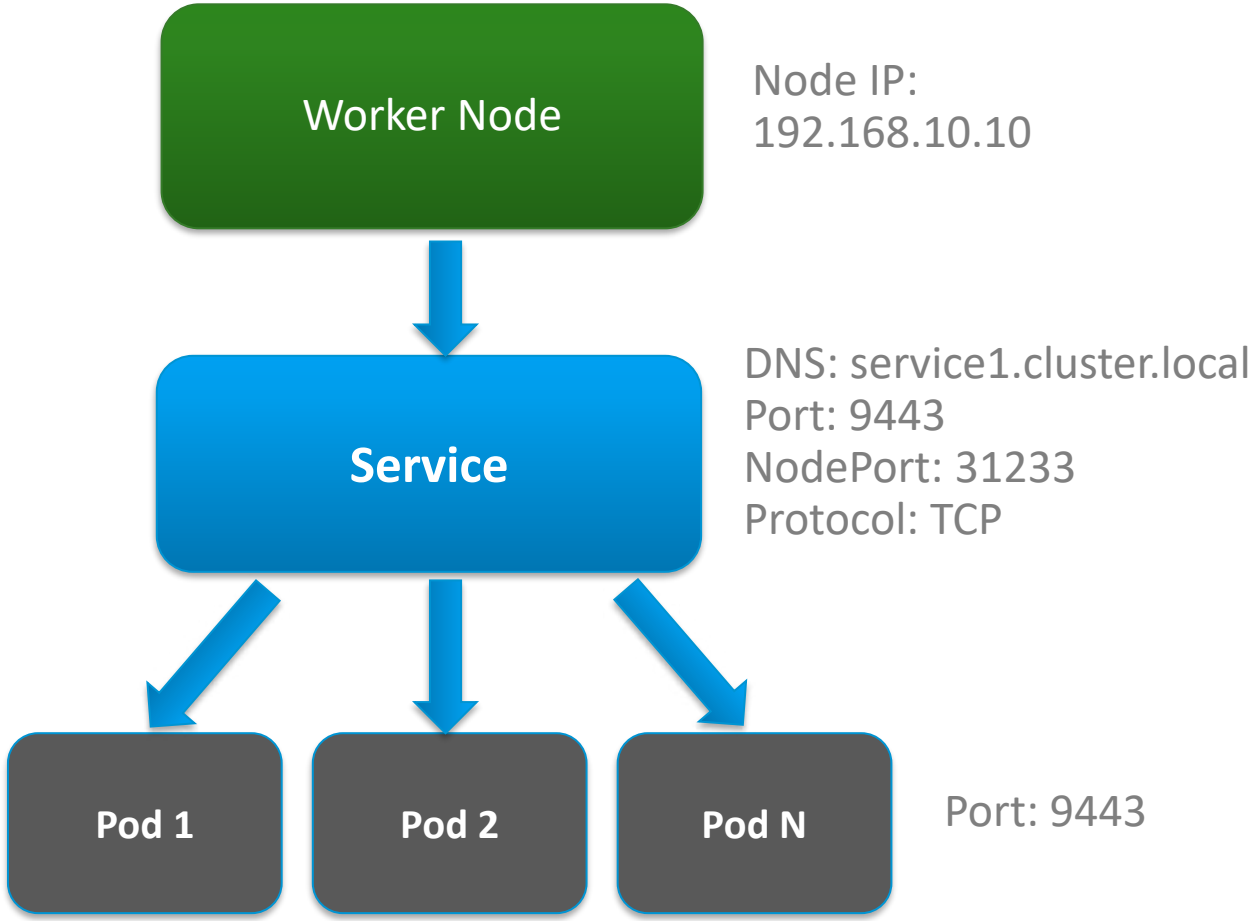
Kubernetes Services

Services provide reliable networking.

Pods are ephemeral by design:

- Pods fail and can be replaced.
- New pods get a new IP address.
- Scale up: New pods can be introduced.
- Scale down: Pods can be removed.

A service is an object in the Kubernetes API

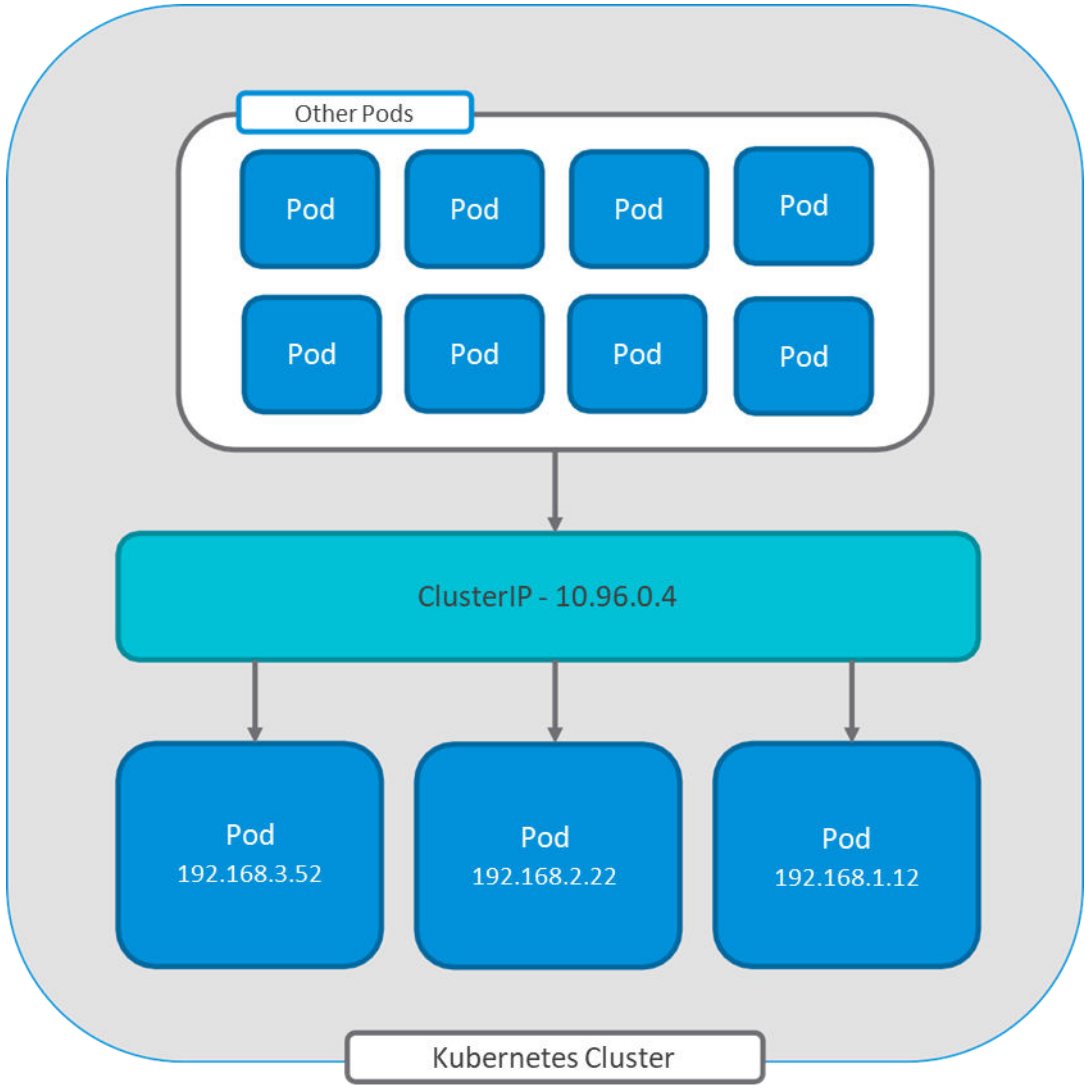


Service Type: ClusterIP

The ClusterIP service type is used for internal-facing services.

Implementation:

- A virtual IP address load balances requests to a set of back-end pods.
- Accessible anywhere within the cluster.
- Not externally accessible.

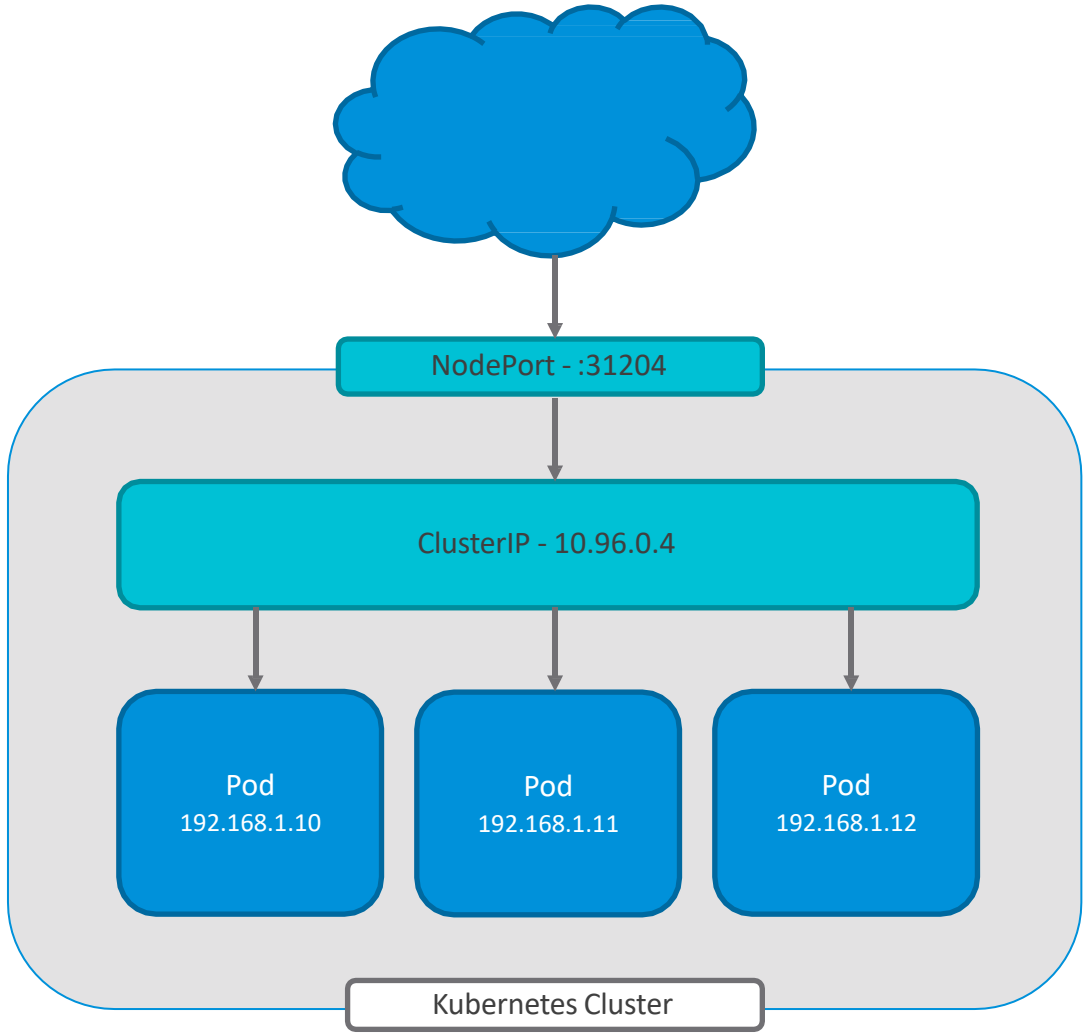


Service Type: NodePort

The NodePort service type is used for external-facing services.

Implementation:

- Exposes a port on each worker node
- Externally accessible
- Uses ClusterIP for load balancing to pods

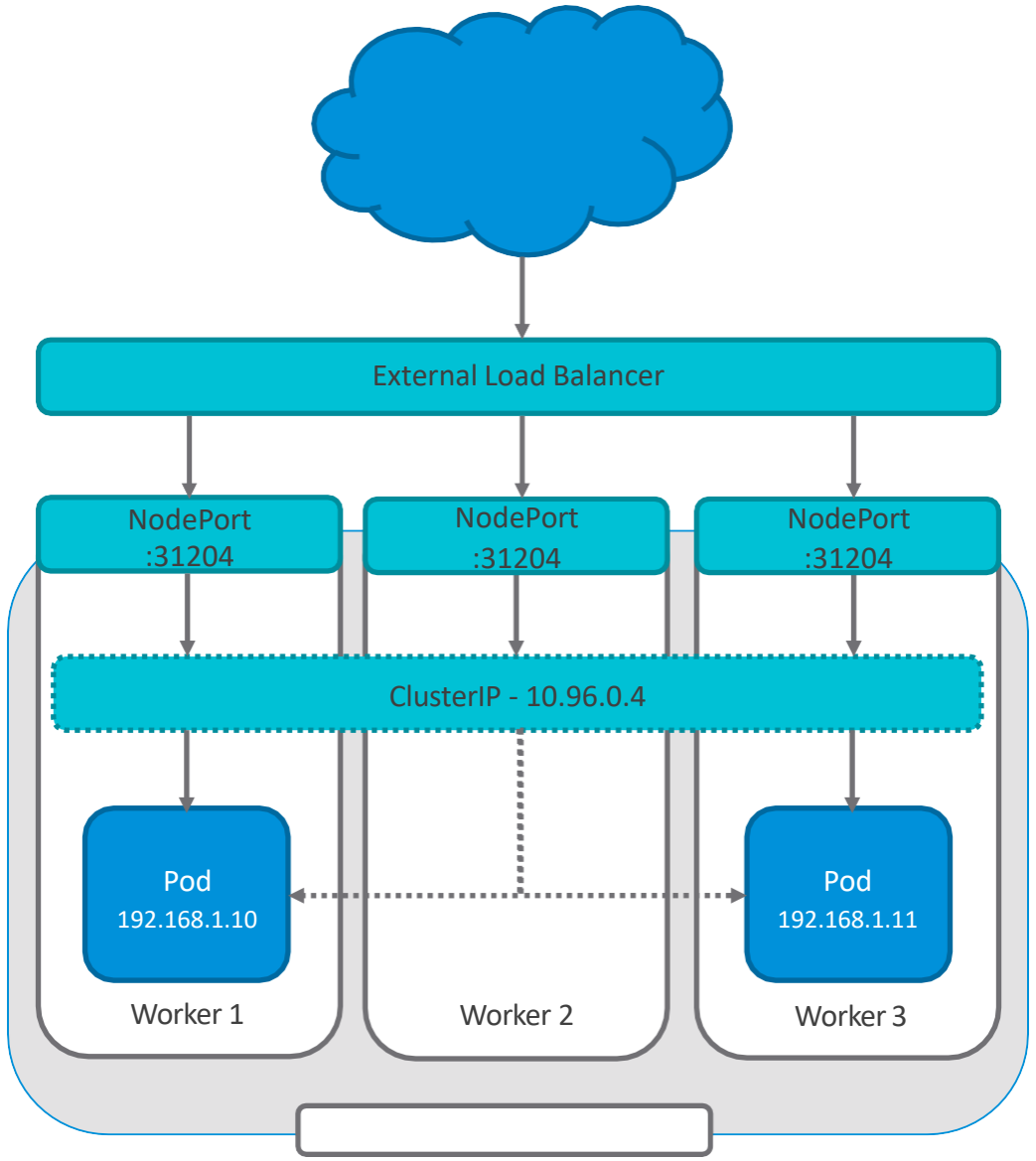


Service Type: LoadBalancer

The LoadBalancer service type creates and manages an external load balancer.

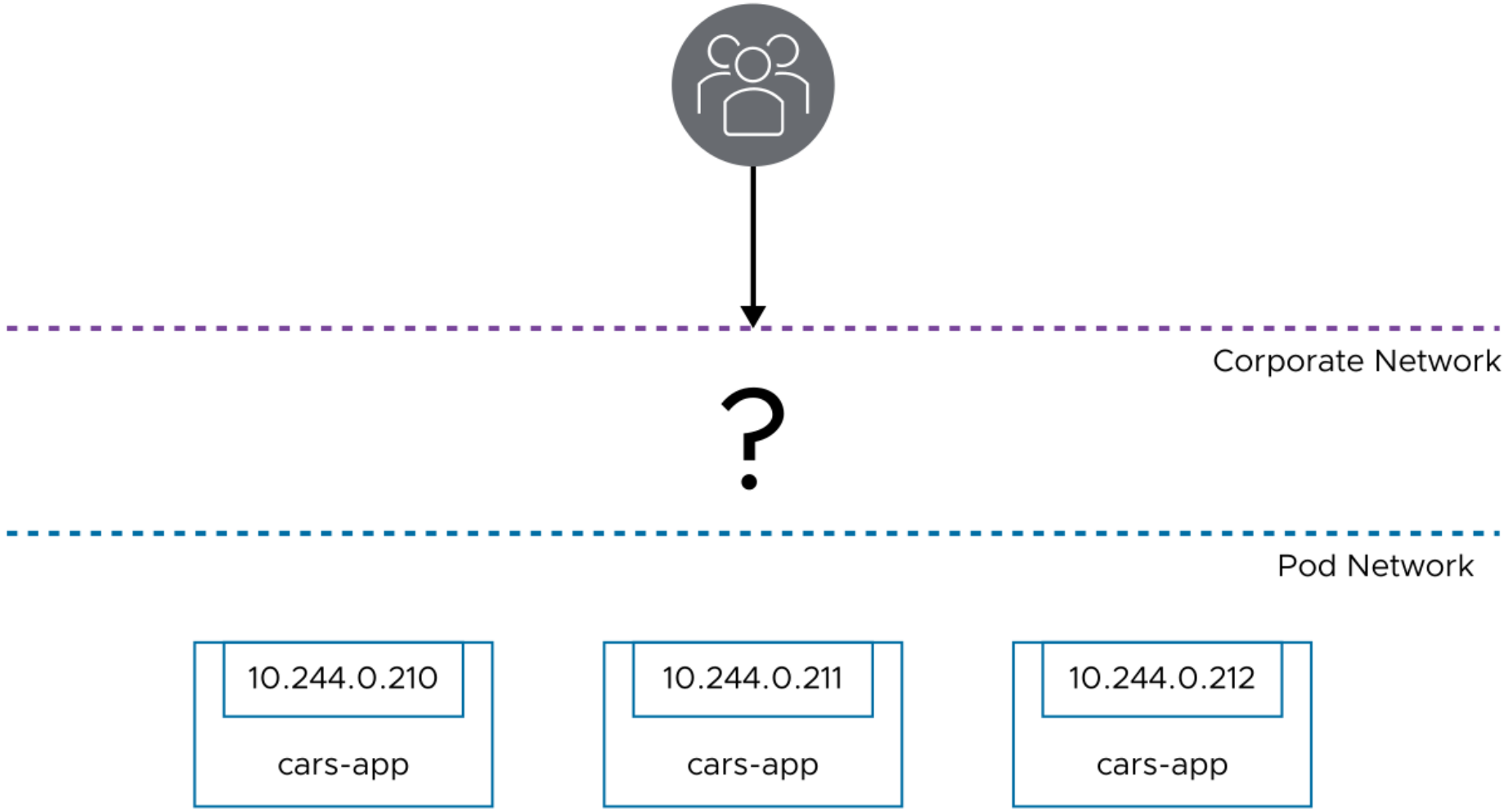
Implementation varies based on the selected CNI or LoadBalancer:

- You can use NodePort for traffic ingress, which uses ClusterIP for load balancing to pods.
- Plug-ins are available for different implementations.
- The NSX-T Data Center load balancer connects directly to the pod.



Kubernetes Services (1)

Typically, pods run on a network that is not routable. To provide external access into pods, you use services.



Kubernetes Services (2)

A service describes how pods discover and communicate with each other and external networks.

A service exposes pods as a single IP address.

Deploy a Service example:

- Service name is cars-service.
- It is a LoadBalancer service.
- It should listen on port 80.

```
apiVersion: v1
kind: Service
metadata:
  name: cars-service
spec:
  type: LoadBalancer
  selector:
    app: cars-app
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

Kubernetes Services (3)

A developer can create a Kubernetes service of the type LoadBalancer.

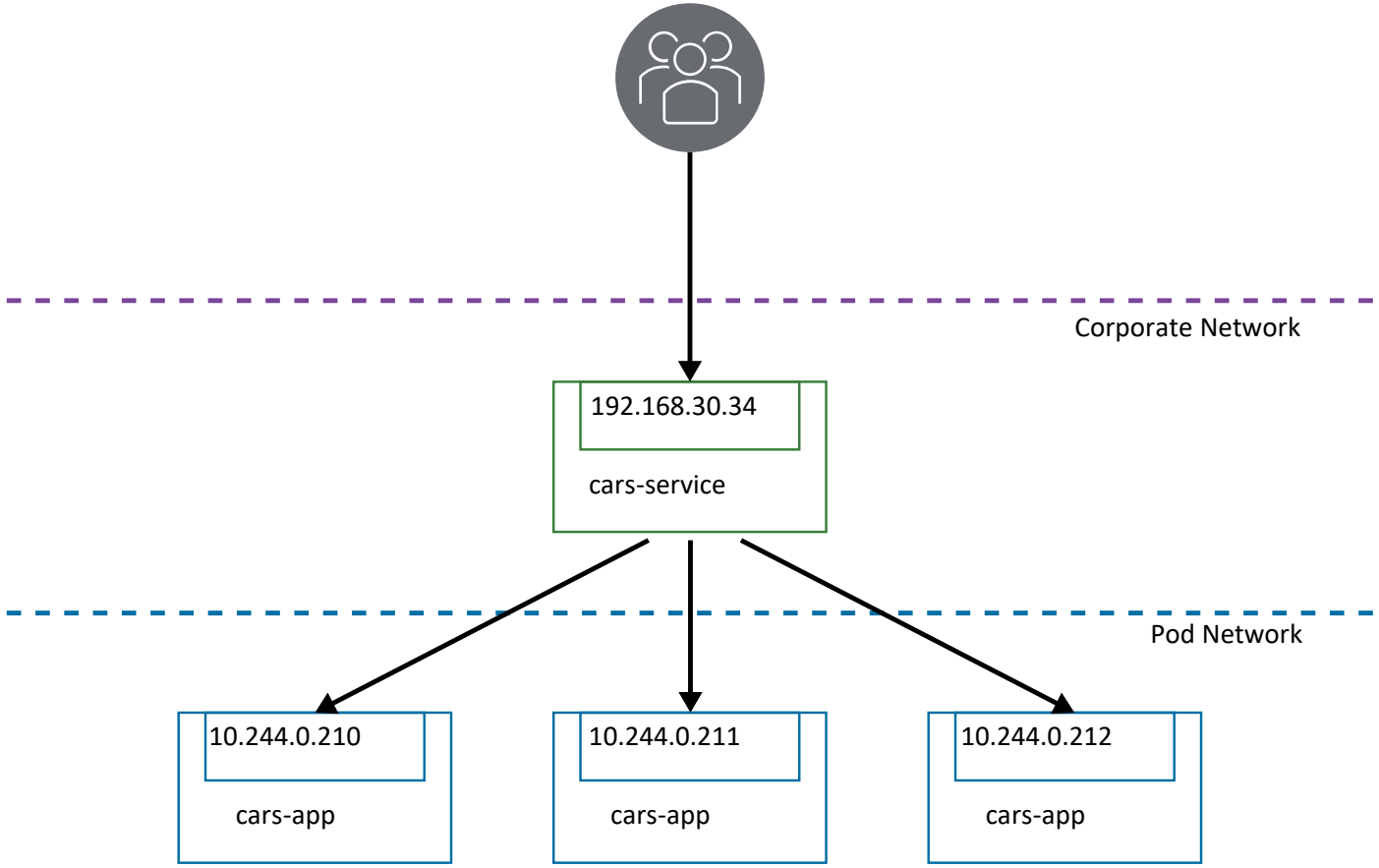
The implementation of LoadBalancer depends on the chosen CNI provider.

```
root@sa-cli-vm [ ~ ]# kubectl get pods,svc -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP
pod/cars-app-56957d7b94-779pv     1/1     Running   0          38m   10.244.0.211
pod/cars-app-56957d7b94-8cxcr     1/1     Running   0          38m   10.244.0.212
pod/cars-app-56957d7b94-s2xh9     1/1     Running   0          38m   10.244.0.210

NAME                    TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
service/cars-service    LoadBalancer   10.96.0.37    192.168.30.34   80:30836/TCP
root@sa-cli-vm [ ~ ]#
```

Kubernetes Services (4)

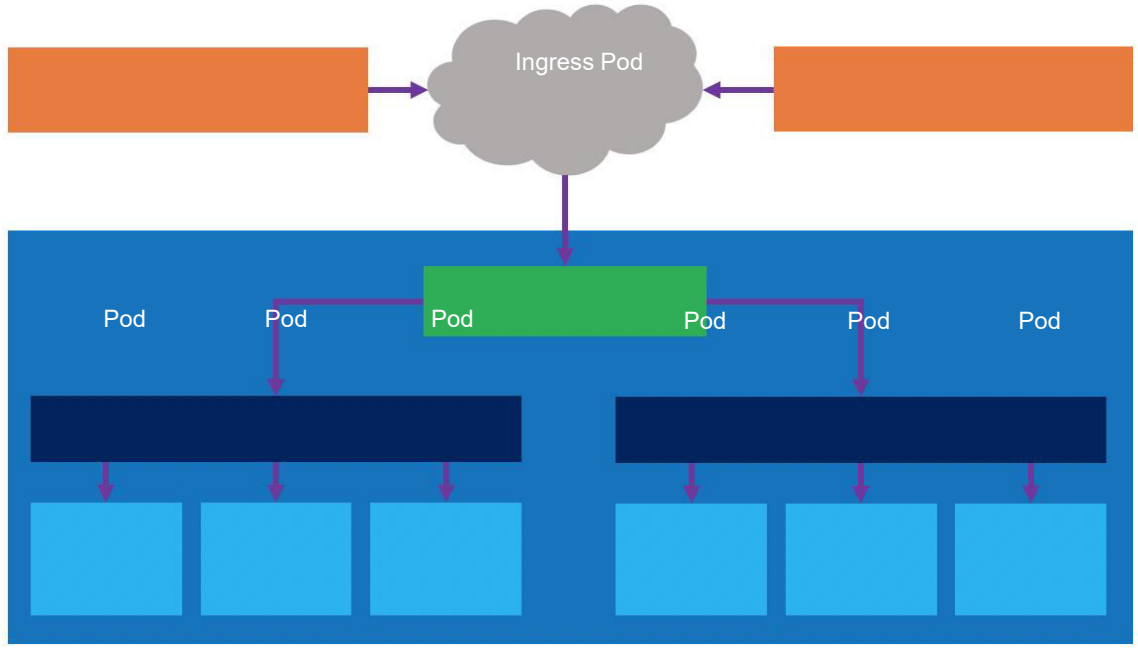
A LoadBalancer service creates an externally accessible IP address and depends on the chosen CNI.



About Ingress

Ingress in Kubernetes is a method for routing external traffic to pods:

- Operates on layer 7 traffic
- Routes requests based on HTTP headers
- Has additional capabilities depending on the implementation.
- Can be implemented by different Ingress controllers:
  - Contour
  - NSX-T Data Center
  - NSX Advanced LB (AVI)
  - NGINX
  - Traefik
  - Amazon Application Load Balancer (ALB)

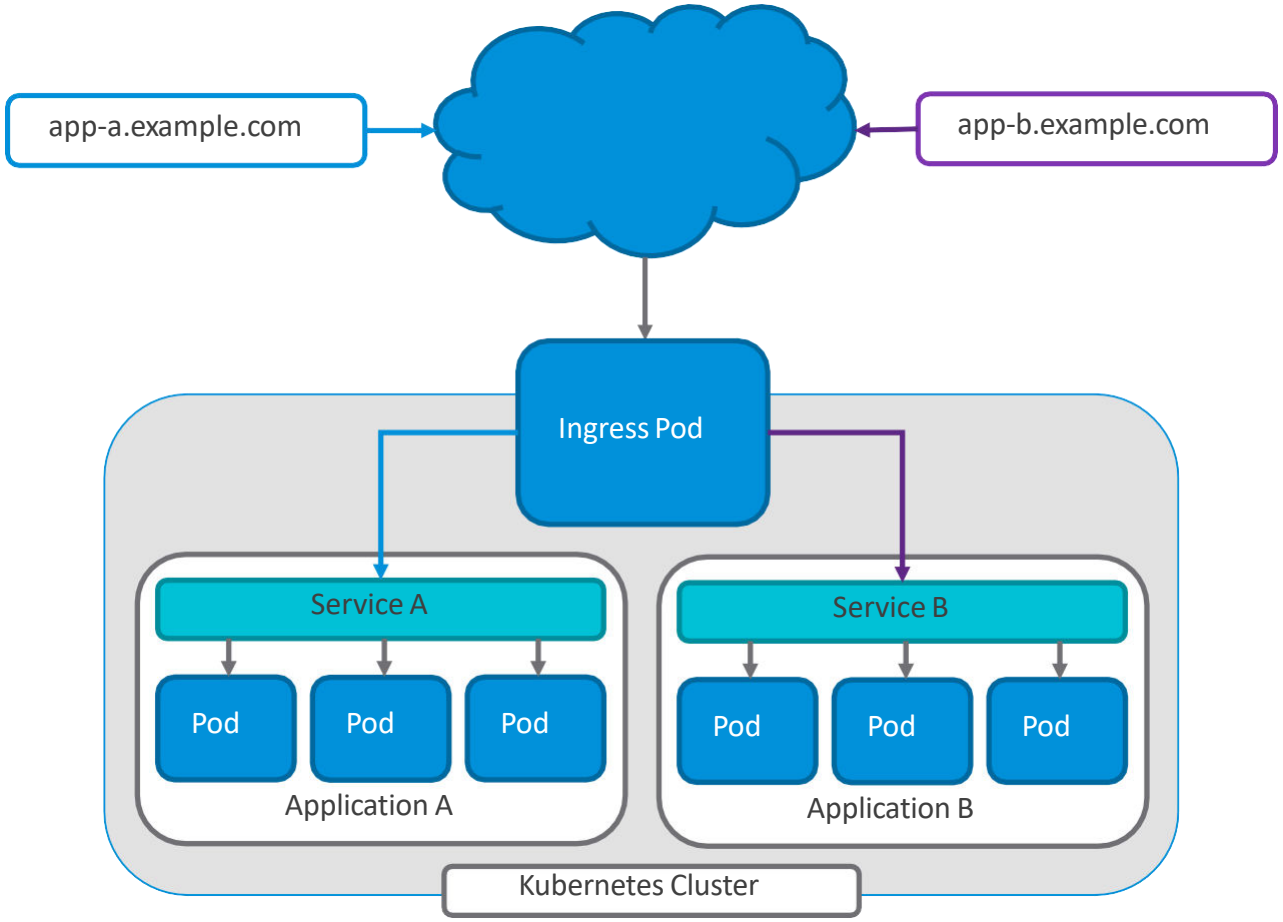


Ingress Controller

The ingress controller is used for external-facing layer 7 services.

Implementation:

- Uses host header and path evaluation to direct traffic
- Externally accessible
- Configured with the ingress object



Ingress Example

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: my-layer7-apps
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: app-a.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: service-app-a
          servicePort: 8080
  - host: app-b.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: service-app-b
          servicePort: 8080
```

About Service Discovery

Kubernetes provides a declarative API for deploying applications, services, ingress, and load balancer resources.

Services, ingress, and load balancer resources provide methods to route IP network traffic to an application, but IP addresses can change as workloads are created and destroyed.

Service discovery solves this problem by allowing applications to be registered in a service registry.

The service registry can be queried by applications or end-users that need to access a specific application.

Container Persistent Storage

KOENIG
step forward

Storage Design

Personas

- Storage administrators
- Application engineers

Abstractions

- PodSpec does not contain file path references outside the container.
- Where the storage comes from depends on the infrastructure and the Container Storage Interface (CSI plug-in) that is selected.

Container Storage Interface

The CSI provides an API between Kubernetes and storage providers:

- CSI provides a common standard for exposing storage systems.
- Third-party providers can enable new storage systems using plug-ins.

Storage: Need for Persistence

By default, containers write to ephemeral storage:

- When a pod is terminated, all data written by its containers is lost.
- To have persistent storage, persistent volumes are attached to the pod.

Storage Volumes

Storage volumes are defined at pod level:

- Path on worker node is abstracted away.

Ephemeral volumes:

- emptyDir is local to the worker node and shared across containers within the pod.

```
kind: Pod
apiVersion: v1
metadata:
    name: mypod1
    labels:
      app: blog
spec:
    containers:
    - name: nginx
      image: nginx:1.13.1
      volumeMounts:
      - mountPath: /usr/local/nginx/html
        name: my-web-content
    - name: www-syncer
      image: syncer:1.2
      volumeMounts:
      - mountPath: /synced-data
        name: my-web-content
  volumes:
    - name: my-web-content
      emptyDir: {}
```

Persistent Storage

Although it is relatively easy to run stateless microservices using container technology, stateful applications require different treatment. Multiple factors must be considered when handling persistent data that use containers:

- Containers are ephemeral. The data that needs to be persisted has to survive through the restart or rescheduling of a container.
- When containers are rescheduled, they can cease to exist on one host and might be scheduled on a different host. The storage should also be shifted and made available on the new host for the container to start gracefully.
- The application does not handle the volume and data. The underlying infrastructure should handle the complexity of unmounting and mounting.
- Certain applications have a strong sense of identity (for example, Kafka and Elastic), and the disk used by a container with a certain identity is tied to it. If a container with a certain ID is rescheduled, the disk associated with that ID must be reattached to the new container instance.

Storage: Persistent Options (1)

Kubernetes persistent volumes (PV) are managed by storage administrators.

PVs have the following characteristics:

- Abstraction away from the storage provider through the CSI (vSphere, AWS EBS, GCE PD, Azure Disk).
- Similar to a volume but have a life cycle that is independent of any individual pod that uses the PV.
- A storage resource in the cluster that is provisioned by an administrator.
- Can be provisioned statically or dynamically.

Storage: Persistent Options (2)

The Kubernetes persistent volume claim (PVC) is used by application engineers:

- Request for storage with specific details (size, access modes, and so on).
- Acts like a search query to find a PV that is best match.
- A PVC consumes PV resources in a similar was as a pod, which consumes node resources, but a PVC can request specific size and access modes (for example, can be mounted read-write or read-only).
- When a PVC is deleted, the data is deleted by default and is controlled by the reclaim policy.

Storage: Persistent Options (3)

Kubernetes storage classes are used by storage administrators:

- Define and assign storage policies to a namespace.
- vSphere storage policies are translated into Kubernetes storage classes.
- Developers can access all assigned storage policies in the form of storage classes.
- Developers cannot manage storage classes.

Kubernetes Storage Primitives: Storage Class

A storage class enables administrators to describe the classes of storage that they offer.

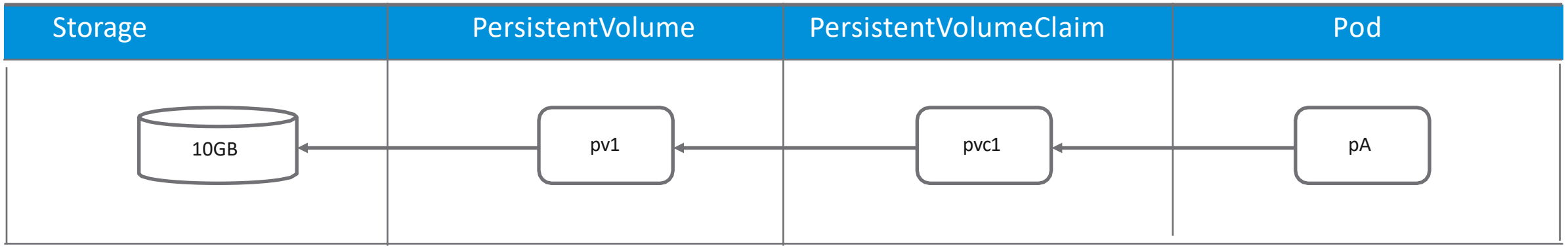
Each storage class contains the fields provisioner, parameters, and reclaimPolicy, which are used when a persistent volume belonging to the class must be dynamically provisioned.

Different classes might map to quality-of-service levels, backup policies, or arbitrary policies determined by cluster administrators.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
provisioner: kubernetes.io/vsphere-volume
parameters:
  diskformat: thin
  hostFailuresToTolerate: "1"
  datastore: vsanDatastore
reclaimPolicy: Retain
```

Storage: Relationship Examples

To implement persistent storage, several different objects must be modified or created. The table shows the various objects and their relationship to each other.



Using Persistent Volumes

To use a persistent volume, a Persistent Volume Claim (PVC) object must be created. The Pod spec must reference the PVC name.

PersistentVolumeClaim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mysql-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
```

PVC in a Pod

```
kind: Pod
apiVersion: v1
metadata:
  name: mypod
spec:
  containers:
    - name: my-mysql
      image: mysql:5.6
      volumeMounts:
      - mountPath: "/var/lib/mysql"
        name: mysql-pd
  volumes:
    - name: mysql-pd
      persistentVolumeClaim:
        claimName: mysql-pvc
```

VM Storage Policies and Kubernetes Storage Classes

Developers can list the available storage classes:

- In their namespace, by running the `kubectl describe ns <namespace-name>` command.
- On the cluster, by running the `kubectl get storageclasses` command.

```
root@sa-cli-vm [ ~ ]# kubectl describe ns namespace-01
Name:         namespace-01
Labels:       vSphereClusterID=domain-c22
…
Status:       Active
Resource Quotas
 Name:
    namespace-01-storagequota
 Resource
    Used  Hard
 --------
    ---   ---
 bronze-storage-
    policy.storageclass.storage.k8s.io/requests.storage  0
    9223372036854775807
 silver-storage-
    policy.storageclass.storage.k8s.io/requests.storage  0
    9223372036854775807

No resource limits.
root@sa-cli-vm [ ~ ]#
```

```
ubuntu@sa-master-01:~$ kubectl get storageclasses
NAME          PROVISIONER                    RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
test-class    kubernetes.io/vsphere-volume   Delete          Immediate           false                  22m
ubuntu@sa-master-01:~$
```

Dynamic Provisioning of Persistent Volumes