UNIVERSITÀ
CATTOLICA
del Sacro Cuore

Kubernetes Details

INTRO TO KUBERNETES

kubernetes

KEY CONCEPTS

Container orchestrator

- Runs and manages containers
- Unified API for deploying web applications, batch jobs, and databases
- Maintains and tracks the global view of the cluster
- Supports multiple cloud and bare-metal environments

Manage applications, not machines

- Specify the application to deploy, let Kubernetes decide what machines to run it on

Designed for extensibility

- Implemented as resources, add custom resource types
- Rich ecosystem of plug-ins for scheduling, storage, networking

Open source project managed by the Cloud Native Computing Foundation in the Linux Foundation

- Inspired and informed by Google's experiences and internal systems
- 100% open source, written in Go

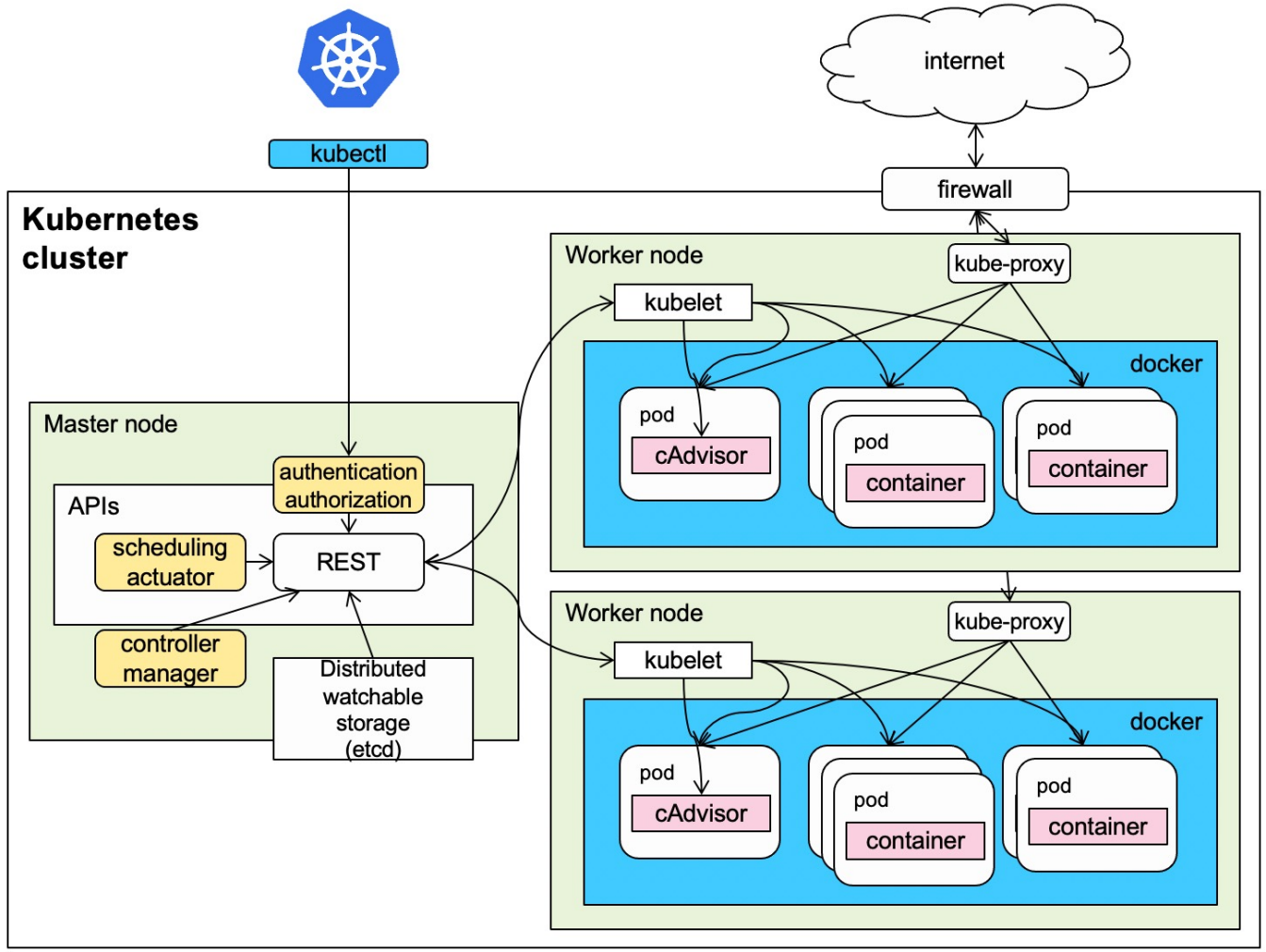
KUBERNETES ARCHITECTURE

Master nodes

- Computer with processes that manage the cluster
- Multiple nodes for High Availability

Worker nodes

- Computers that host pods
- Pods host application containers



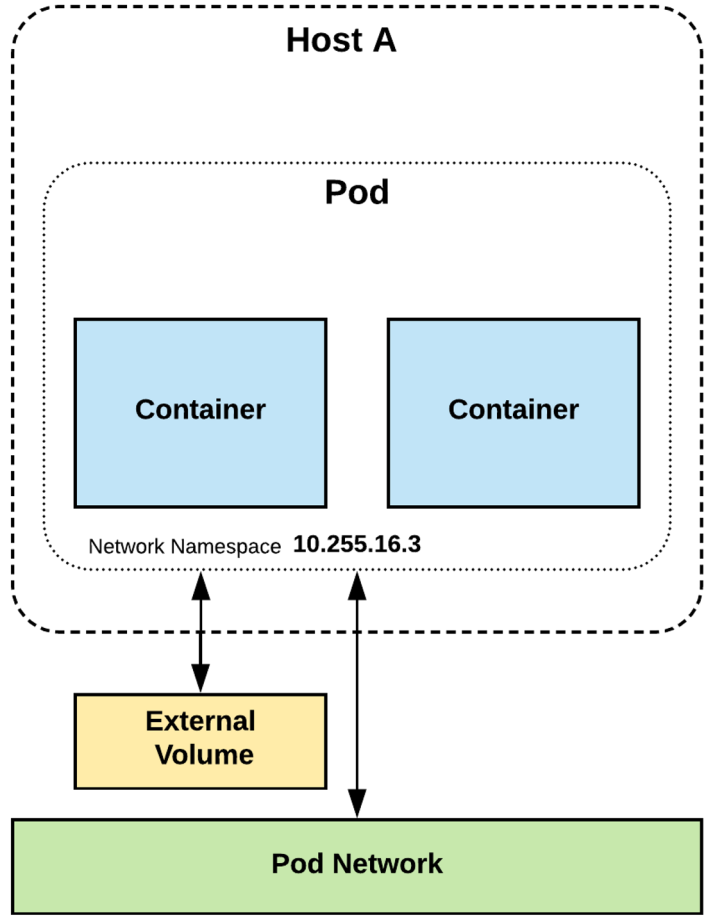
KEY CONCEPTS

- Cluster
  - Collection of worker nodes managed by the same master node
  - Makes the worker nodes behave like one big computer
- Master node
  - Controls and manages the cluster
  - Scheduling and replication logic
  - Generally two or more master nodes for resiliency, but are not used for scaling out the cluster
- Worker node
  - Node where workloads run in pods
  - Docker – Docker engine for running containers
  - kubelet
    - Kubernetes agent that accepts commands from the master
    - Manages pods in the node
  - cAdvisor – Container Advisor provides resource usage and performance statistics
  - kube-proxy – network proxy service responsible for routing activities for inbound or ingress traffic

POD

Atomic unit or smallest "*unit of work*"of Kubernetes.

Pods are **one or MORE containers** that share volumes, a network namespace, and are a part of a **single context**.



KUBERNETES ARCHITECTURE: WORKLOADS

Container

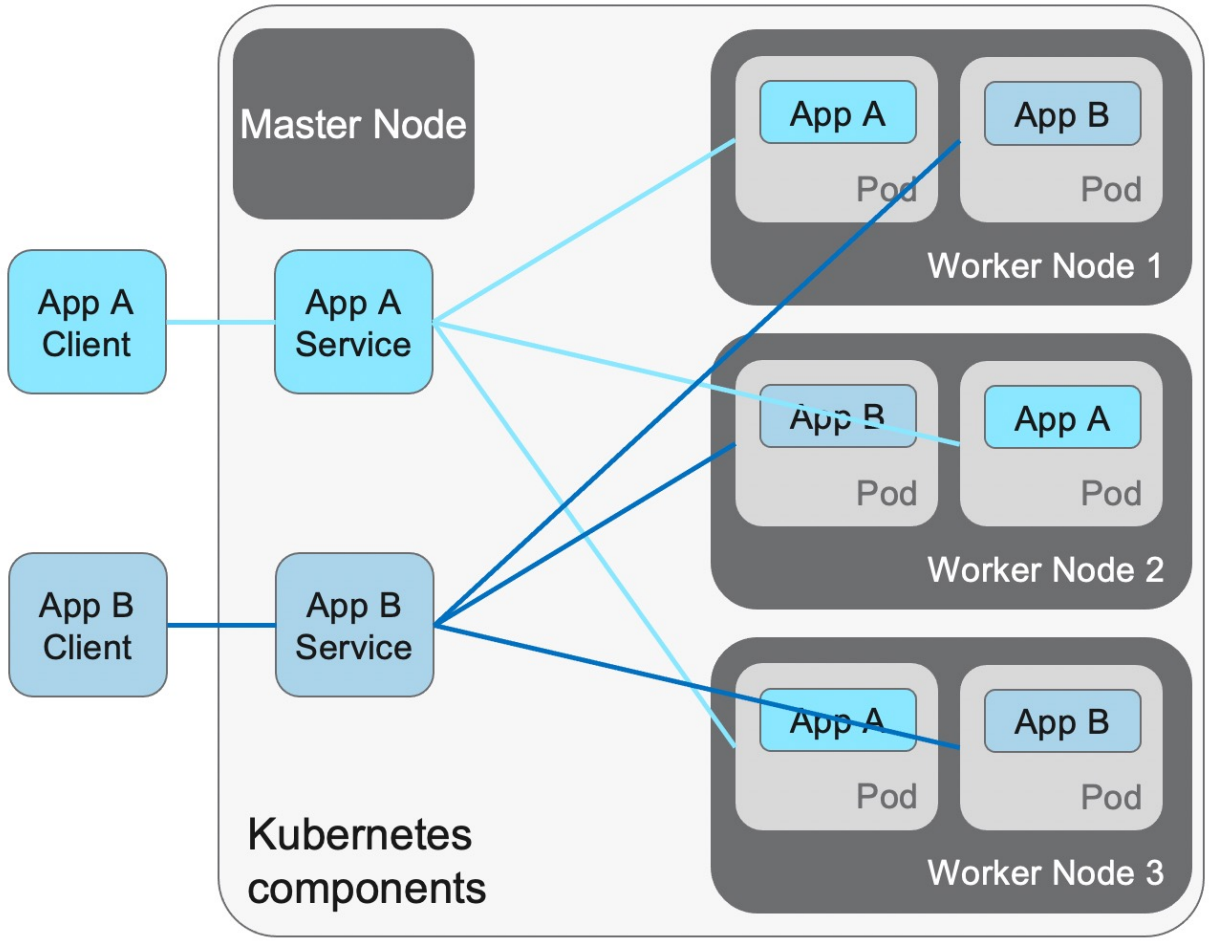
Packaging of an app

Pod

Unit of deployment

Service

Fixed endpoint for 1+ pods



KEY CONCEPTS

- Container
  - Unit of packaging
- Pod
  - Smallest deployment unit in Kubernetes
  - A collection of containers that run on a worker node
  - Each pod has its own IP
  - A pod shares a PID namespace, network, and hostname
- Service
  - Collection of pods exposed as an endpoint
  - Information stored in the Kubernetes cluster state and networking info propagated to all worker nodes
  - Types
    - ClusterIP – Exposes the service on a cluster-internal IP
    - NodePort – Exposes the service on each Node's IP at a static port
    - LoadBalancer – Exposes the service externally using a cloud provider's load balancer

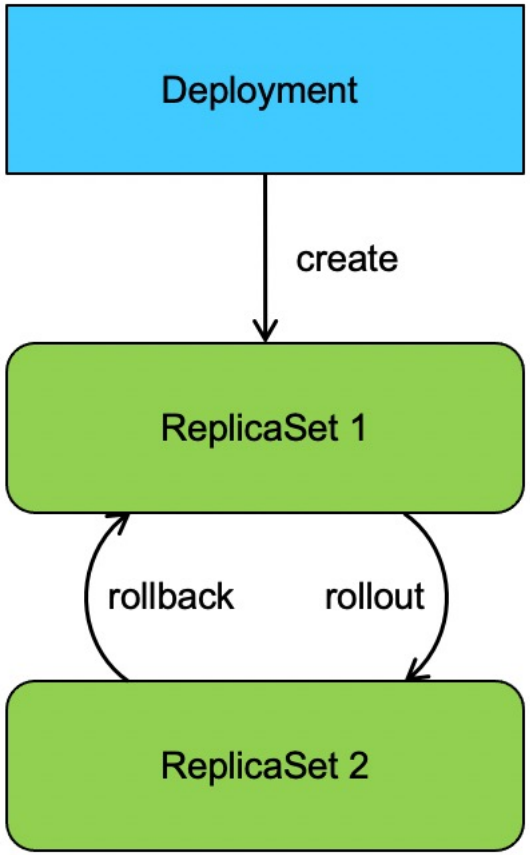
KUBERNETES TERMINOLOGY: DEPLOYMENT

Deployment

- **A set of pods to be deployed together, such as an application**
- Declarative: Revising a Deployment creates a ReplicaSet describing the desired state
- Rollout: Deployment controller changes the actual state to the desired state at a controlled rate
- Rollback: Each Deployment revision can be rolled back
- Scale and autoscale: A Deployment can be scaled

ReplicaSet

- A set of pod templates that describe a set of pod replicas
- Uses a template that describes specifically what each pod should contain
- **Ensures that a specified number of pod replicas are running at any given time**



KUBERNETES TERMINOLOGY: NAMING

- Name
  - Each resource object by type has a unique name
- Namespace
  - Resource isolation: Each namespace is a virtual cluster within the physical cluster
    - Resource objects are scoped within namespaces
    - Low-level resources are not in namespaces: nodes, persistent volumes, and namespaces themselves
    - Names of resources need to be unique within a namespace, but not across namespaces
  - Resource quotas: Namespaces can divide cluster resources
  - Initial namespaces
    - `default` – The default namespace for objects with no other namespace
    - `kube-system` – The namespace for objects created by the Kubernetes system
- Resource Quota
  - Limits resource consumption per namespace
  - Limit can be number of resource objects by type (pods, services, etc.)
  - Limit can be total amount of compute resources (CPU, memory, etc.)
  - Overcommit is allowed; contention is handled on a first-come, first-served basis

KUBERNETES CONFIGURING RESOURCES AND CONTAINERS

Label

- Metadata assigned to Kubernetes resources (pods, services, etc.)
- Key-value pairs for identification
- Critical to Kubernetes as it relies on querying the cluster for resources that have certain labels

Selector

- An expression that matches labels to identify related resources

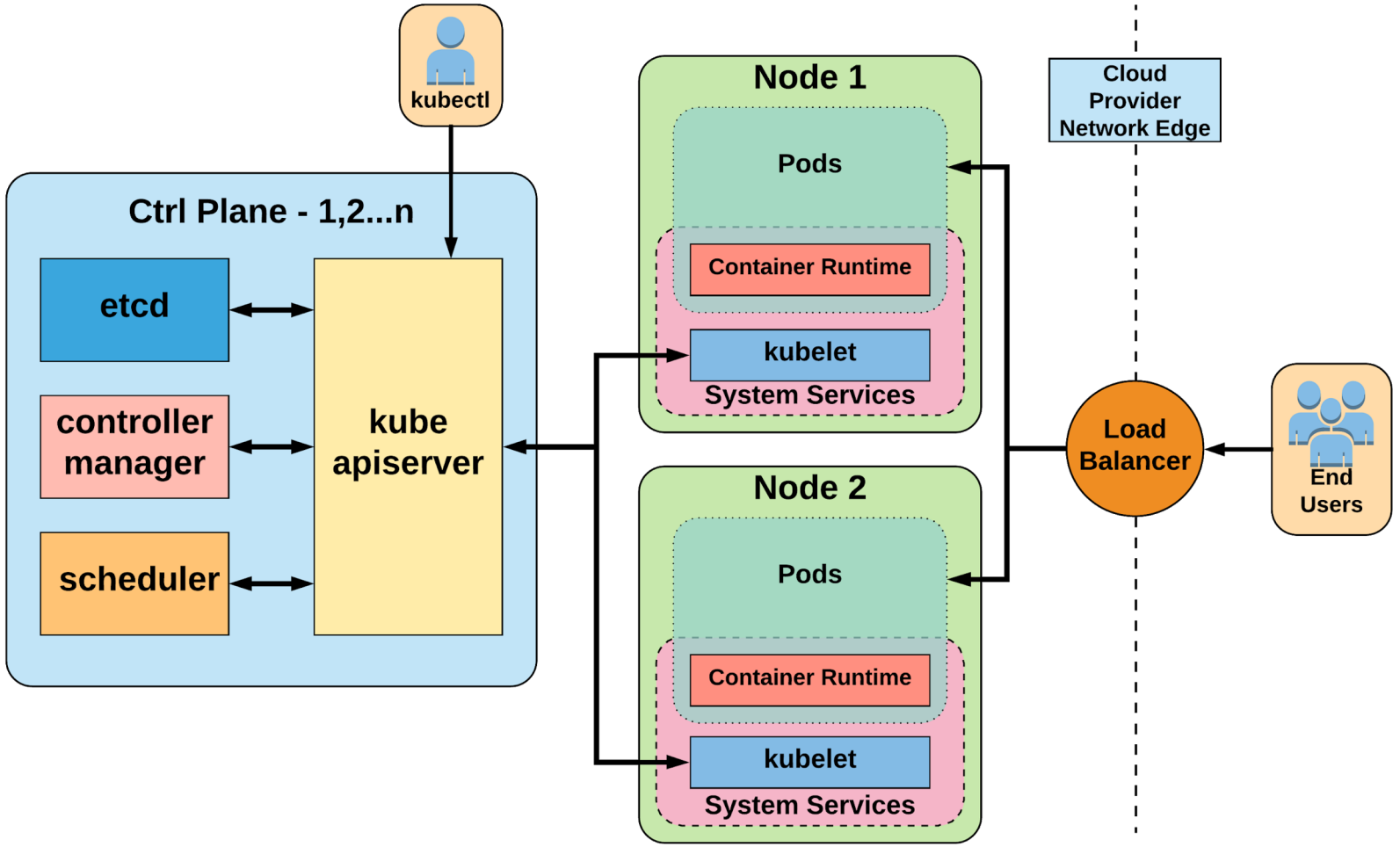
ConfigMap

- Configuration values to be used by containers in a pod
- Stores configuration outside of the container image, making containers more reusable

Secrets

- Sensitive info that containers need to read or consume
- Encrypted in special volumes mounted automatically

ARCHITECTURE RECAP



OBJECT EXPRESSION - YAML

Files or other representations of Kubernetes Objects are generally represented in YAML.

A "*Human Friendly*" data serialization standard.

Uses white space (specifically spaces) alignment to denote ownership.

Three basic data types:

mappings - hash or dictionary,

sequences - array or list

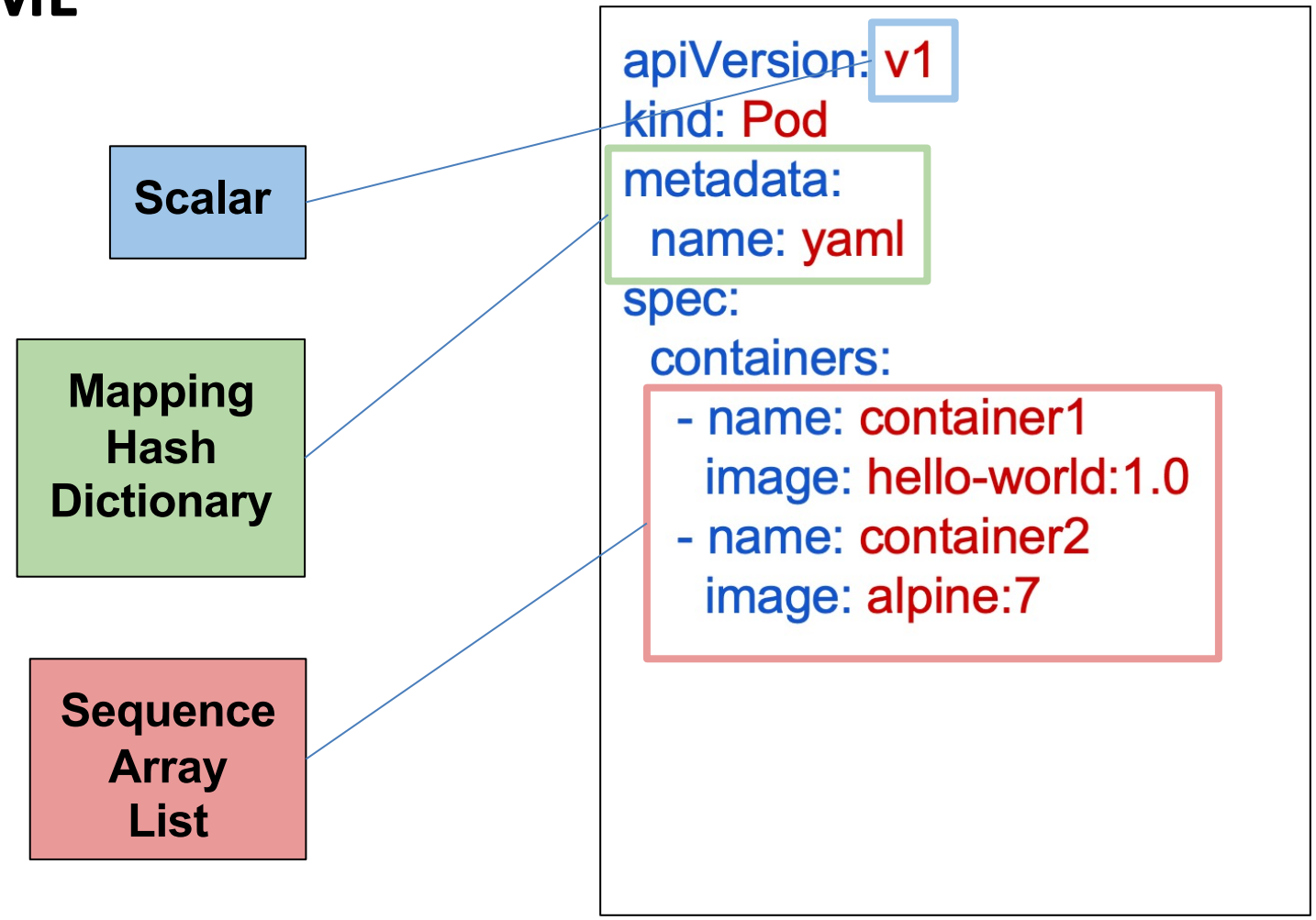
scalars - string, number, boolean etc

..THERE IS ALSO A COMMAND LINE INTERFACE (CLI)

OBJECT EXPRESSION - YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: yaml
spec:
  containers:
    - name: container1
      image: hello-world:1.0
    - name: container2
      image: alpine:7
```

OBJECT EXPRESSION - YAML



WORKING WITH KUBERNETES

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

JSON

```json
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name": "pod-example"
  },
  "spec": {
    "containers": [
      {
        "name": "nginx",
        "image": "nginx:stable-alpine",
        "ports": [ { "containerPort": 80 } ]
      }
    ]
  }
}
```

POD CONTAINER ATTRIBUTES

- `name` - The name of the container
- `image` - The container image
- `ports` - array of ports to expose. Can be granted a friendly name and protocol may be specified
- `env` - array of environment variables
- `command` – Entrypoint array (equiv to Docker `ENTRYPOINT`)
- `args` - Arguments to pass to the command (equiv to Docker `CMD`)

Container

```
name: nginx
image: nginx:stable-alpine
ports:
  - containerPort: 80
    name: http
    protocol: TCP
env:
  - name: MYVAR
    value: isAwesome
command: ["/bin/sh", "-c"]
args: ["echo ${MYVAR}"]
```

SERVICES

Unified method of accessing the exposed workloads of Pods.

Durable resource (unlike Pods)

- static cluster-unique IP
- static namespaced DNS name

```
<service name>.<namespace>.svc.cluster.local
```

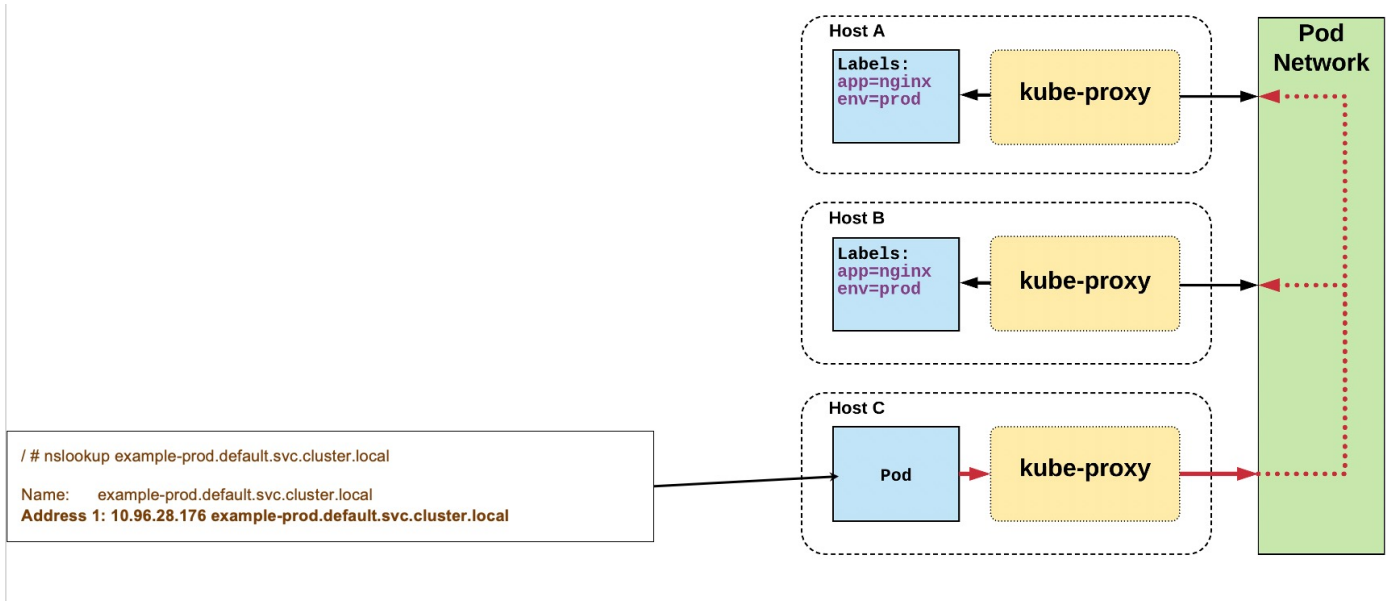
SERVICES

There are 3 major service types:

ClusterIP (default)
NodePort
LoadBalancer

SERVICES

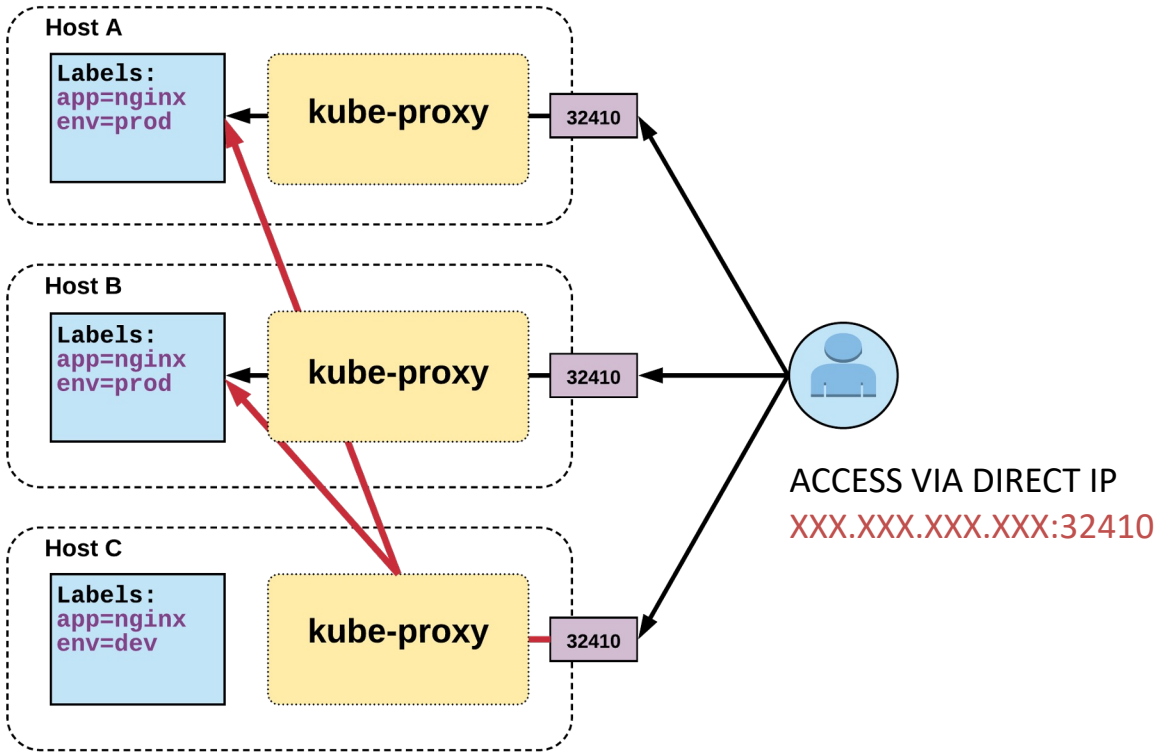
ClusterIP services exposes a service on a strictly cluster internal virtual IP.




```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

SERVICES

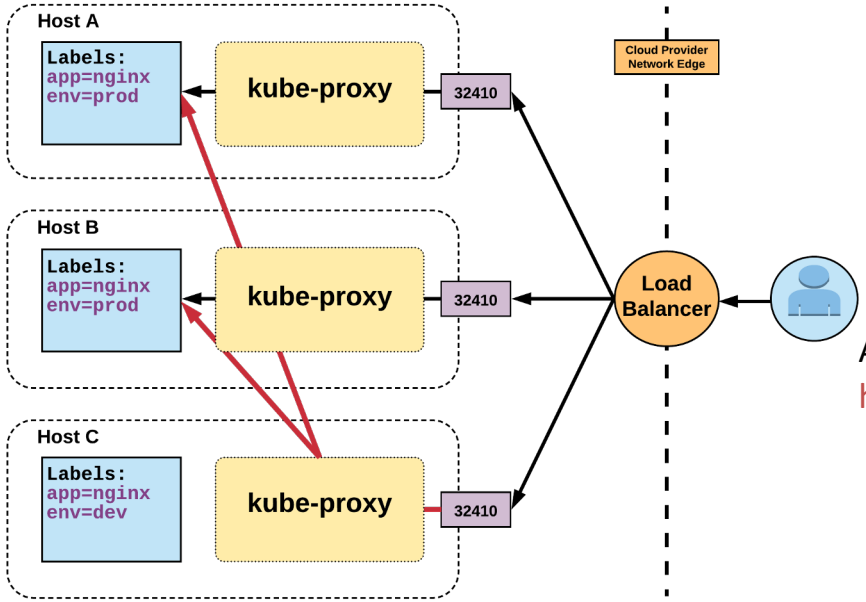
NodePort Exposes a port on every node's IP.




```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: NodePort
  selector:
    app: nginx
    env: prod
  ports:
  - nodePort: 32410
    protocol: TCP
    port: 80
    targetPort: 80
```

SERVICES

LoadBalancer Works in conjunction with an external system to map a cluster external IP to the exposed service.



ACCESS VIA URL
http://someURL.someDomain

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

SERVICES

ExternalName is used to reference endpoints **OUTSIDE** the cluster.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ExternalName
spec:
  externalName: example.com
```

DEPLOYMENT

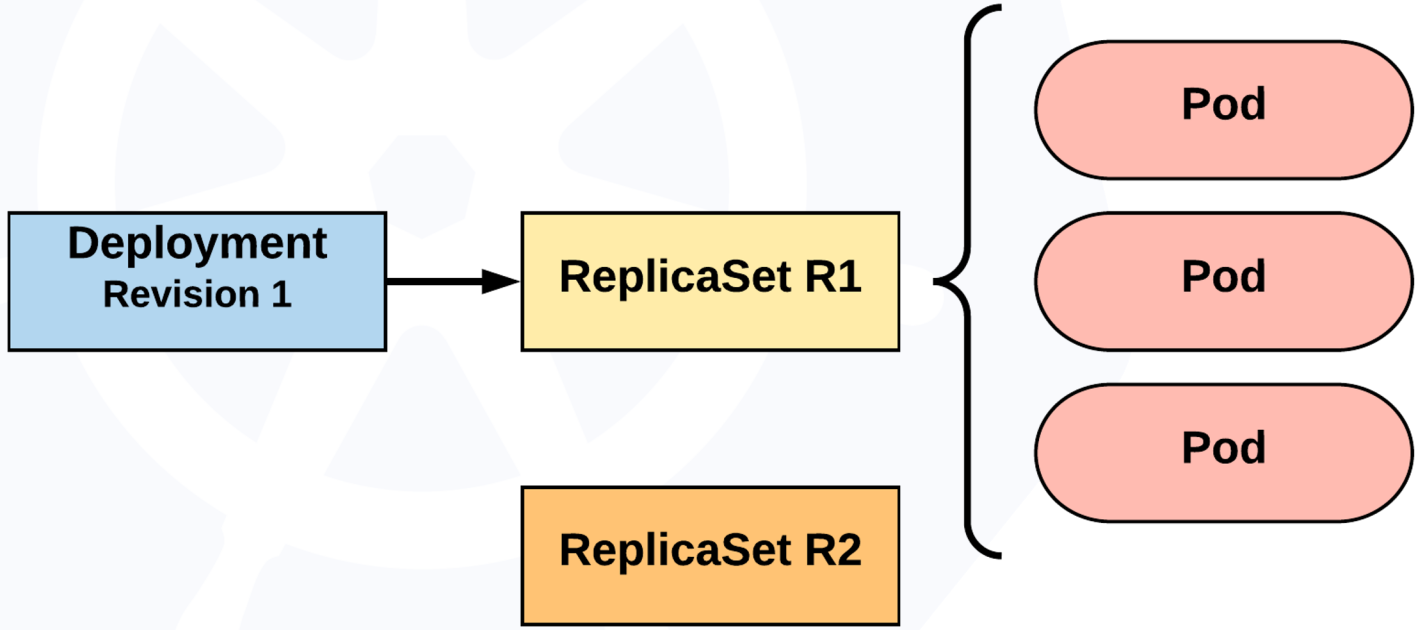
- `revisionHistoryLimit:` The number of previous iterations of the Deployment to retain.
- `strategy:` Describes the method of updating the Pods based on the `type`. Valid options are `Recreate` or `RollingUpdate`.
  - `Recreate:` All existing Pods are killed before the new ones are created.
  - `RollingUpdate:` Cycles through updating the Pods according to the parameters: `maxSurge` and `maxUnavailable`.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```

ROLLING UPDATE CONCEPT

Updating pod template generates a new **ReplicaSet** revision.

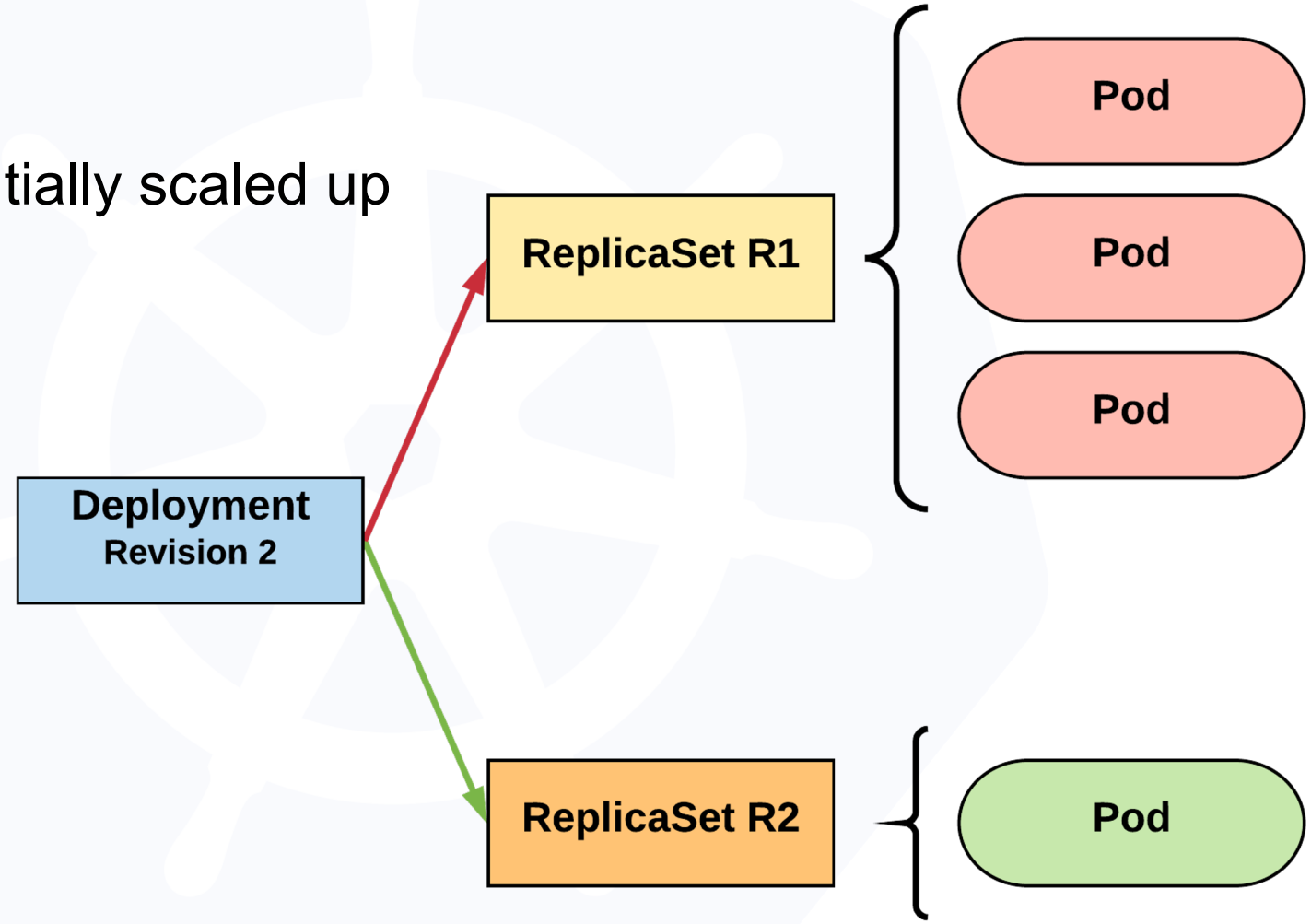
R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



ROLLING UPDATE CONCEPT

New **ReplicaSet** is initially scaled up based on `maxSurge`.

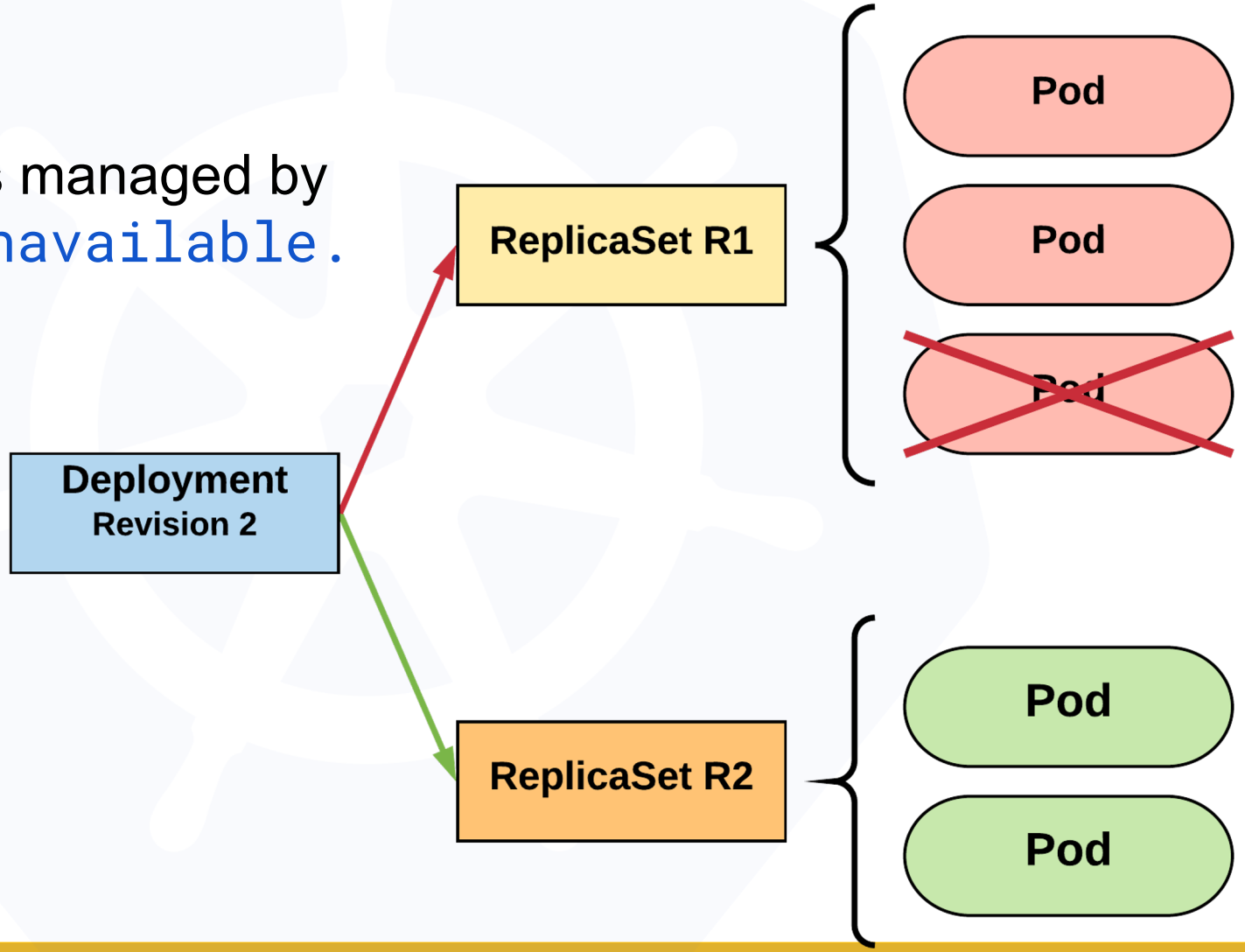
R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



ROLLING UPDATE CONCEPT

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

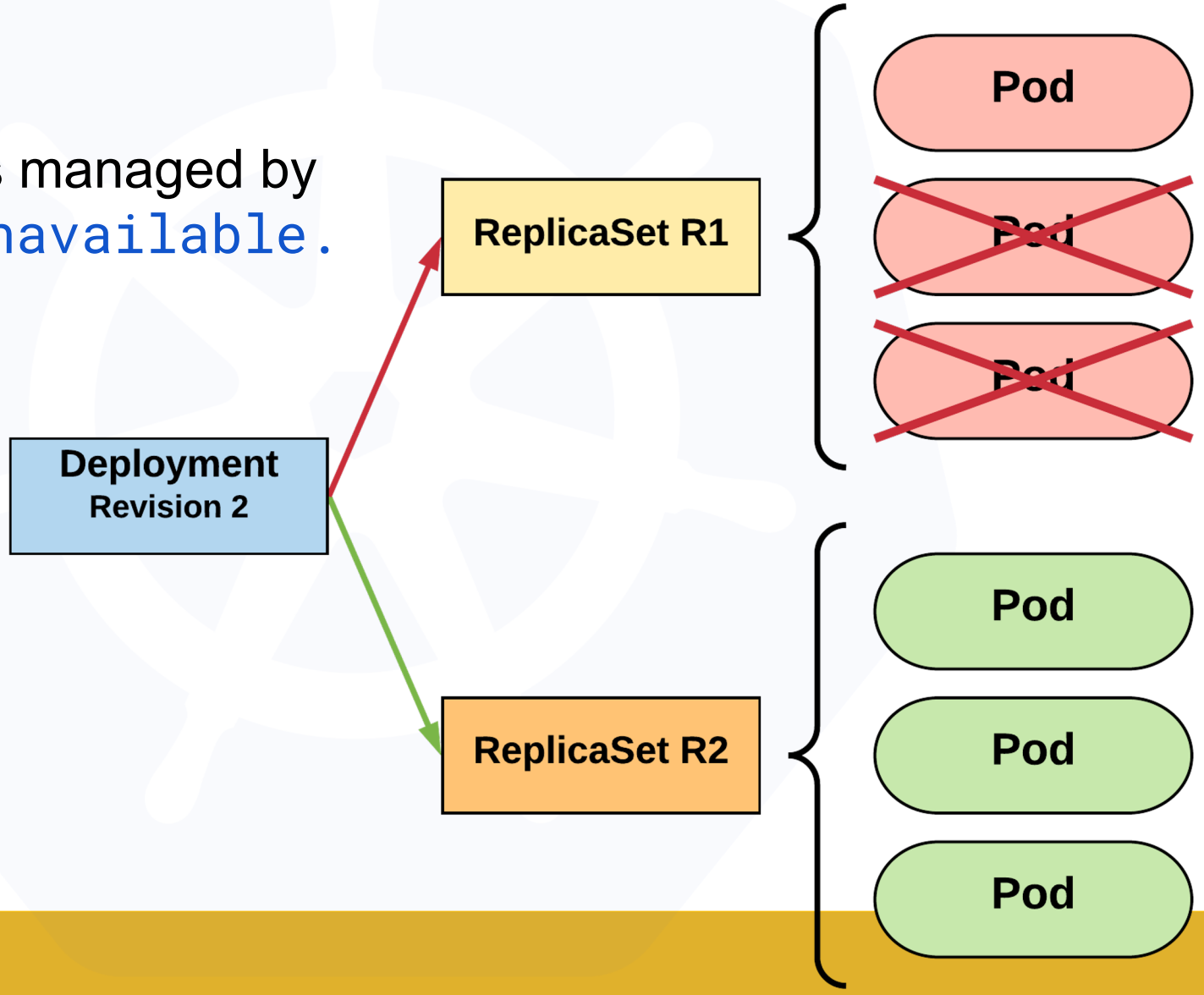
R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



ROLLING UPDATE CONCEPT

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

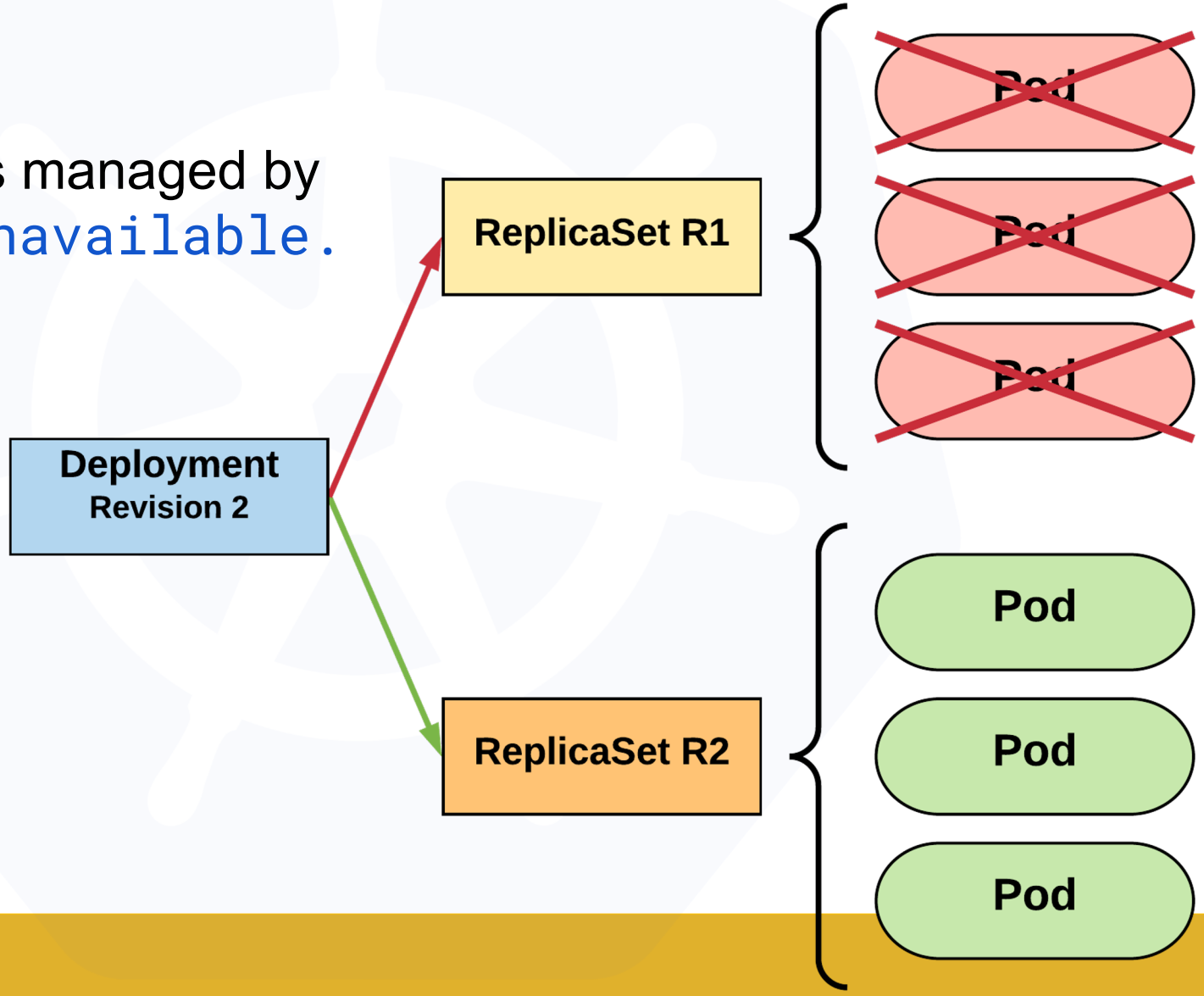
R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



ROLLING UPDATE CONCEPT

Phase out of old Pods managed by `maxSurge` and `maxUnavailable`.

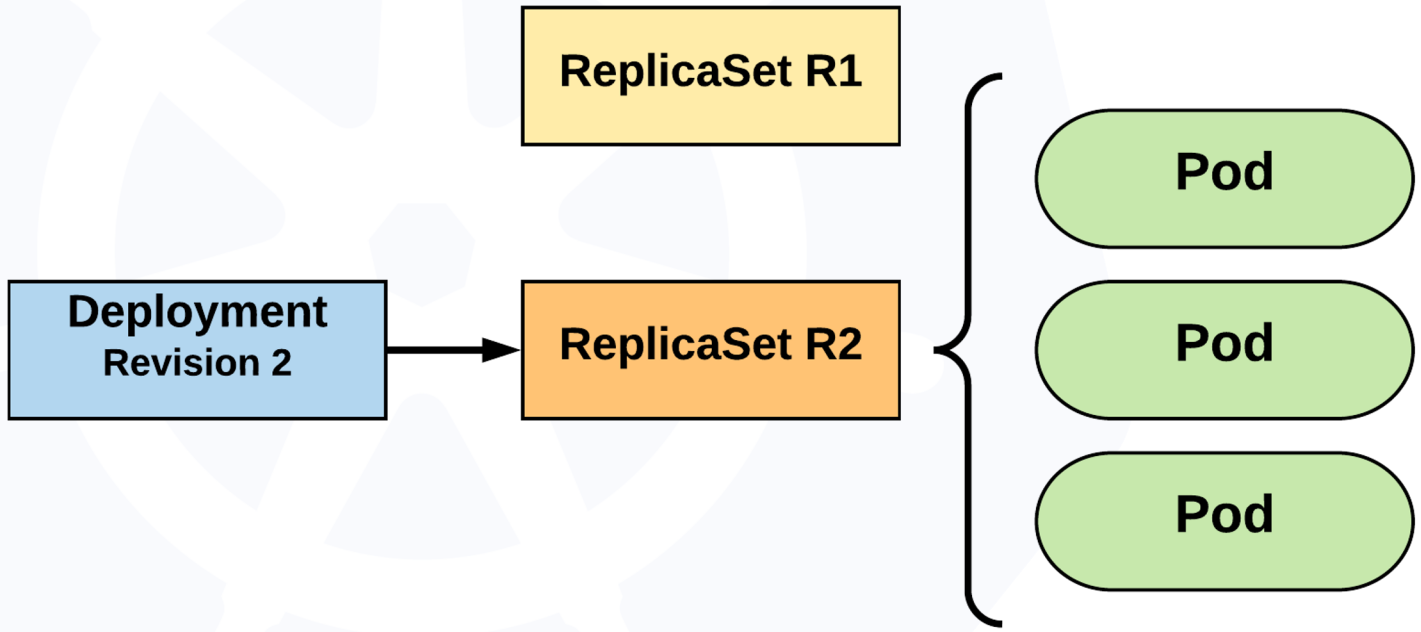
R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



ROLLING UPDATE CONCEPT

Updated to new deployment revision completed.

R1 pod-template-hash:
`676677fff`
R2 pod-template-hash:
`54f7ff7d6d`



CONFIGURATION AND SENSITIVE DATA

Kubernetes has an integrated pattern for decoupling configuration from application or container.

This pattern makes use of two Kubernetes components: **ConfigMaps** and **Secrets.**

There are instances in your application where you want to be able to store runtime-specific information.
Things like connection info, SSL certs, or parameters that will probably change over time.

CONFIGURATION AND SENSITIVE DATA

CONFIGMAP: Configmap data is stored “outside” of the worker node (in etcd)

Externalized data stored within kubernetes.

Can be referenced through several different means:

- environment variable
- a command line argument (via env var)
- injected as a file into a volume mount

Can be created from a manifest, literals, directories, or files directly.

CONFIGURATION AND SENSITIVE DATA

All produce a **ConfigMap** with the same content

YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-
example
data:
  city: Ann Arbor
  state: Michigan
```

CLI – from user input

```
$ kubectl create configmap literal-example \
> --from-literal="city=Ann Arbor" --from-literal=state=Michigan
configmap “literal-example” created
```

CLI – from file

```
$ cat info/city
Ann Arbor
$ cat info/state
Michigan
$ kubectl create configmap file-example --from-file=cm/city --
from-file=cm/state
configmap "file-example" created
```

CONFIGURATION AND SENSITIVE DATA

SECRET: Functionally identical to a ConfigMap.

Stored as base64 encoded content.

Encrypted at rest within etcd (if configured).

Ideal for username/passwords, certificates or other sensitive information that should not be stored in a container.

Can be created from a manifest, literals, directories, or from files directly.

CONFIGURATION AND SENSITIVE DATA

All produce a **Secret** with the same content

YAML (need to insert in base64)

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-example
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password:
bXlwYXNzd29yZA==
```

CLI – from user input

```
$ kubectl create secret generic literal-secret \
> --from-literal=username=example \
> --from-literal=password=mypassword
secret "literal-secret" created
```

CLI – from file

```
$ cat secret/username
example
$ cat secret/password
mypassword
$ kubectl create secret generic file-secret --from-file=secret/username --
from-file=secret/password
Secret "file-secret" created
```

CONFIGURATION AND SENSITIVE DATA

Example of how to invoke

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: cm-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv CITY"]
        env:
        - name: CITY
          valueFrom:
            configMapKeyRef:
              name: manifest-example
              key: city
      restartPolicy: Never
```

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: secret-env-example
spec:
  template:
    spec:
      containers:
      - name: mypod
        image: alpine:latest
        command: ["/bin/sh", "-c"]
        args: ["printenv USERNAME"]
        env:
        - name: USERNAME
          valueFrom:
            secretKeyRef:
              name: manifest-example
              key: username
      restartPolicy: Never
```

RESOURCES

Kubernetes tutorial
https://kubernetes.io/docs/tutorials/kubernetes-basics/

Introduction to container orchestration
https://www.exoscale.ch/syslog/2016/07/26/container-orch/

TNS Research: The Present State of Container Orchestration
https://thenewstack.io/tns-research-present-state-container-orchestration/

Large-scale cluster management at Google with Borg
https://research.google.com/pubs/pub43438.html

Free Kubernetes Courses
https://www.edx.org/

Interactive Kubernetes Tutorials
https://www.katacoda.com/courses/kubernetes

Learn Kubernetes the Hard Way
https://github.com/kelseyhightower/kubernetes-the-hard-way

Official Kubernetes Youtube Channel
https://www.youtube.com/c/KubernetesCommunity

Official CNCF Youtube Channel
https://www.youtube.com/c/cloudnativefdn

Awesome Kubernetes
https://ramitsurana.gitbooks.io/awesome-kubernetes/content/

Q&A

Thanks

for your attention!

Questions?

Federico Accetta federico_accetta@it.ibm.com