



# kubernetes

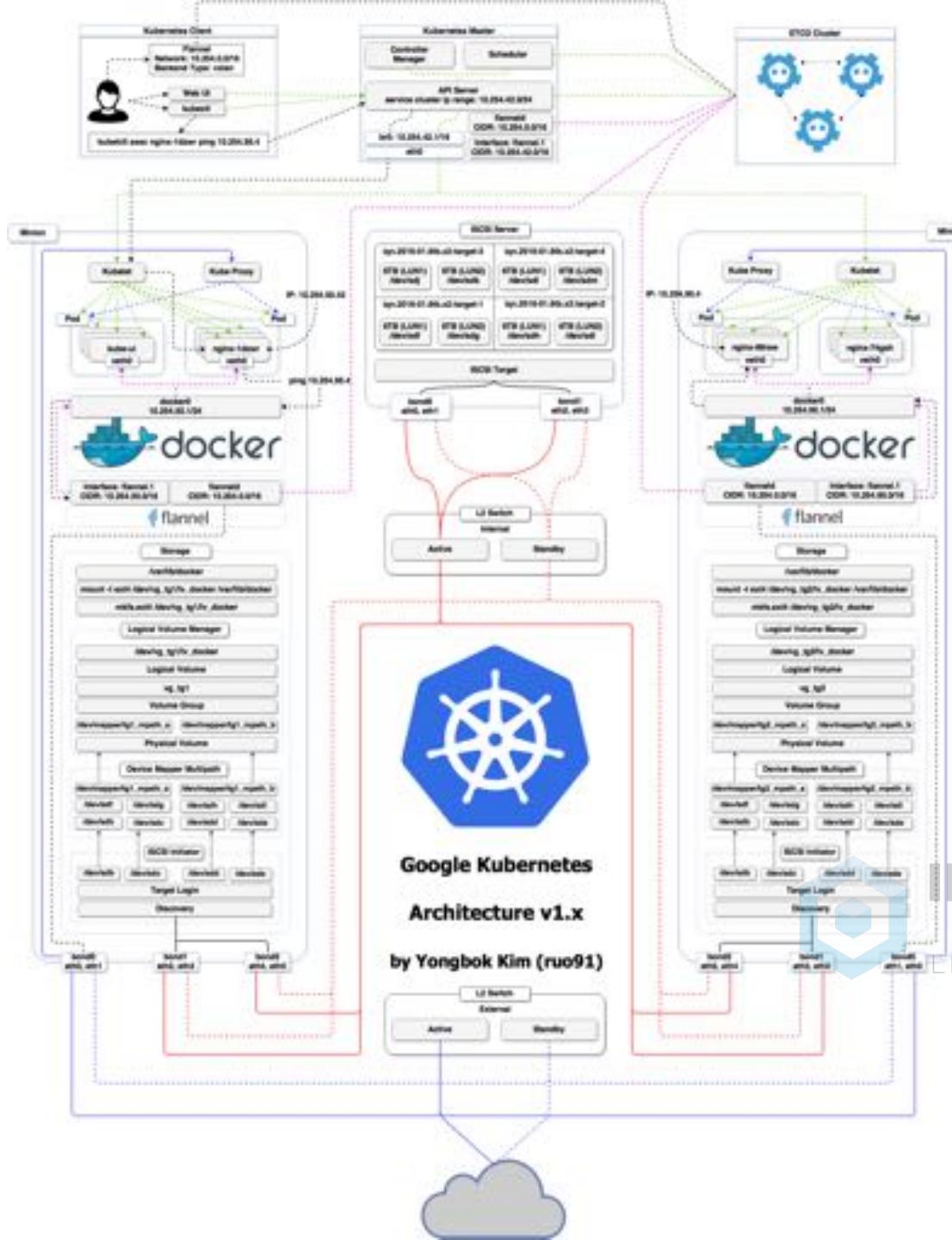
# Course Objectives

By the end of the course you will be able to:

- Describe Kubernetes architecture and components
  - API
  - Scheduler
  - Kubelet
- Deploy PODs in a resilient and high-performance configuration
- Deploy, configure, monitor, and diagnose Kubernetes services.
- Understand and implement custom scheduler(s)
- Use ConfigMap to store values for Kubernetes and Applications
- Use Secrets to securely store sensitive data
- Orchestrate startup of applications using init-containers

# Kubernetes Architecture



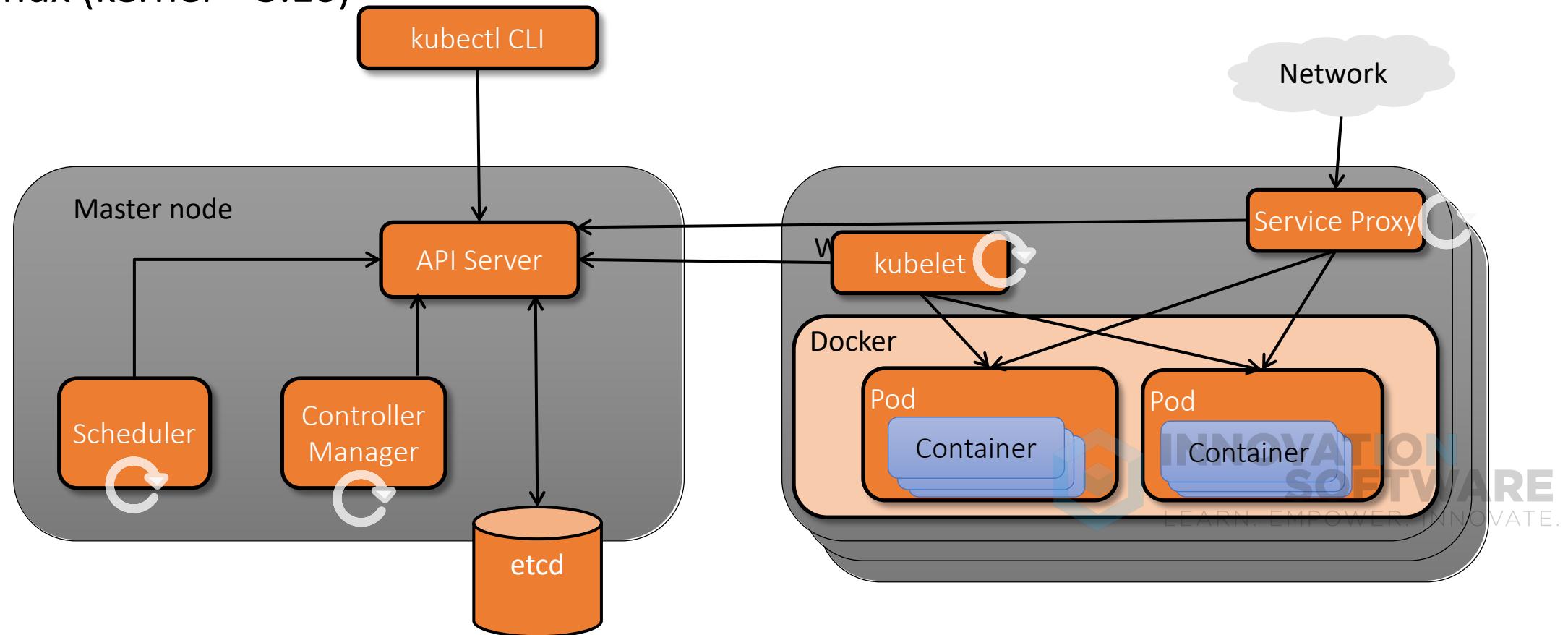


# INNOVATION SOFTWARE

LEARN. EMPOWER. INNOVATE.

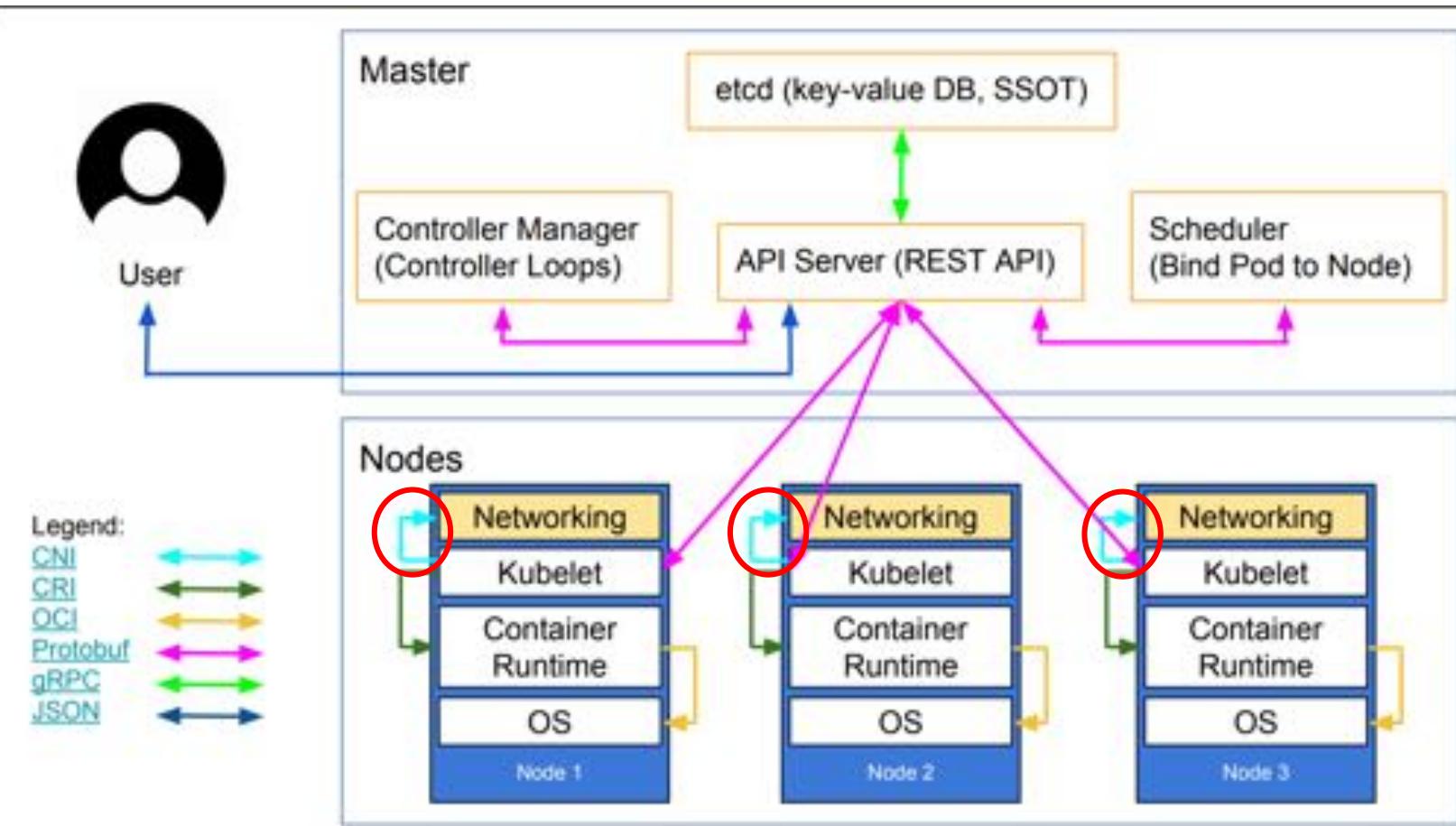
# Kubernetes Cluster Architecture

- Kubernetes nodes can be physical hosts or VM's running a container-friendly Linux (kernel > 3.10)

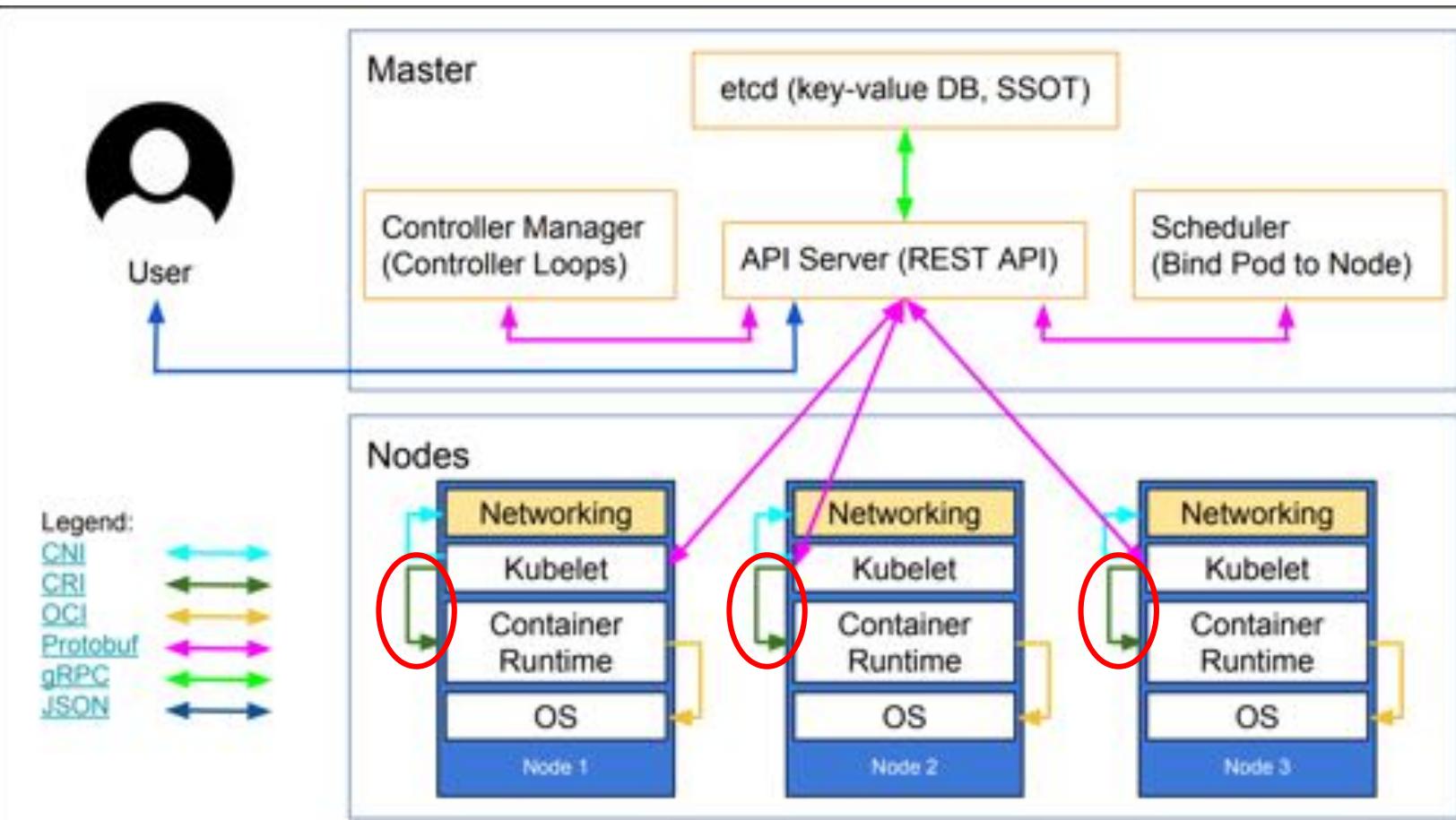


# Components Communication

- CNI = Container Network Interface

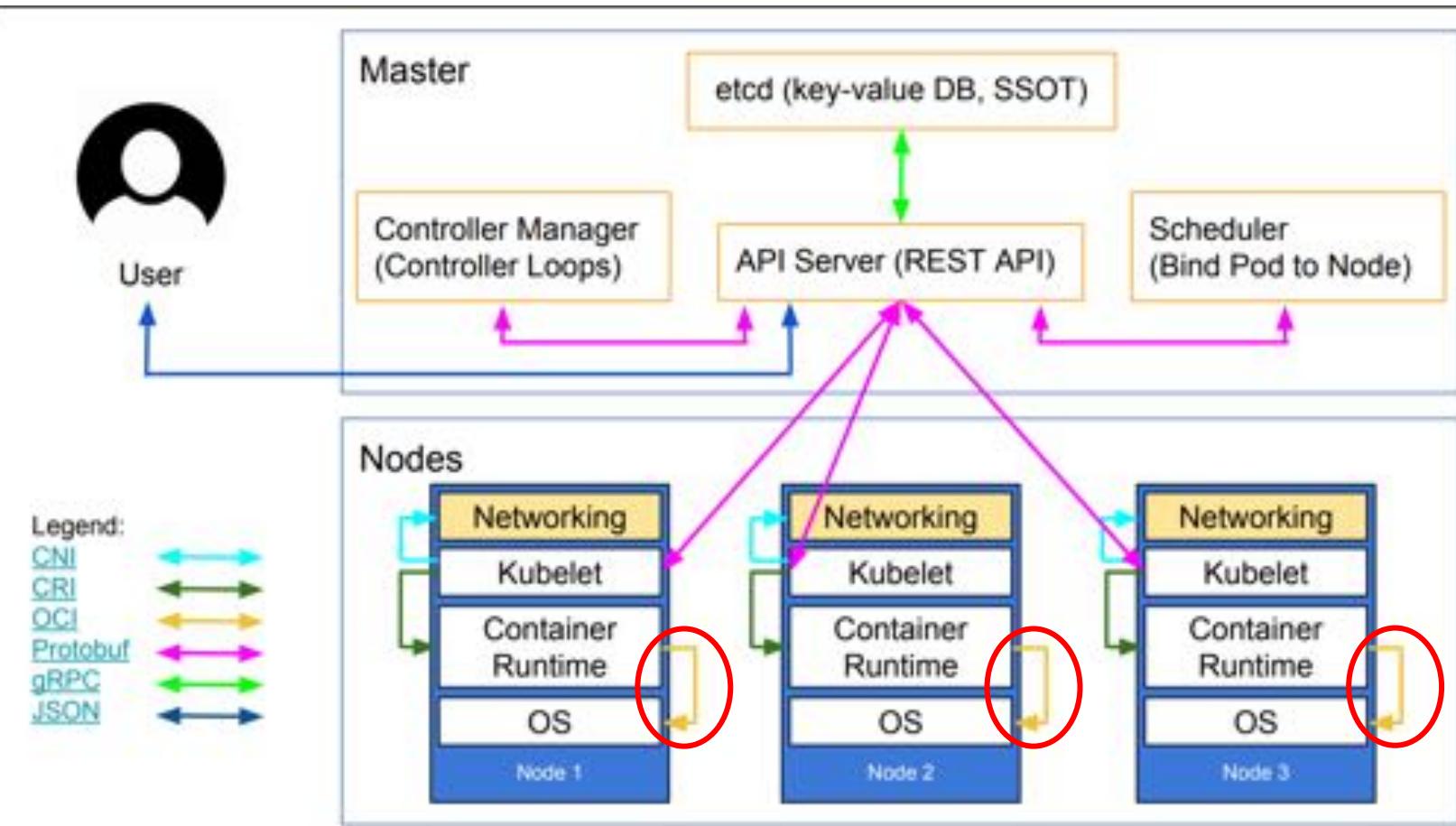


# Components Communication



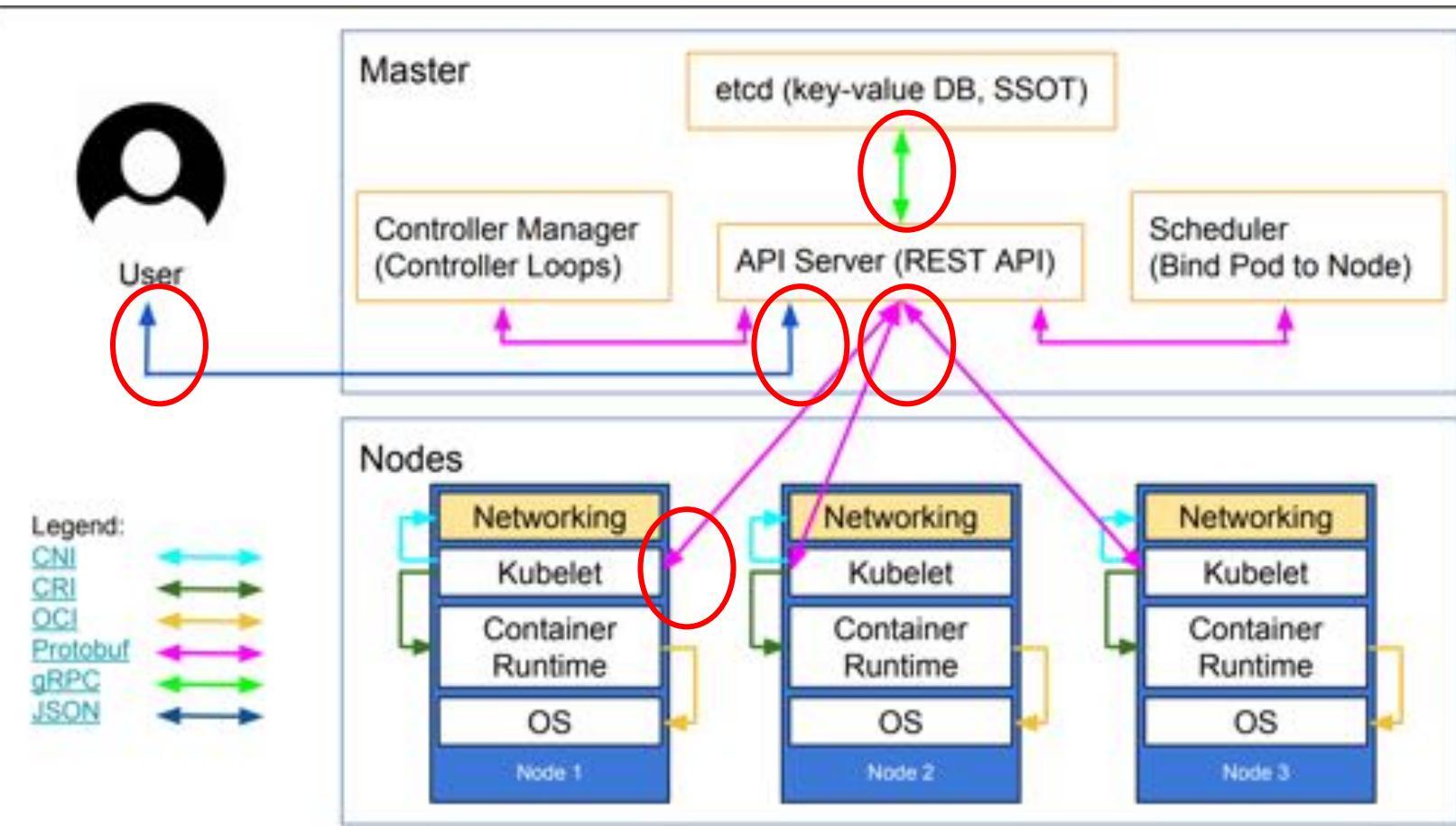
- CNI = Container Network Interface
- CRI = Container Runtime Interface
  - Docker, CRI-O, Containerd

# Components Communication



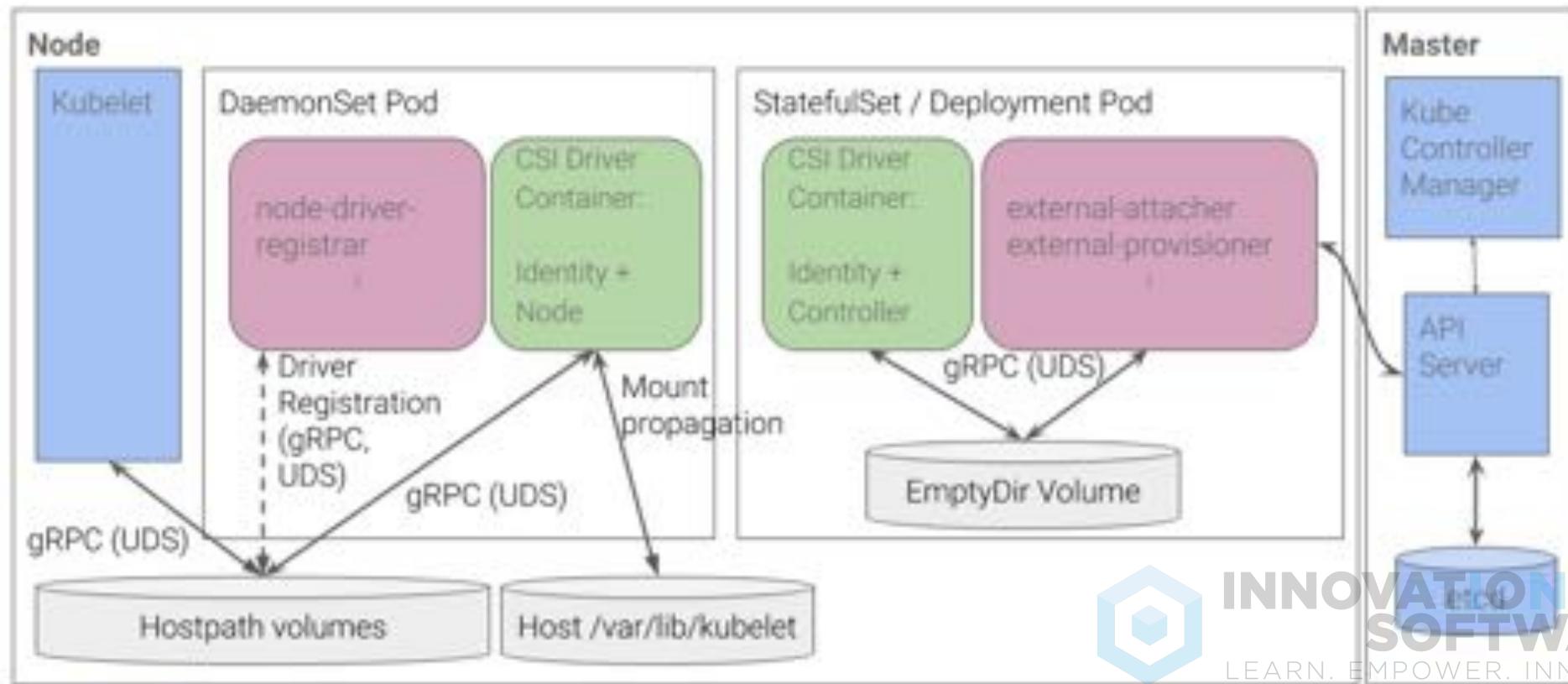
- CNI = Container Network Interface
- CRI = Container Runtime Interface
  - Docker, CRI-O, Containerd
- OCI = Open Container Initiative

# Components Communication



- CNI = Container Network Interface
- CRI = Container Runtime Interface
  - Docker, CRI-O, Containerd
- OCI = Open Container Initiative
- Protobuf
- gRPC
- JSON

# Container Storage Interface



Third Party Storage Vendor Container



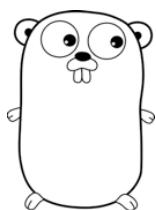
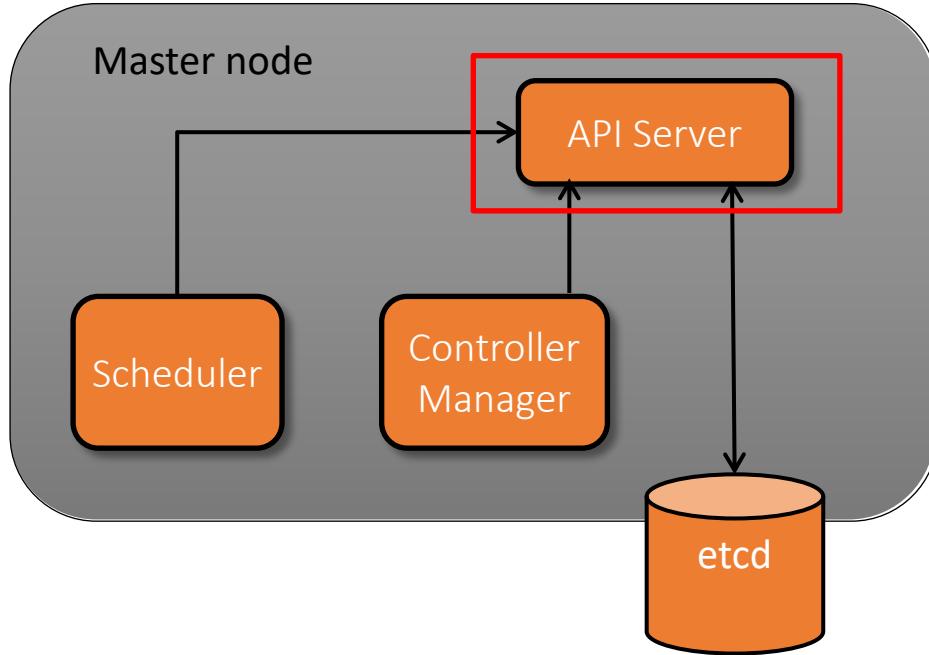
Sidecar containers by Kubernetes Team



**INNOVATION**  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

UDS - Unix Domain Socket

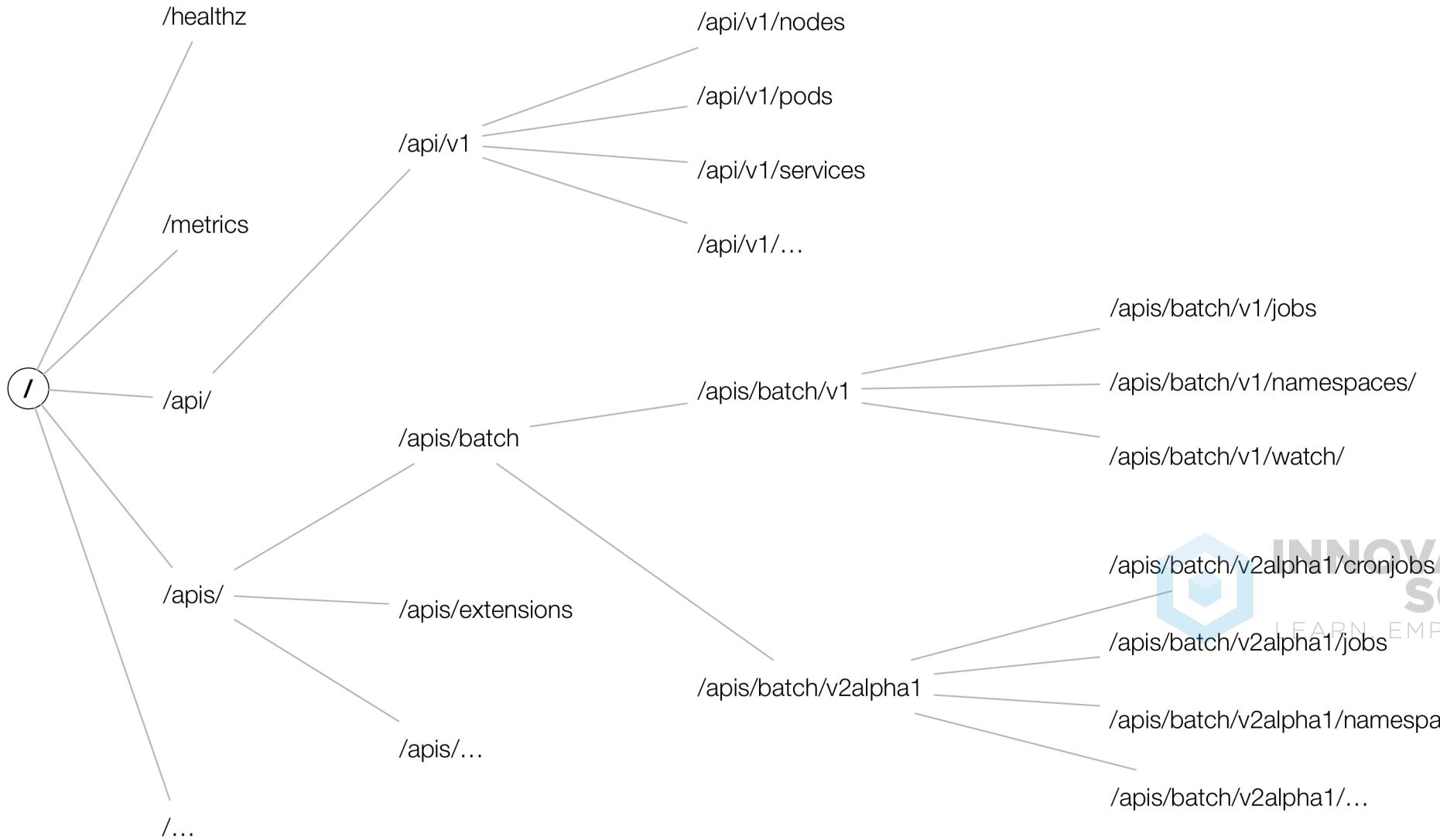
# Kubernetes Master Node Components



K8s components  
written in Go  
([golang.org](https://golang.org))

- **API Server (`kube-apiserver`)**: exposes the Kubernetes REST API, and can be scaled horizontally
- **Scheduler (`kube-scheduler`)**: selects nodes for newly created pods to run on
- **Controller manager (`kube-controller-manager`)**: runs background controller processes for the system to enforce declared object states, e.g. Node Controller, Replication Controller, ...
- **Persistent data store (`etcd`)**: all K8s system data is stored in a distributed, reliable key-value store. `etcd` may run on separate nodes from the master

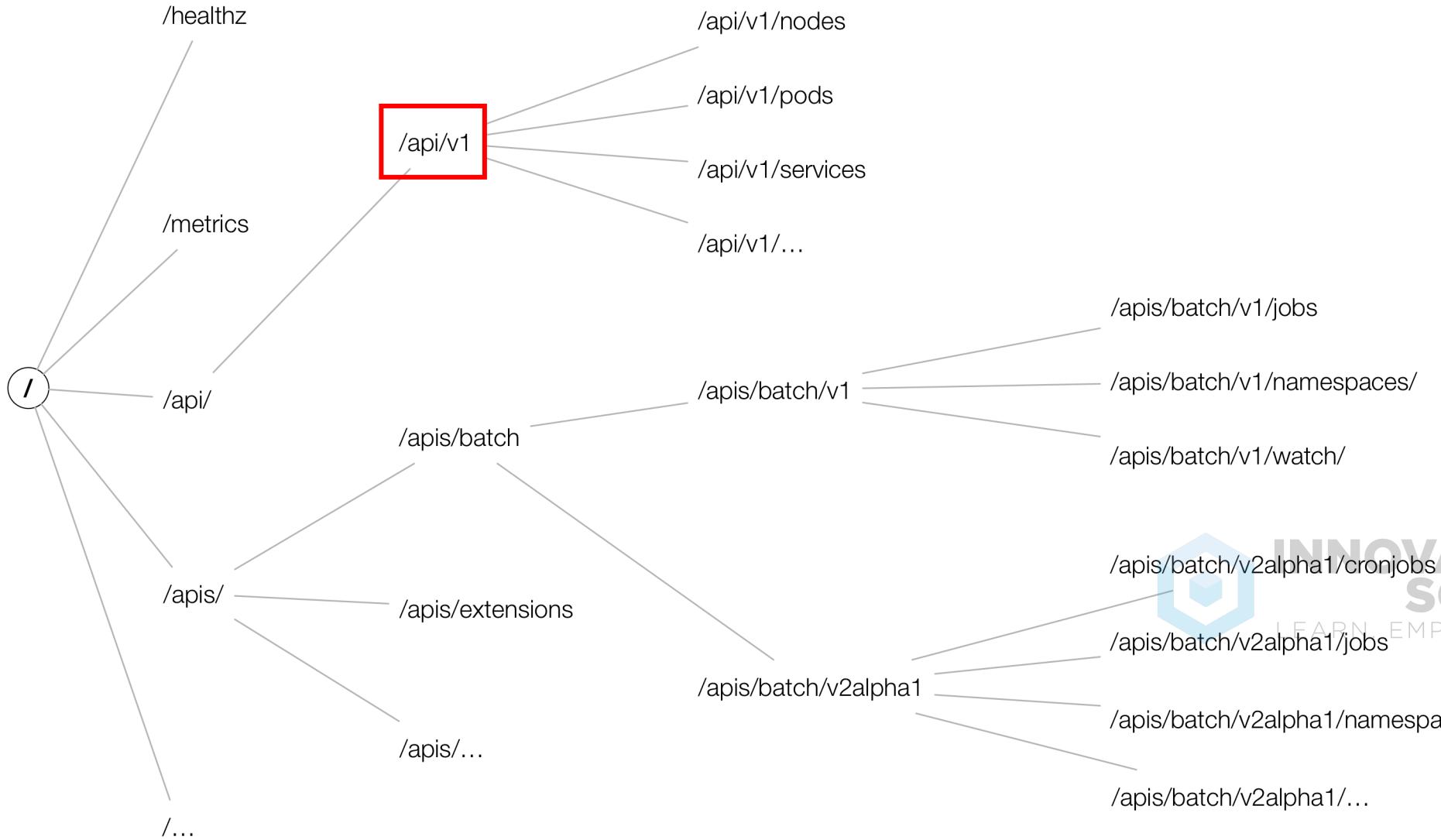
# API Deep Dive



**INNOVATION  
SOFTWARE**

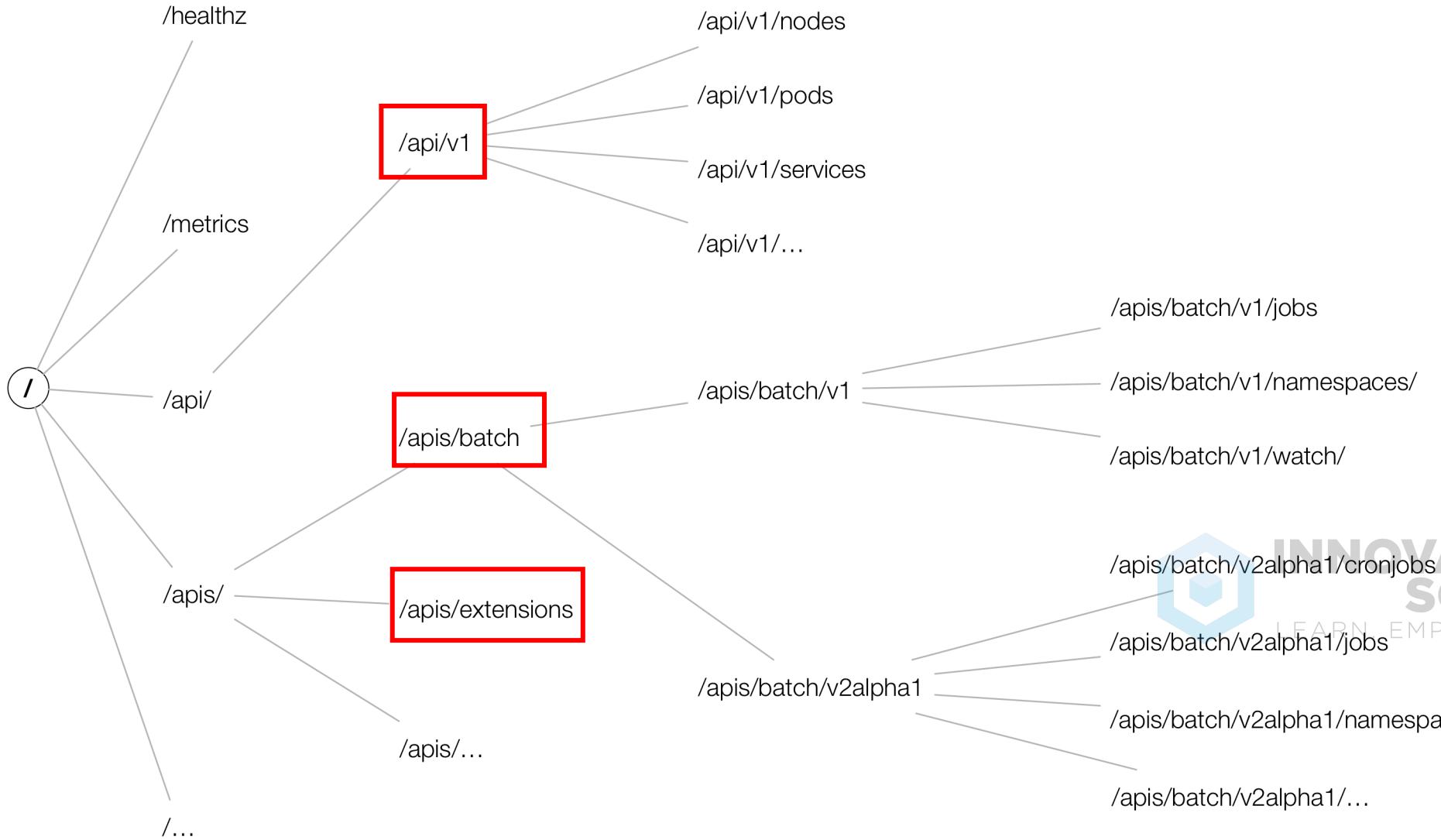
LEARN. EMPOWER. INNOVATE.

# API Groups



LEARN. EMPOWER. INNOVATE.

# API Groups

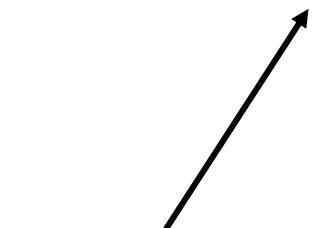


**INNOVATION SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# API Group

- Collection of Kinds that are logically related
  - Job, ScheduledJob in batch API Group

/apis/**batch**/v1/jobs



Group

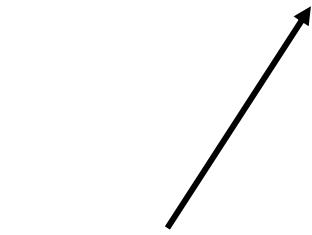


**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

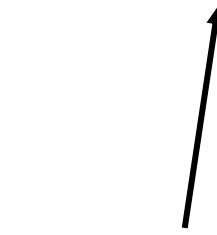
# API Version

- Each API Group can be part of multiple versions
  - v1alpha1 -> v1beta1 -> v1

/apis/**batch**/**v1**/jobs



Group



Version

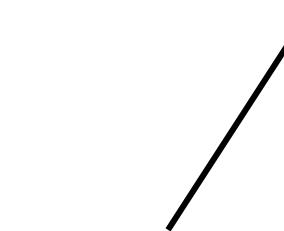


**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

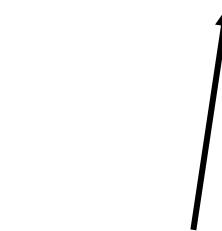
# API Resource

- System entity being manipulated as JSON over HTTP

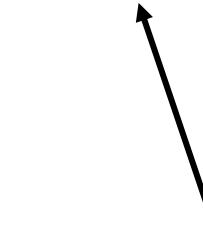
/apis/**batch**/v1/jobs



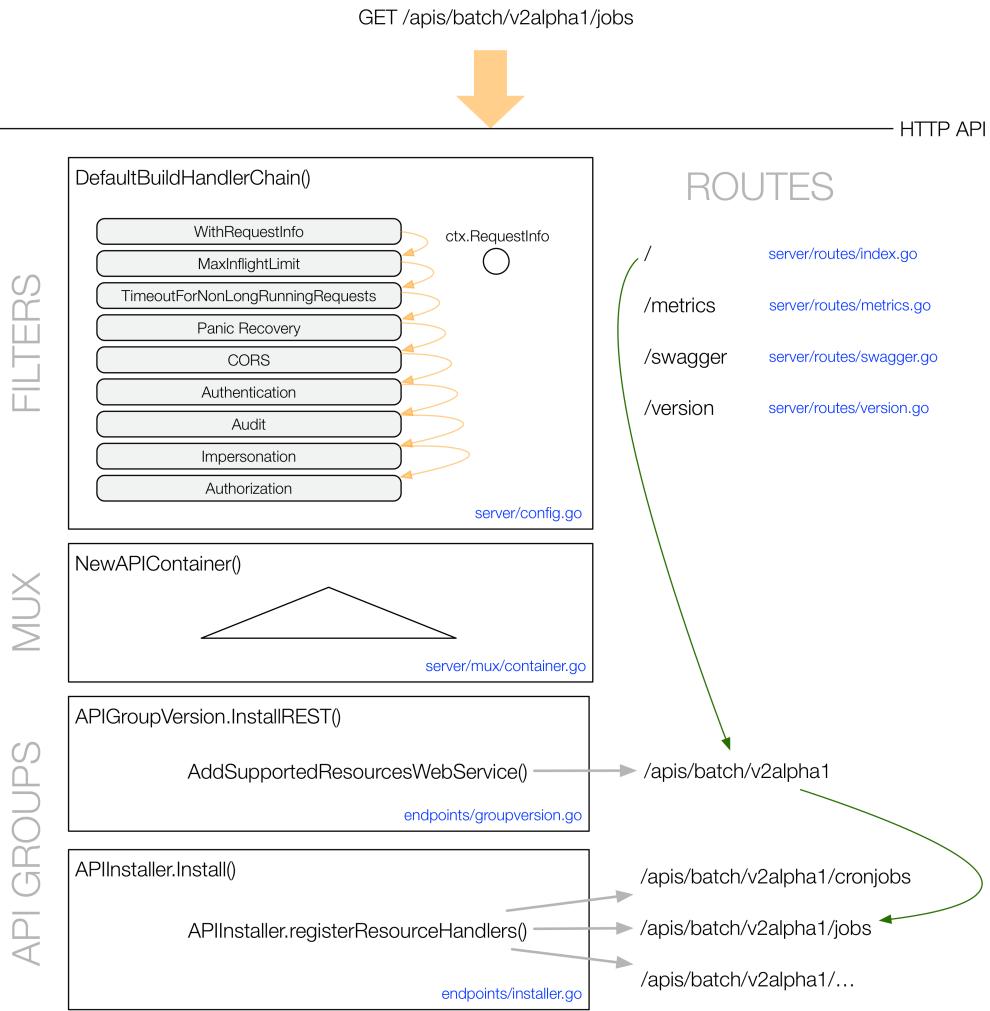
Group



Version



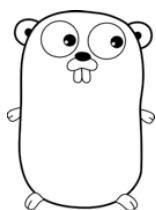
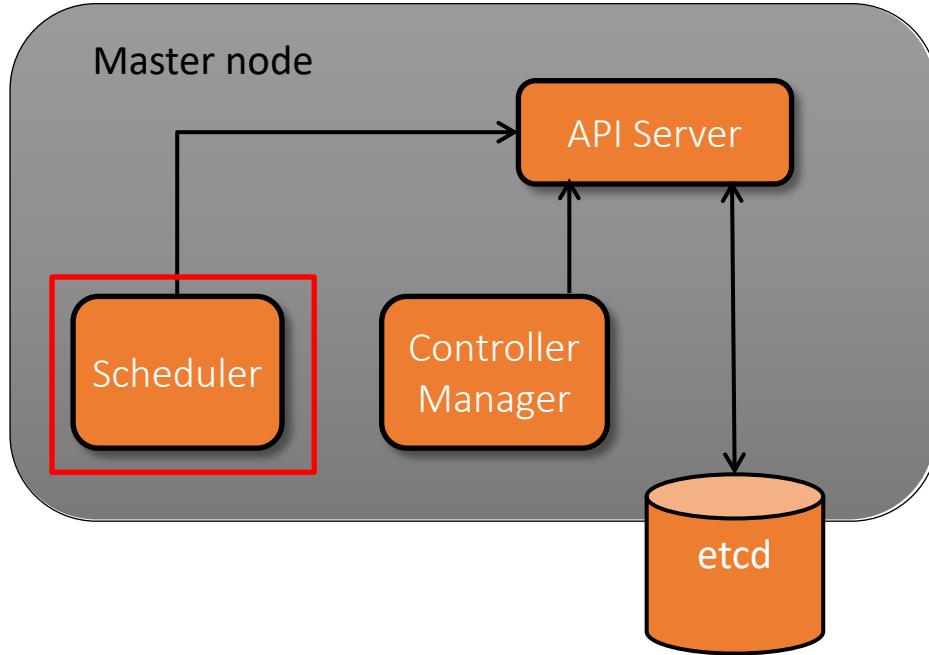
# Request Flow and Processing



1. HTTP request is processed
2. Multiplexer routes the HTTP request to handler depending on path
3. Routes connect handlers with HTTP paths
4. The handler, registered per API Group takes the HTTP request context (user, rights etc.) delivers the requested object from storage (etcd)



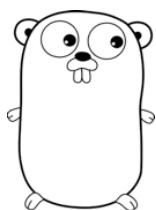
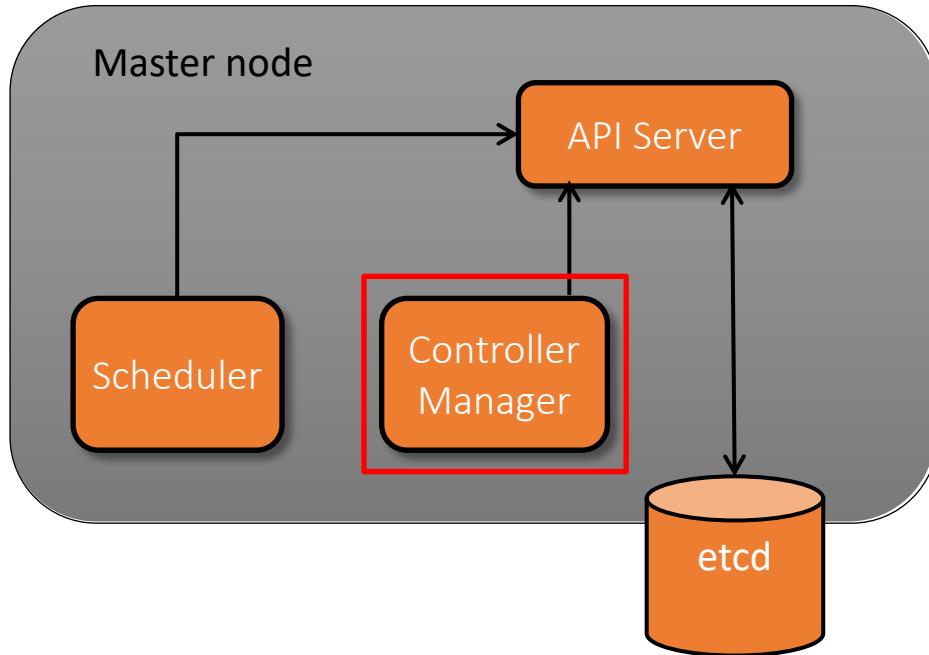
# Kubernetes Master Node Components



K8s components  
written in Go  
([golang.org](https://golang.org))

- **API Server (`kube-apiserver`)**: exposes the Kubernetes REST API, and can be scaled horizontally
- **Scheduler (`kube-scheduler`)**: selects nodes for newly created pods to run on
- **Controller manager (`kube-controller-manager`)**: runs background controller processes for the system to enforce declared object states, e.g. Node Controller, Replication Controller, ...
- **Persistent data store (`etcd`)**: all K8s system data is stored in a distributed, reliable key-value store. `etcd` may run on separate nodes from the master

# Kubernetes Master Node Components



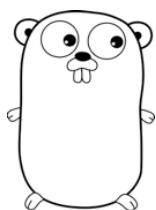
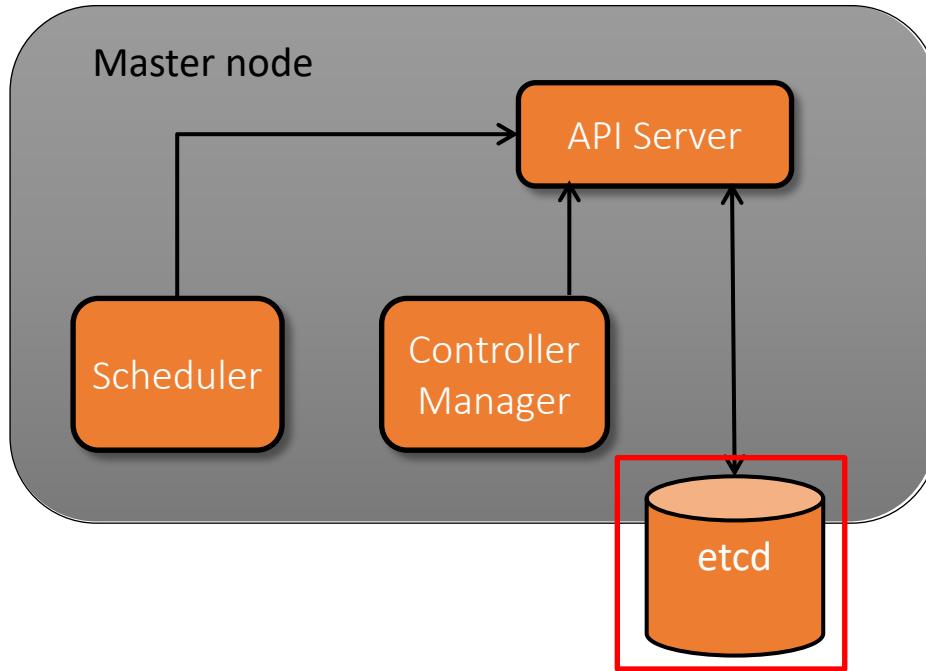
K8s components  
written in Go  
([golang.org](https://golang.org))

- **API Server (`kube-apiserver`)**: exposes the Kubernetes REST API, and can be scaled horizontally
- **Scheduler (`kube-scheduler`)**: selects nodes for newly created pods to run on
- **Controller manager (`kube-controller-manager`)**: runs background controller processes for the system to enforce declared object states, e.g. Node Controller, Replication Controller, ...
- **Persistent data store (`etcd`)**: all K8s system data is stored in a distributed, reliable key-value store. `etcd` may run on separate nodes from the master



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

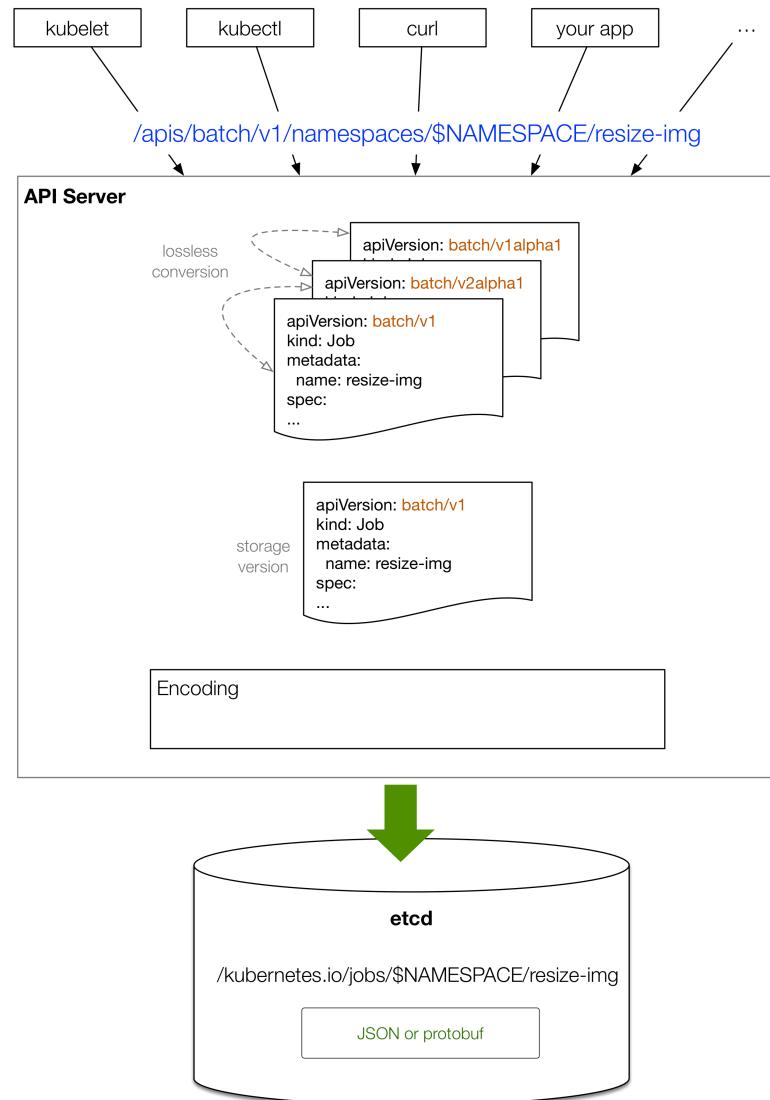
# Kubernetes Master Node Components



K8s components  
written in Go  
([golang.org](https://golang.org))

- **API Server (`kube-apiserver`)**: exposes the Kubernetes REST API, and can be scaled horizontally
- **Scheduler (`kube-scheduler`)**: selects nodes for newly created pods to run on
- **Controller manager (`kube-controller-manager`)**: runs background controller processes for the system to enforce declared object states, e.g. Node Controller, Replication Controller, ...
- **Persistent data store (`etcd`)**: all K8s system data is stored in a distributed, reliable key-value store. etcd may run on separate nodes from the master

# Etcd flow

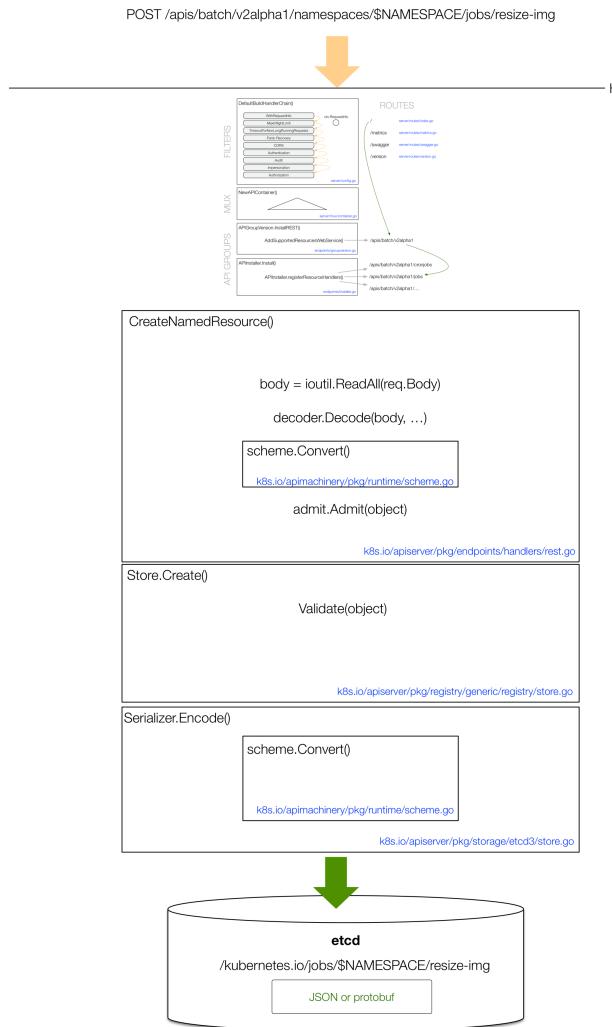


- **Etcd flow:**

- client provides desired object state in YAML or JSON
- kubectl converts YAML to JSON and sends it to API
- API server turns input object state into canonical storage version
- storage process in etcd, at a certain key, into a value with the encoding to JSON or protobuf.



# Serialization of State Flow

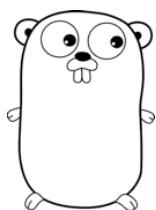
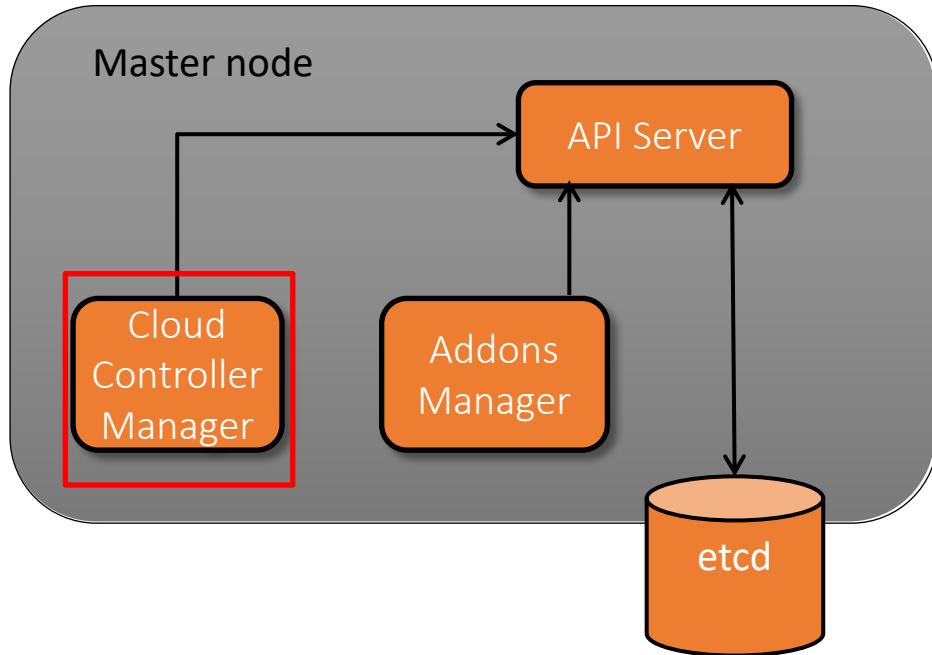


- **Serialization flow:**

- API Server keeps all known k8s object kinds in a Go registry (Scheme).
- Version of kinds defined along with how they can be:
  - converted
  - new objects created
  - objects encoded/decoded to JSON/protobuf



# Master Node Additional Components



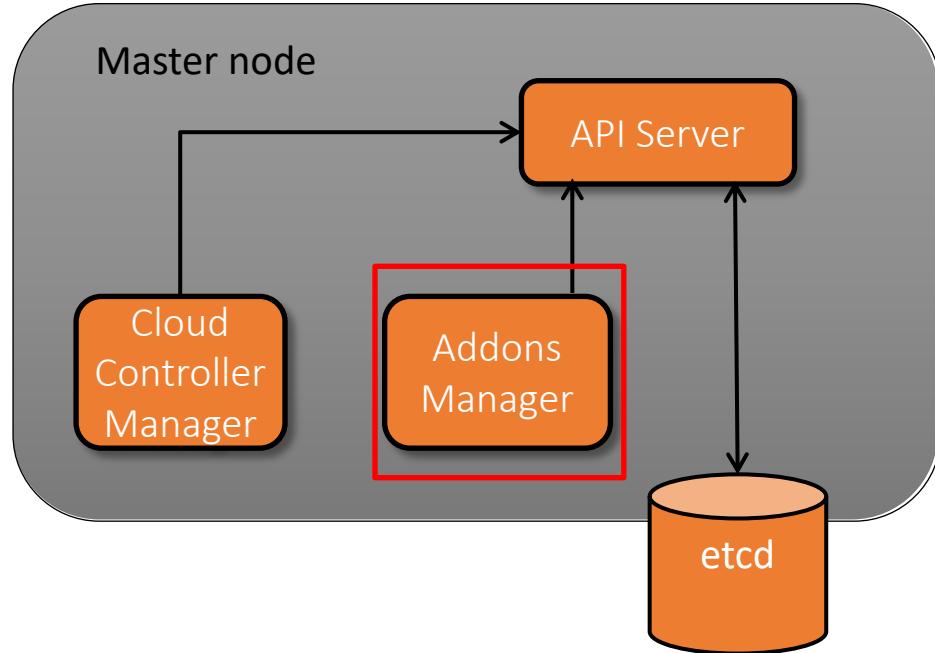
K8s components  
written in Go  
([golang.org](https://golang.org))

- **cloud-controller-manager:** runs controllers interacting with underlying IaaS providers – alpha feature
  - Allows cloud vendor-specific code to be separate from main K8s system components
- **addons-manager:** creates and maintains cluster addon resources in 'kube-system' namespace, e.g.
  - Kubernetes Dashboard:** general web UI for application and cluster management
  - kube-dns:** serves DNS records for K8s services and resources
  - Container resource monitoring and cluster-level logging



INNOVATION  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Master Node Additional Components

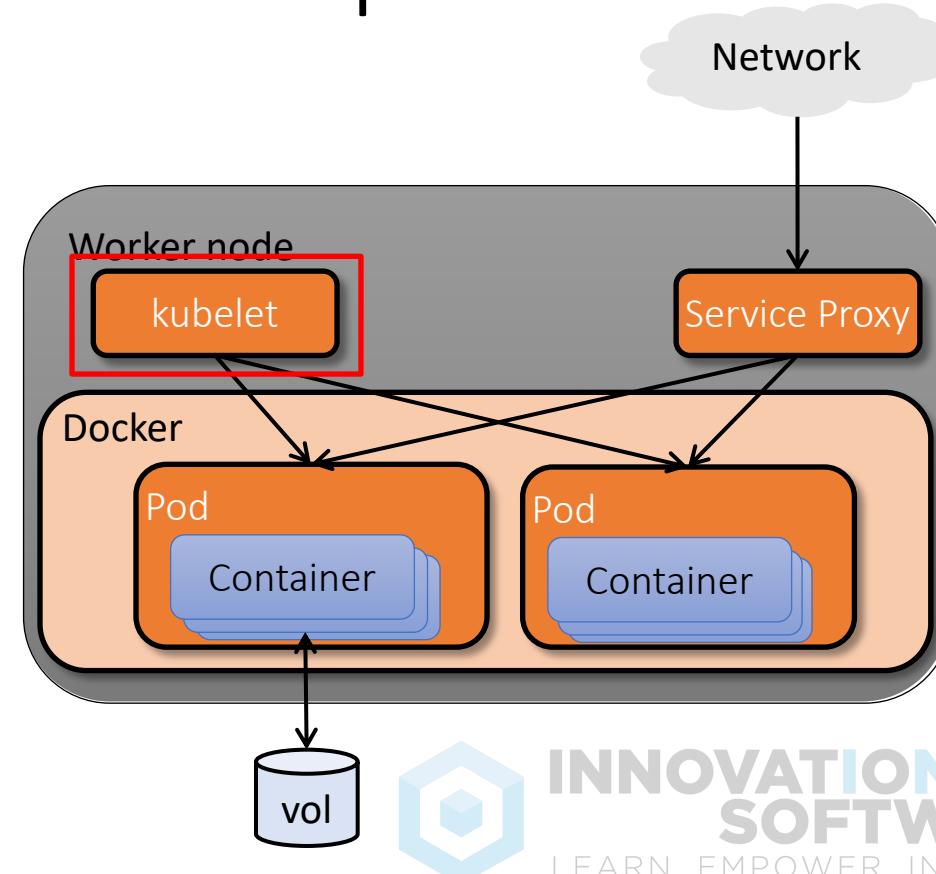


K8s components  
written in Go  
([golang.org](https://golang.org))

- **cloud-controller-manager**: runs controllers interacting with underlying IaaS providers – alpha feature
  - Allows cloud vendor-specific code to be separate from main K8s system components
- **addons-manager**: creates and maintains cluster addon resources in 'kube-system' namespace, e.g.
  - Kubernetes Dashboard**: general web UI for application and cluster management
  - kube-dns**: serves DNS records for K8s services and resources
  - Container resource monitoring and cluster-level logging

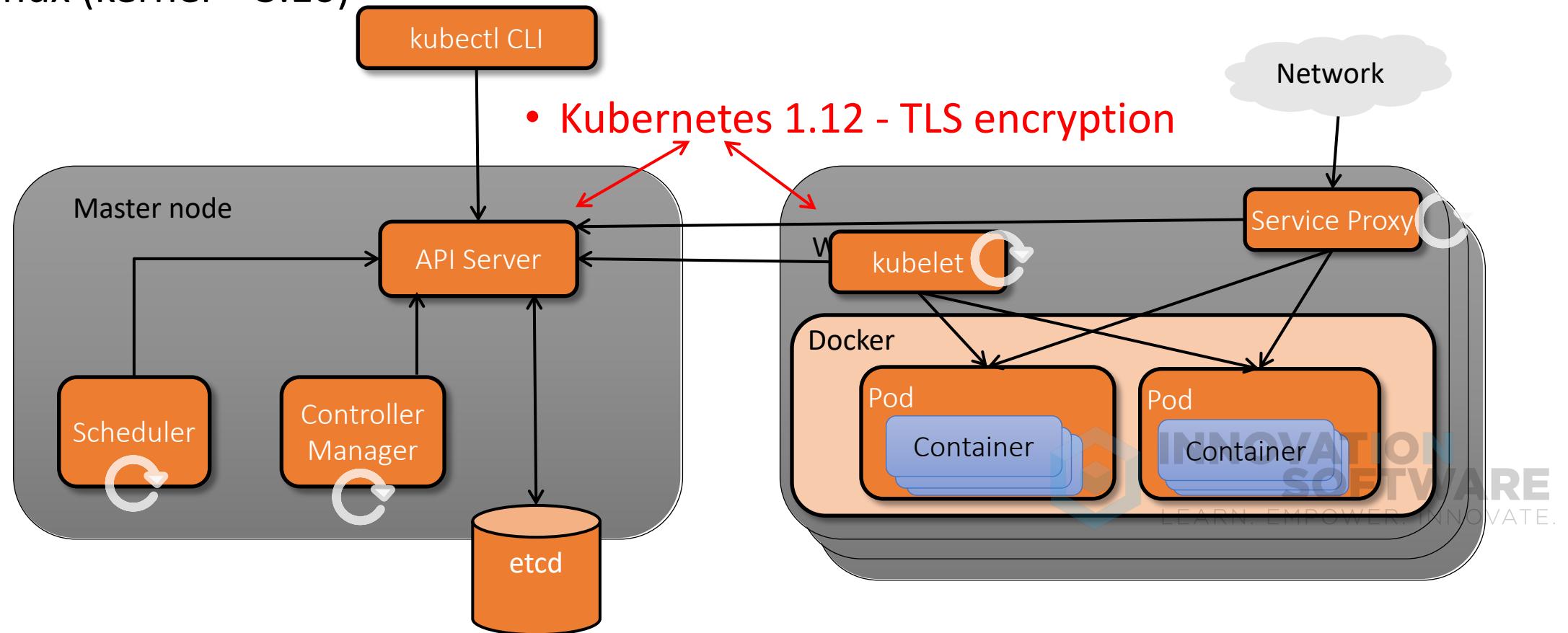
# Kubernetes Worker Node Components

- **kubelet**: local K8s agent that is responsible for operations on the node, including
  - Watching for pod assignments
  - Mounting pod required volumes
  - Running a pod's containers
  - Executing container liveness probes
  - Reporting pod status to system
  - Reporting node status to system
- **Service proxy (kube-proxy)**: enables K8s service abstractions by maintaining host network rules and forwarding connections
- **Docker**: runs the containers



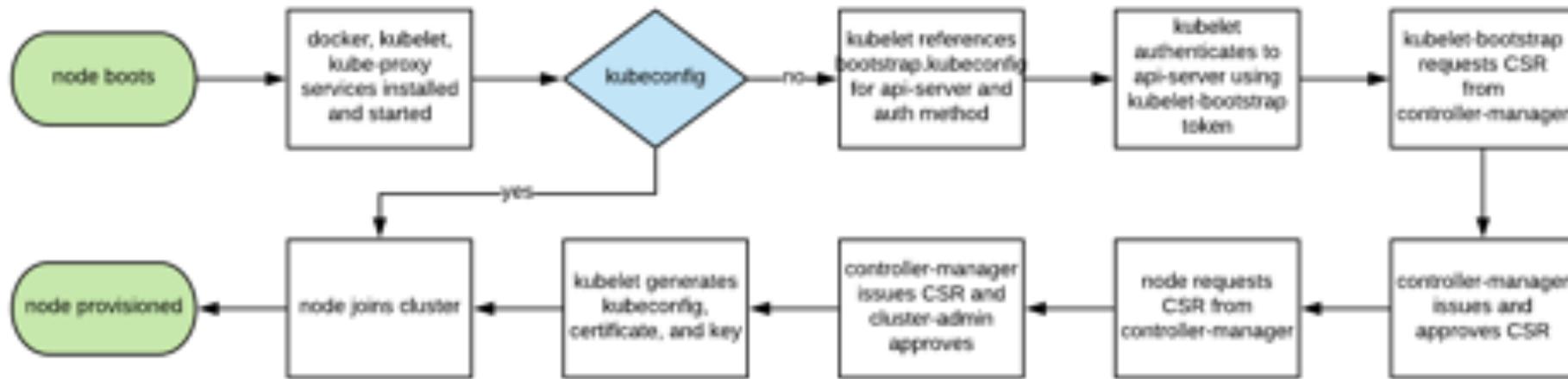
# Kubernetes Cluster Architecture

- Kubernetes nodes can be physical hosts or VM's running a container-friendly Linux (kernel > 3.10)



# Kubernetes Cluster Architecture

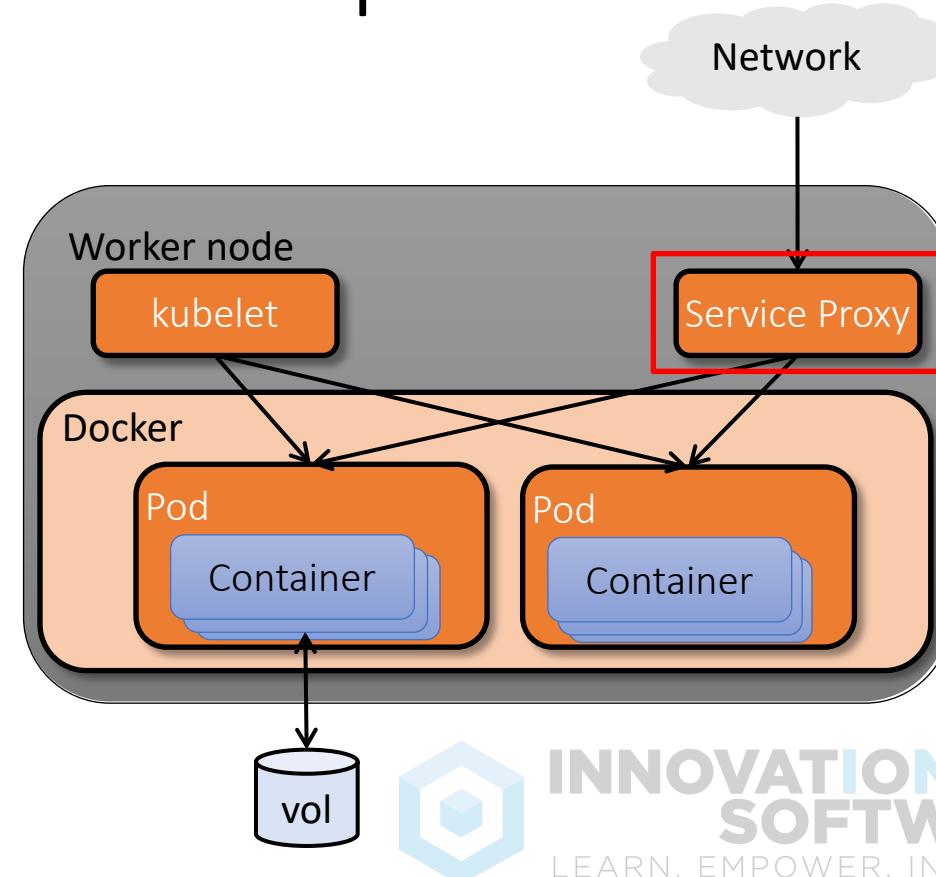
- Kubernetes with TLS bootstrap process



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

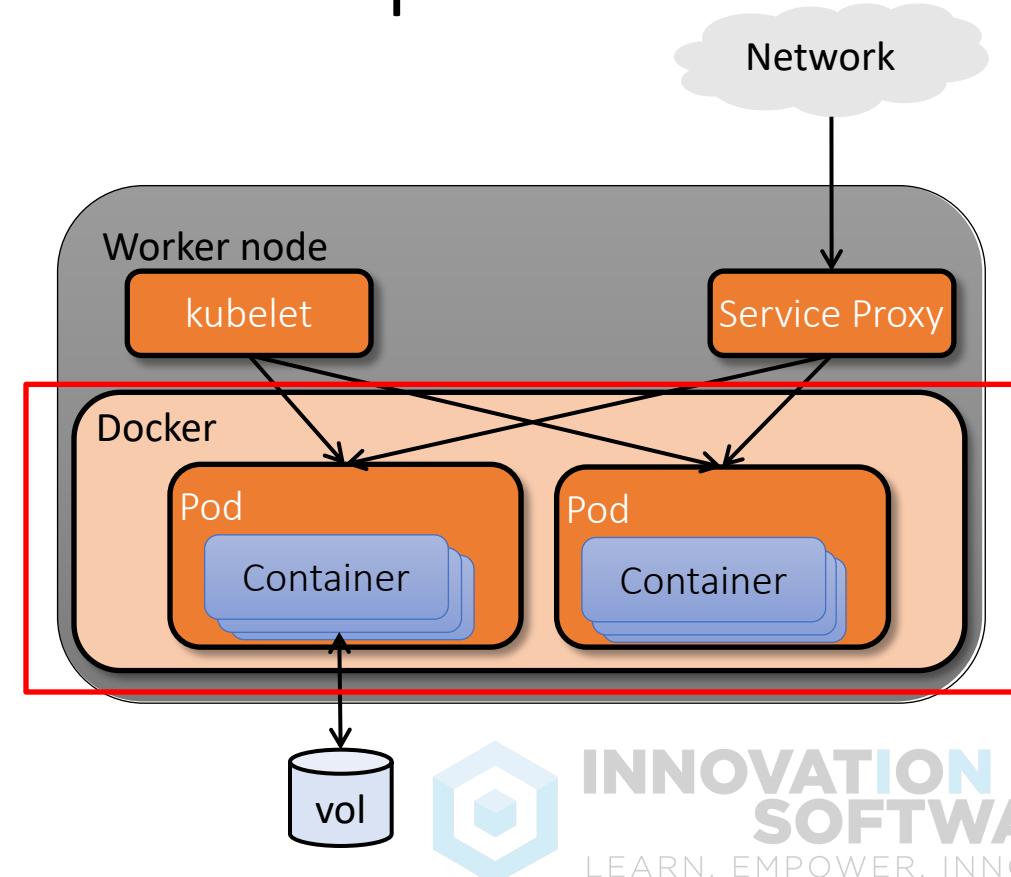
# Kubernetes Worker Node Components

- **kubelet**: local K8s agent that is responsible for operations on the node, including
  - Watching for pod assignments
  - Mounting pod required volumes
  - Running a pod's containers
  - Executing container liveness probes
  - Reporting pod status to system
  - Reporting node status to system
- **Service proxy (kube-proxy)**: enables K8s service abstractions by maintaining host network rules and forwarding connections
- **Docker**: runs the containers



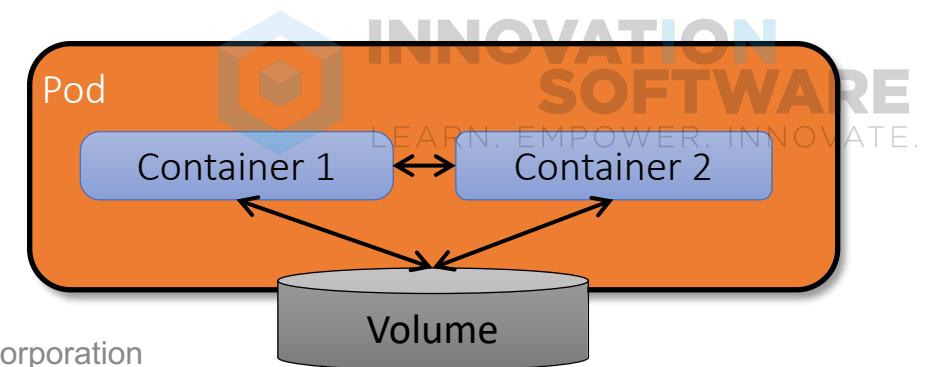
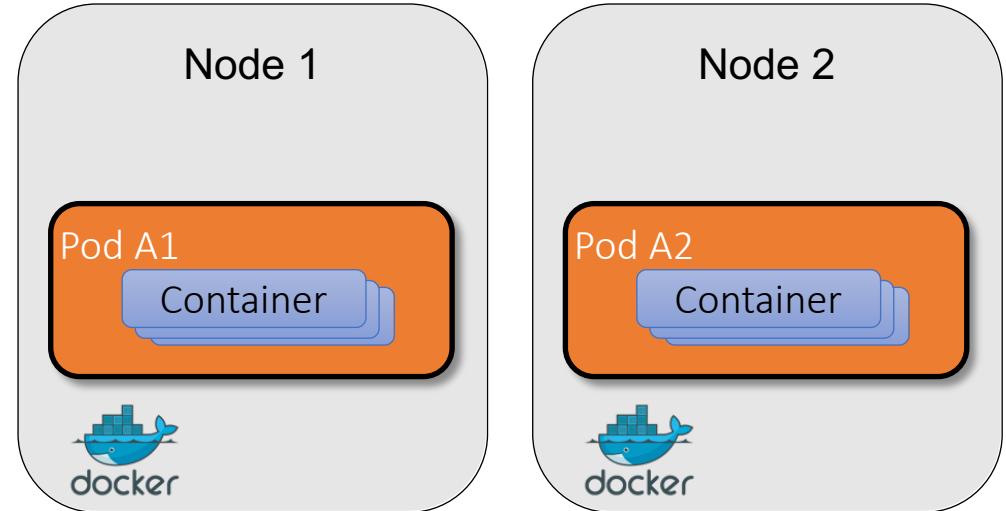
# Kubernetes Worker Node Components

- **kubelet**: local K8s agent that is responsible for operations on the node, including
  - Watching for pod assignments
  - Mounting pod required volumes
  - Running a pod's containers
  - Executing container liveness probes
  - Reporting pod status to system
  - Reporting node status to system
- **Service proxy (kube-proxy)**: enables K8s service abstractions by maintaining host network rules and forwarding connections
- **Docker**: container runtime



# Kubernetes Pods

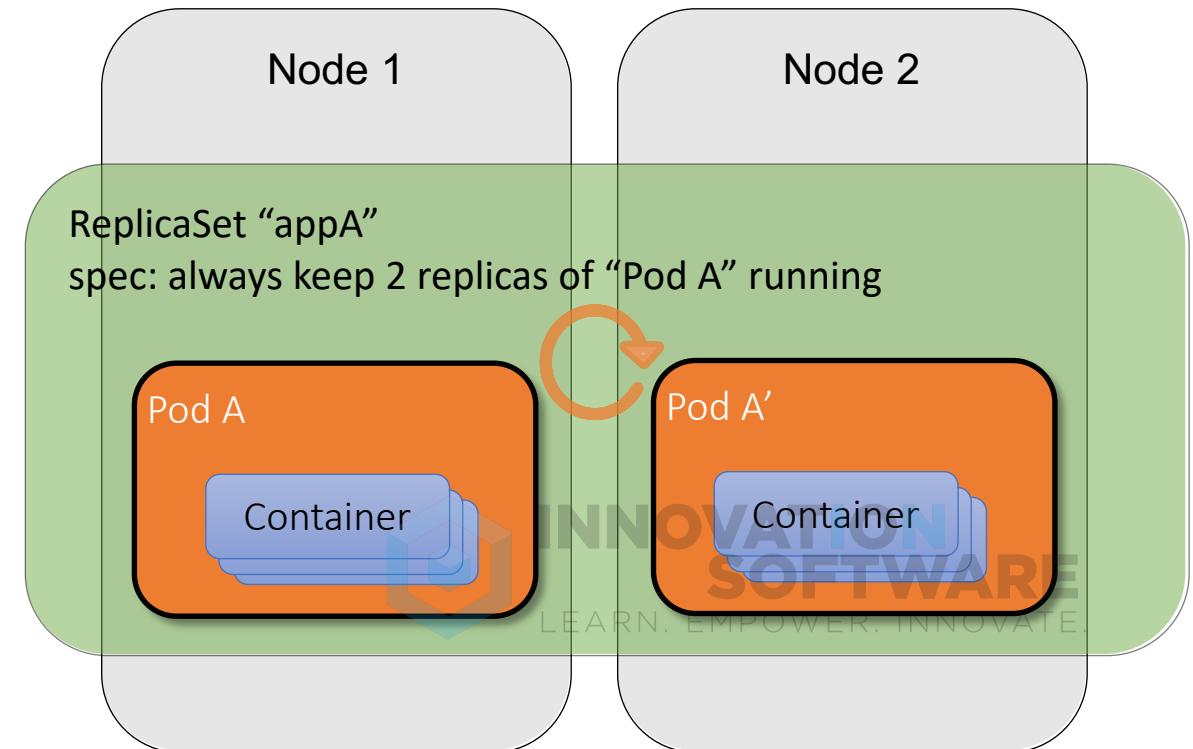
- Smallest K8s workload unit is the **Pod**, a set of co-scheduled containers
- A Pod == an application instance
- Pods can include more than one container, for tightly-coupled application components
- Containers in the same Pod share networking and storage resources
- Kubernetes handles efficient placement of Pods across available Nodes
- Pods and other K8s objects carry user-defined labels



# Controllers for Different Application Patterns

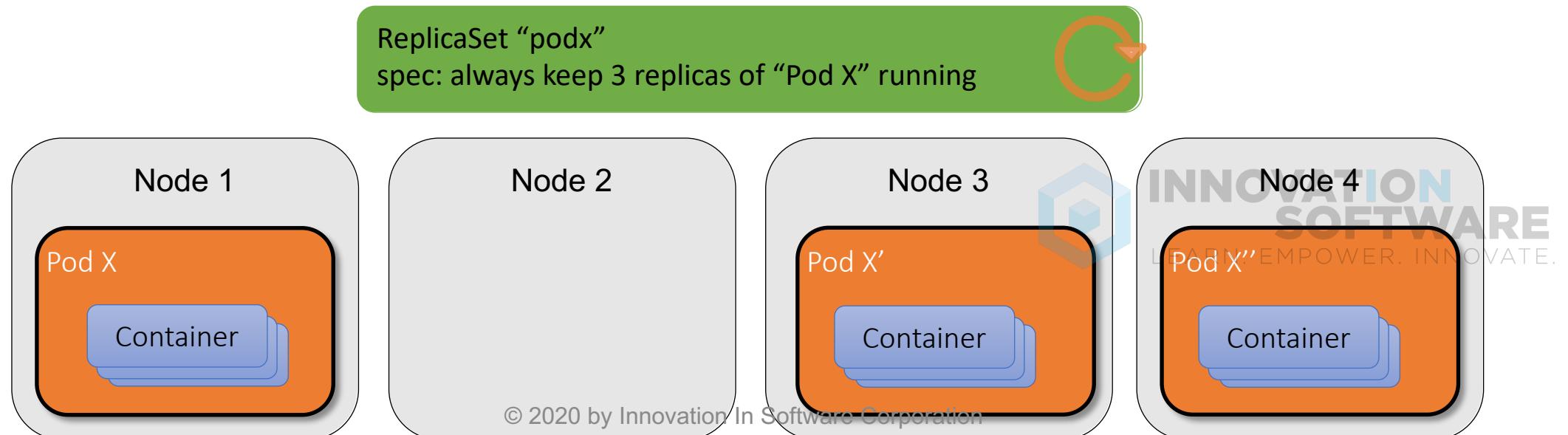
- K8s controller objects used to create and manage Pods according to different application patterns => control loops
- **ReplicaSets** manage sets of replicas of stateless workloads to ensure availability
- **StatefulSets** manage stateful workloads on stable storage to ensure consistency
- **DaemonSets** manage workloads that must run on every node, or set of nodes
- **Jobs** manage parallel batch processing workloads

Controller example: ReplicaSet



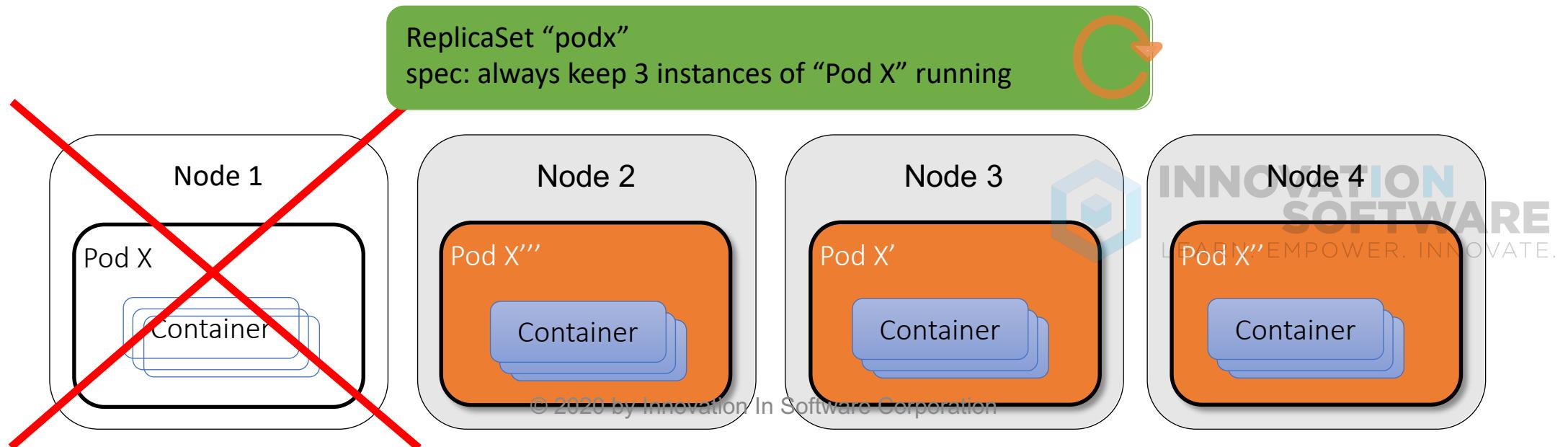
# Controller Example: ReplicaSets

- ReplicaSet configuration specifies how many instances of given Pod exist
- Configuration includes Pod template to define managed Pod configuration
- ReplicaSet used for web applications, mobile back-ends, API's
  - Usually managed by Deployment controllers



# Replication Ensures Application Availability

- When a Node fails, its Pods are lost
- K8s system manages the state of the ReplicaSet back to the declared configuration
- Changing the configuration will result in management to new state, e.g. scale out



# Kubernetes Installation



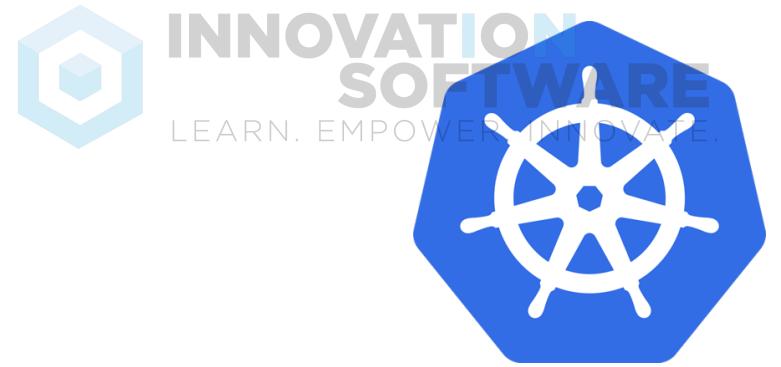
INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Install Kubernetes

- Install pre-requisite packages on all 3 nodes
- Initialize master
- Join nodes to cluster



# Deployments



# What is a Deployment?

*Kubernetes controller optimal for stateless applications*

- Deployments allow you to declaratively manage pods, including replication
- Deployments support
  - Creating, rolling out, and rolling back changes to homogeneous set of pods
  - Scaling set of pods out and back declaratively
- Deployments include
  - Implicit Replica Set controller to handle pod replicas
  - Template spec of pods to be created and managed – no need to separately create pods
- Deployments used for web applications, mobile back-ends, API's



# What if my Application isn't Stateless?

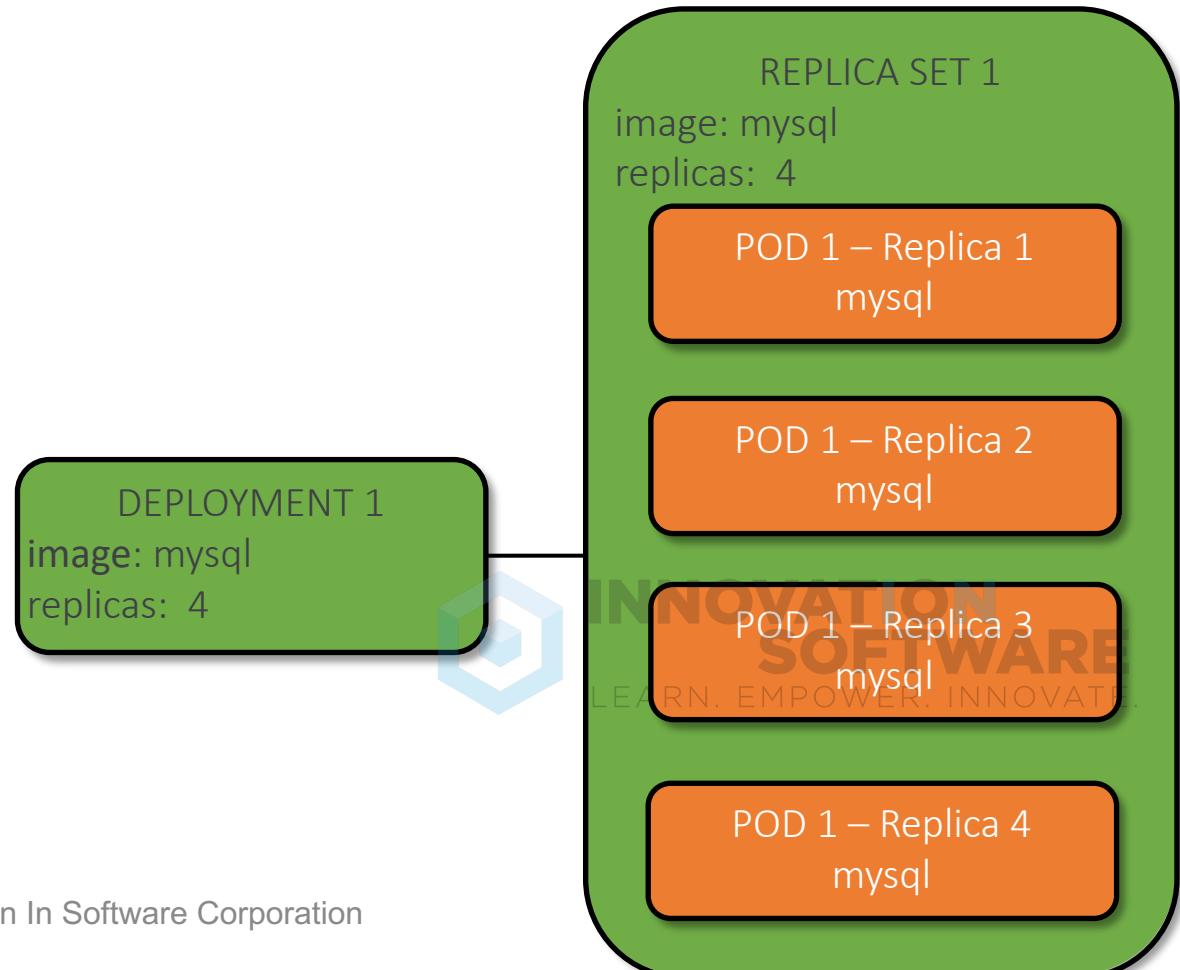
- Kubernetes provides other controller objects for applications that need different deployment schemes
- **StatefulSets** (previously PetSets) control deployment of pods for applications that need more stable deployment contexts
  - Pods in StatefulSets have unique ordinal, stable network identity and stable storage using persistent volumes
  - When pods are deployed, they are created in sequence of ordinals 1..N
  - Pod N must be running and ready before Pod N+1 is deployed
  - When pods are destroyed, they are terminated in reverse sequence N..1
- **DaemonSets** ensure that a replica of a specified pod is running on every node (or every selected node) in the cluster
- **Jobs** manage sets of pods where N must run to successful completion



# Deployments Control ReplicaSet Controllers

*Definition of how many replicated Pods should exist*

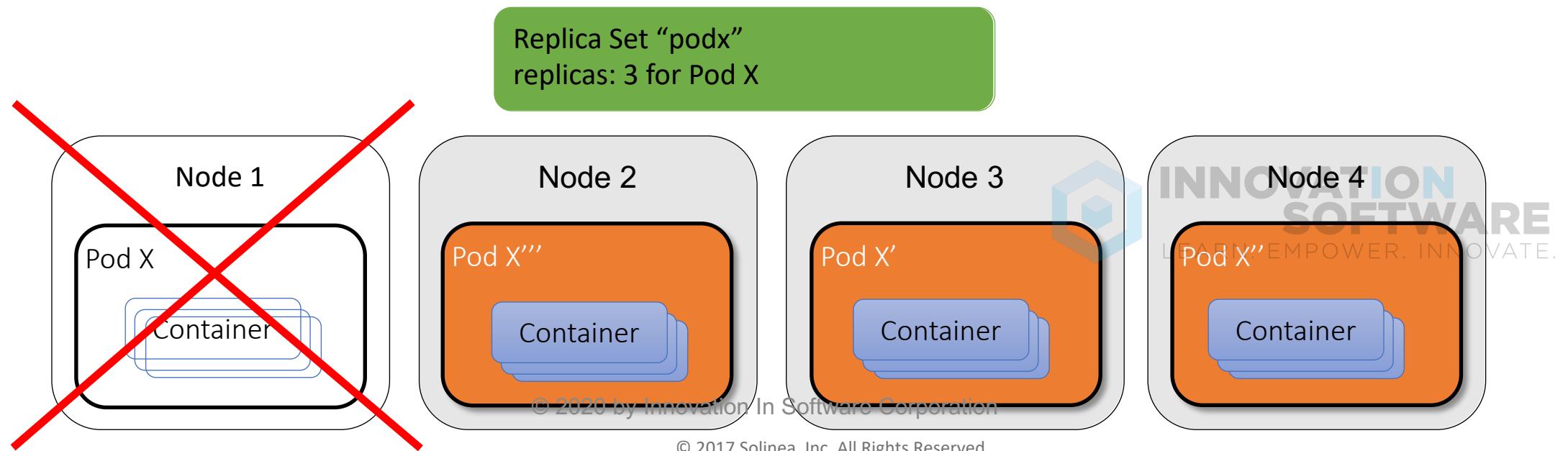
- Deployment creates and manages a Replica Set that manages a set of pods
- Replica count can be adjusted as needed to scale the Replica Set out and back
- Replica Set successor to the ReplicationController object



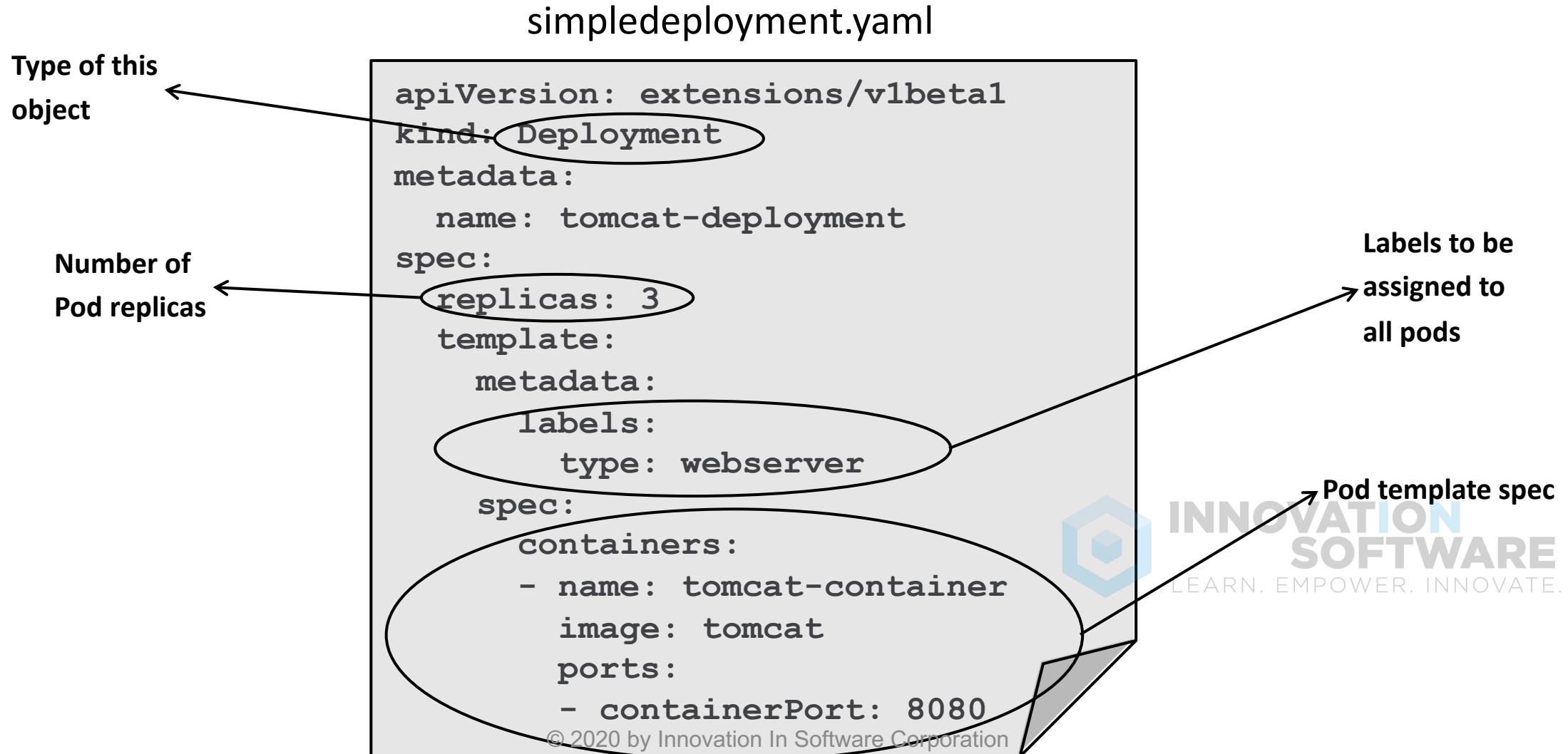
# What is a Replica Set?

*Provides scaling and high availability*

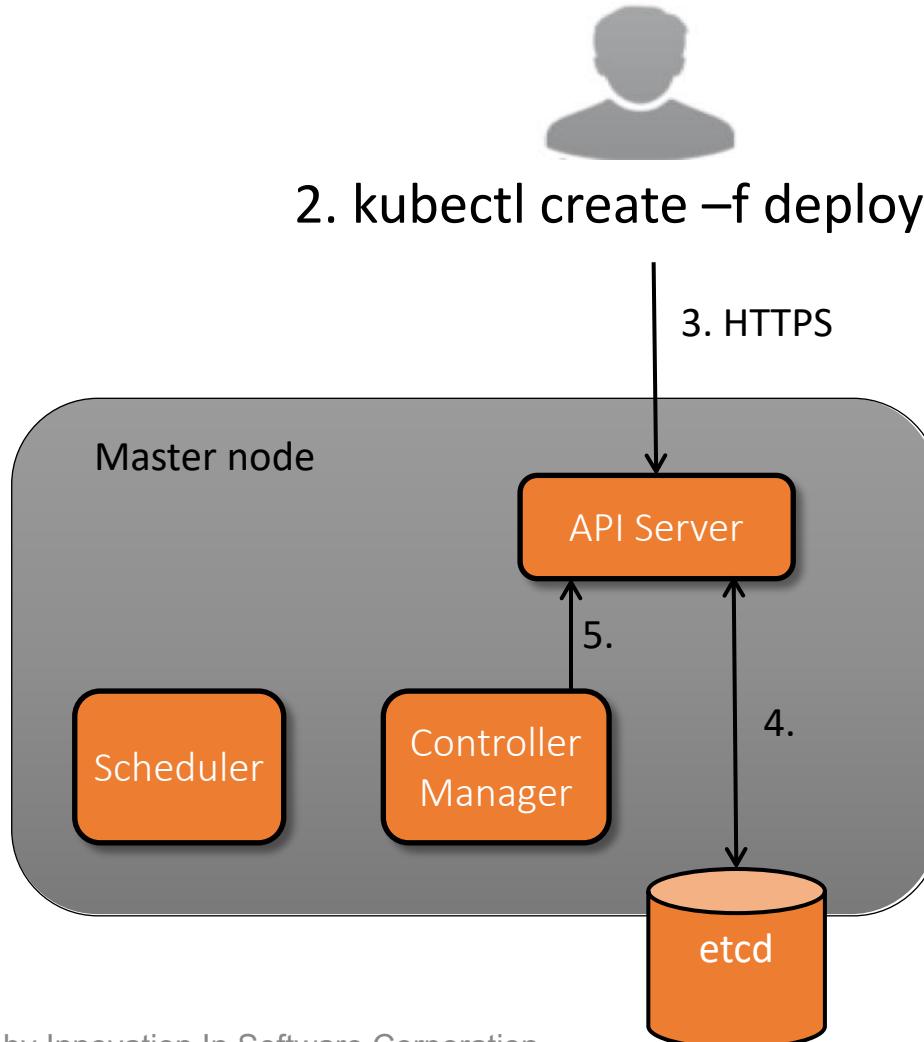
- Replica count can be changed to provide scaling on demand as needed
- If the node hosting a pod fails, the Kubernetes cluster will recreate the pod elsewhere to achieve the target number of replicas



# Examining a Deployment Manifest File



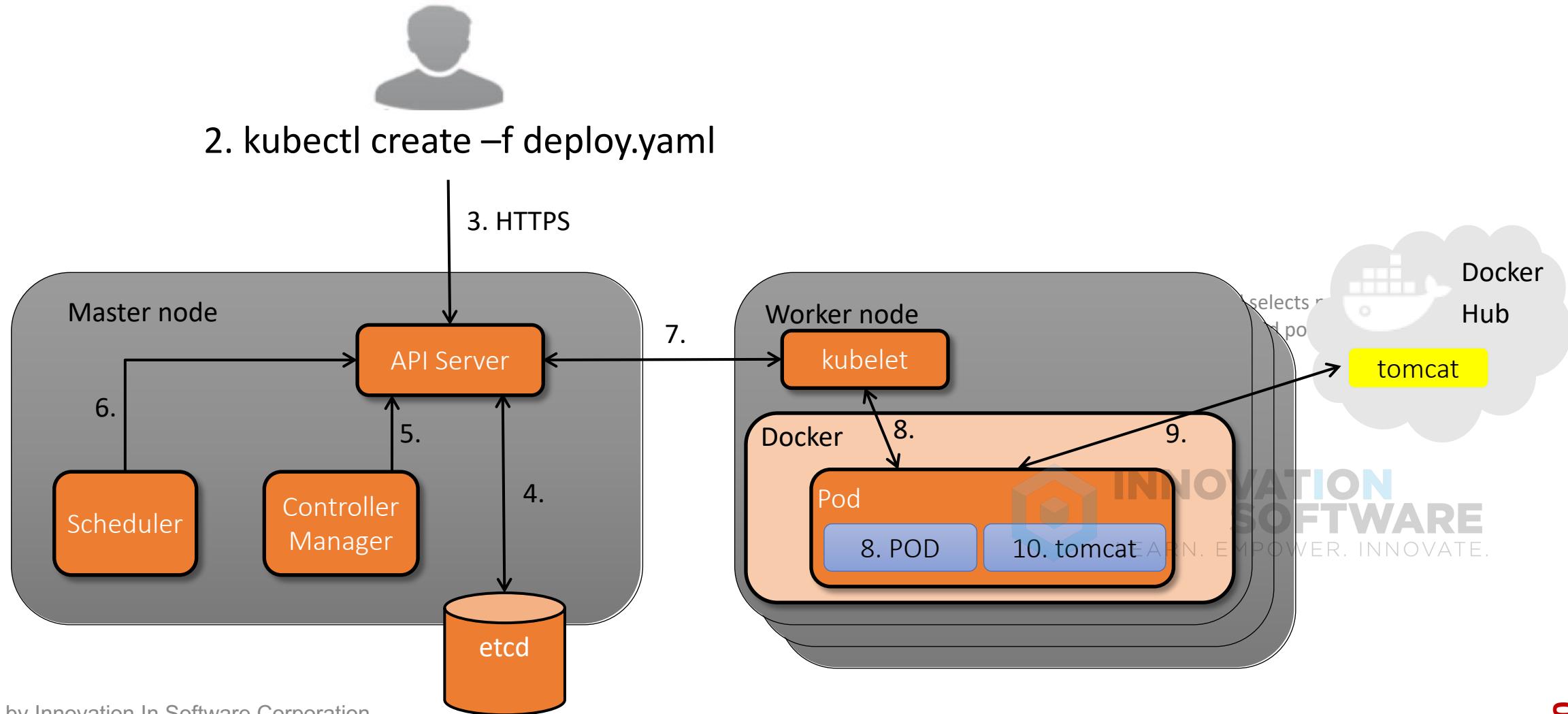
# Deployment Creation Process



1. User writes a deployment manifest file
2. User requests creation of deployment from manifest via CLI
3. CLI tool marshals parameters into K8s RESTful API request (HTTP POST)
4. kube-apiserver creates new deployment object record in etcd, and new Replica Set object
5. kube-controller-manager sees new Replica Set and
  - Evaluates state of existing vs. required replicas
  - Submits pod creation requests to API to create required number of replicas



# Deployment Creation Process



# Deployments Management



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE



# Deployments Control ReplicaSets

*The deployment creates a ReplicaSet that controls pod creation*

Deployment name defined in the yaml	Number of replicas defined in the spec	Number of existing pods	Number of Pods that match the Deployment config	Actual number of Pods running
\$ kubectl get deployments	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE
NAME				
tomcat-deployment	3	3	3	3

\$ kubectl get rs	NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
	tomcat-deployment-1699985759	3	3	3	3	2m



# Replica Set Details

*Replica Set name is <Deployment name>-<pod template hash value>*

```
$ kubectl describe replicaset tomcat-deployment-1699985759
```

Name: tomcat-deployment-1699985759

Namespace: default

Image(s): tomcat

Selector: pod-template-hash=1699985759,type=webserver

Selector uses labels  
from pod template  
and template hash

Labels: pod-template hash-1699985759

type=webserver

Replicas: 3 current / 3 desired

Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed

...



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Pod Naming

*Pod name is <Replica Set name>-<pod unique random string>*

```
$ kubectl get pods
```

NAME	READY	STATUS
tomcat-deployment-1699985759-h11sz	0/1	ContainerCreating
tomcat-deployment-1699985759-ka13j	0/1	ContainerCreating
tomcat-deployment-1699985759-u03b5	1/1	Running

```
$ kubectl get pods
```

NAME	READY	STATUS
tomcat-deployment-1699985759-h11sz	1/1	Running
tomcat-deployment-1699985759-ka13j	1/1	Running
tomcat-deployment-1699985759-u03b5	1/1	Running

Pod name based on  
Replica Set name

Random string to  
differentiate Pods

Status of Pods



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Modifying a Deployment to Trigger a Rollout

*Multiple ways to change a Deployment configuration*

- Change the manifest file and apply it via *kubectl apply*
- Change specific Deployment attribute via *kubectl set*
- Edit the Deployment config in the cluster via *kubectl edit*
- **Any changes to the Pod template will trigger a rollout of the Deployment to create and replicate new Pods with the new template**
  - **This means any changes – even things like pod labels!**



# Pausing a Deployment

*Pause a Deployment to temporarily halt rollout of updated pods*

```
$ kubectl set image deployment/tomcat-deployment tomcat-container=tomcat:8.5.5;  
kubectl rollout pause deployment/tomcat-deployment  
deployment "tomcat-deployment" image updated  
deployment "nginx-deployment" paused
```

```
$ kubectl rollout resume deployment/tomcat  
deployment "tomcat-deployment" resumed
```

```
kubectl rollout status deployment/tomcat-deployment  
deployment "tomcat-deployment" successfully rolled out
```



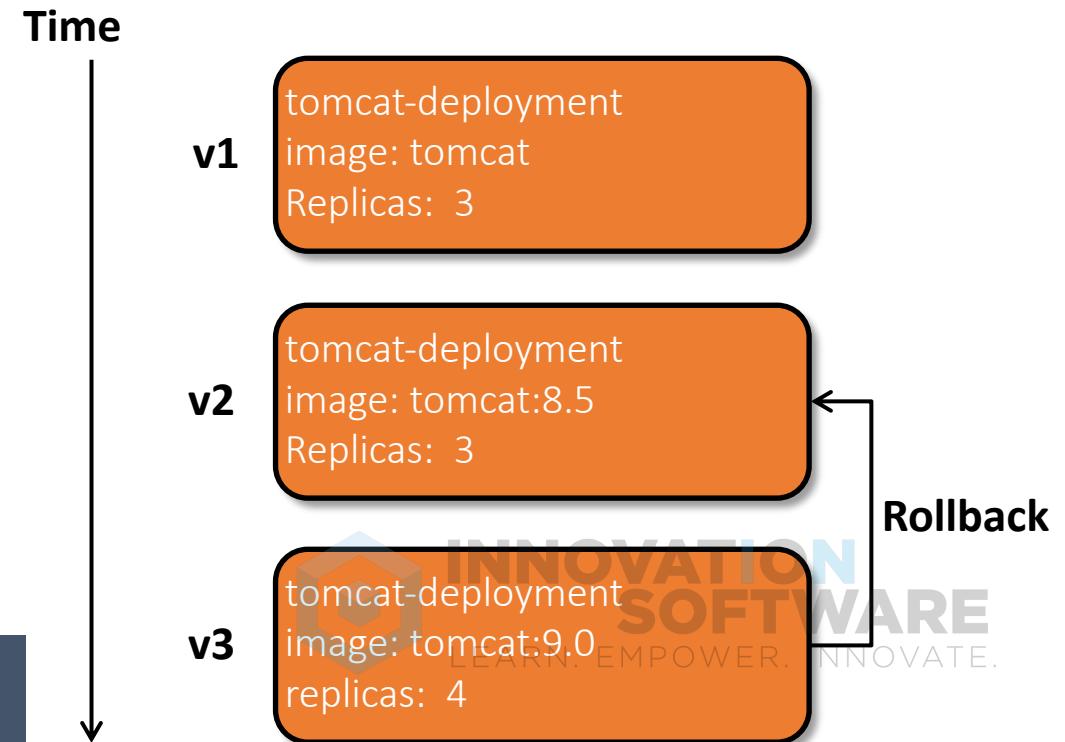
**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Checking Deployment Rollout History

*Kubernetes tracks revisions made to a Deployment*

- Users can query history of a deployment and see how many versions existed, and what change was made
  - Need to use *-record* flag on kubectl to record details of change commands
  - History allows you to roll back state of a Deployment to a previous version

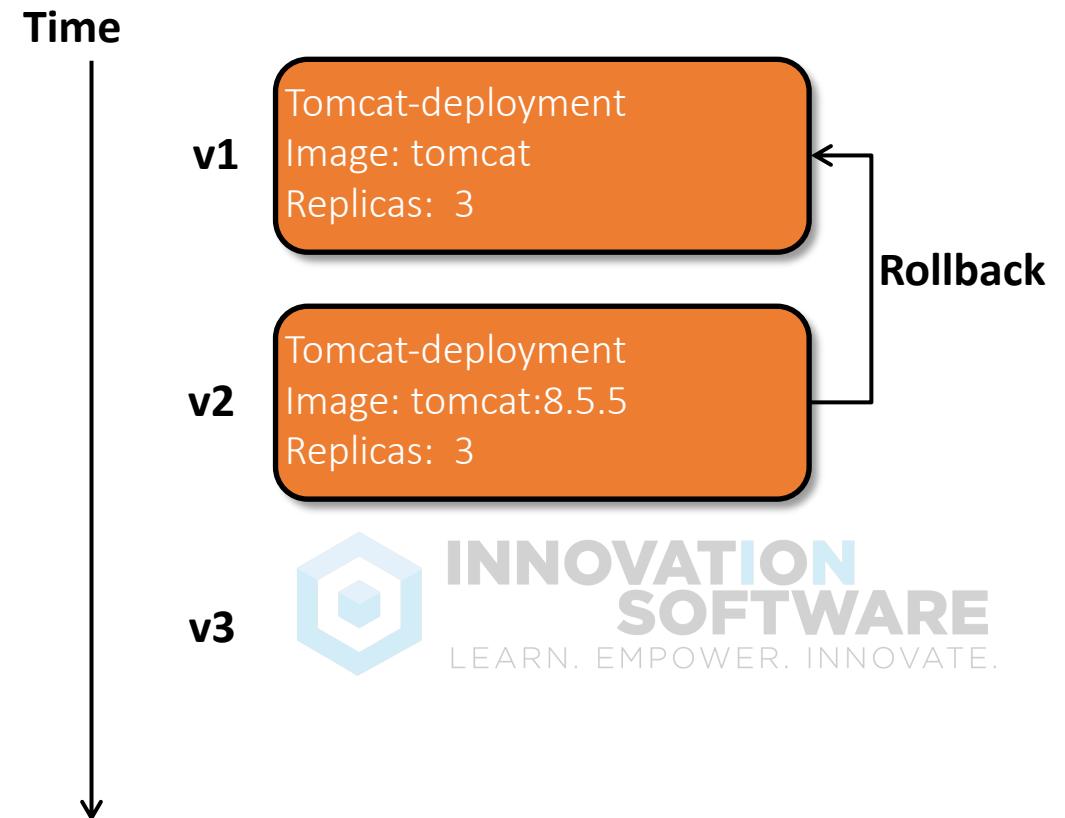
```
$ kubectl rollout history deploy/<deployment>
```



# Rolling back a Deployment

*Undoing Deployment changes via rollback operation*

- You can undo the last change or roll back to a specific version
- The revision order will be changed to reflect the change
  - In this example, revision v1 will become the new revision v3
  - Version numbers increase monotonically
- Revision state stored in the corresponding Replica Sets

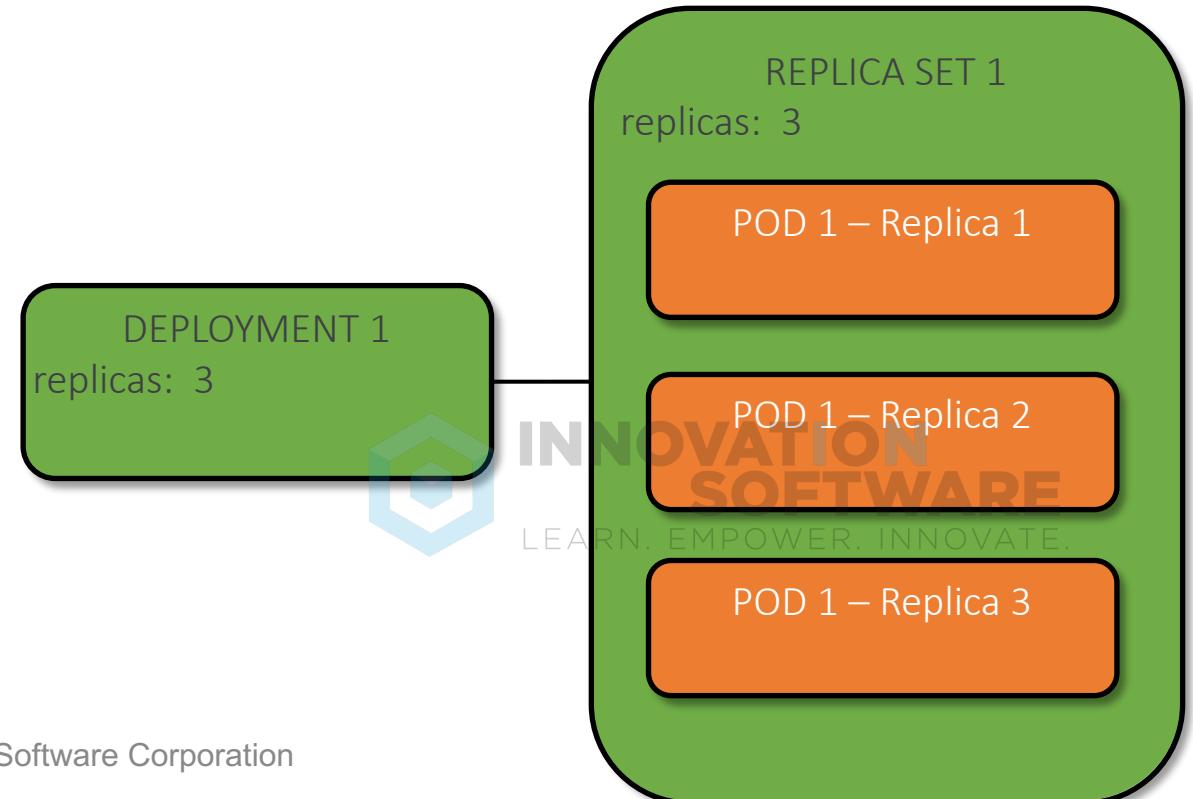


**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

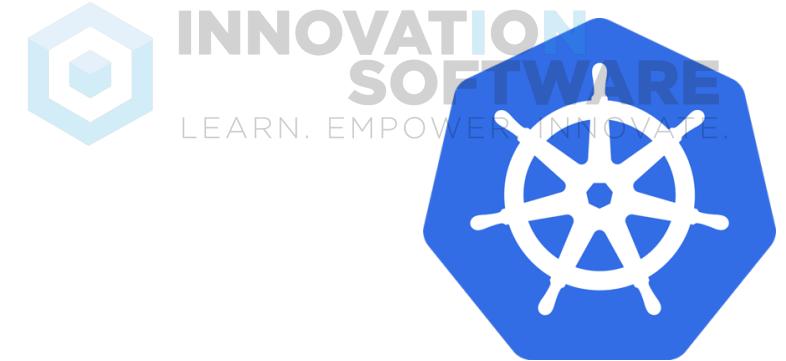
# Deleting a Deployment

*Deleting a Deployment will delete its Replica Set and all its Pods*

- By default, when Deployment is deleted, its Replica Sets are deleted
- Deletion of Replica Set cascades to deletion of pods managed by the rs
- Any Replica Sets reflecting previous versions of the Deployment will also be deleted



# Deployment Strategies Overview



# What is a Deployment Strategy?

*Approaches to manage risks on updating Deployments*

- On each Deployment update/change, all pods in the deployment will be deleted and recreated
- Recreation process can have service impacts, especially for large Deployments
- A Deployment strategy defines how this rebuild process is done, to minimize downtime due to application failures or malfunctions



# Types of Deployment Strategies

*Kubernetes supports two basic strategies, but users can also leverage multiple Deployments when applying changes*

- Strategies for single Deployments
  - Recreate
  - RollingUpdate
- Strategic approaches using two Deployments with a Service
  - Canary deployments
  - Blue/Green deployments



Each approach has a specific behavior and advantages/disadvantages.

# Deployment Strategy: Recreate



INNOVATION  
SOFTWARE

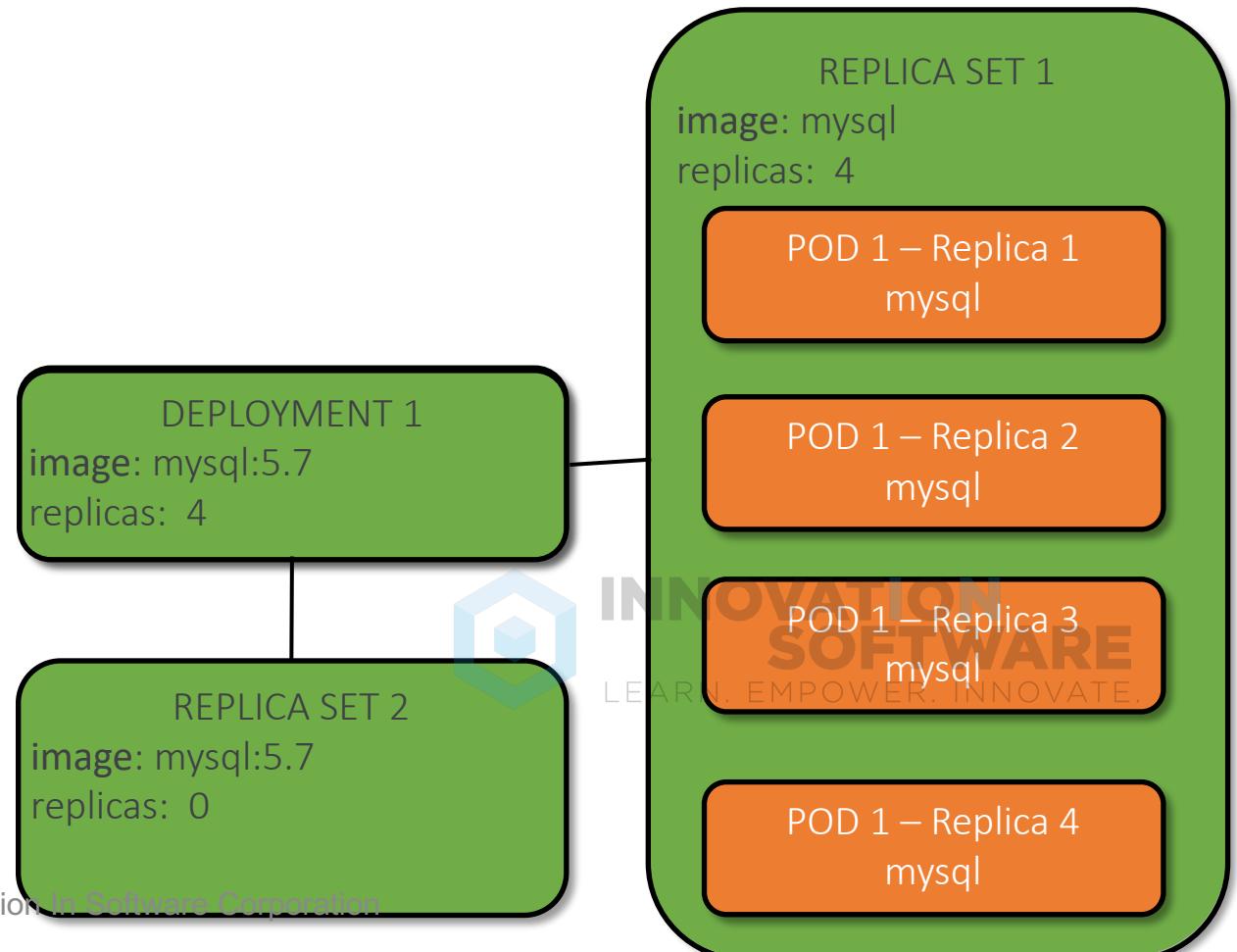
LEARN. EMPOWER. INNOVATE.



# Deployment Strategy: Recreate

*Simplest strategy for deployments*

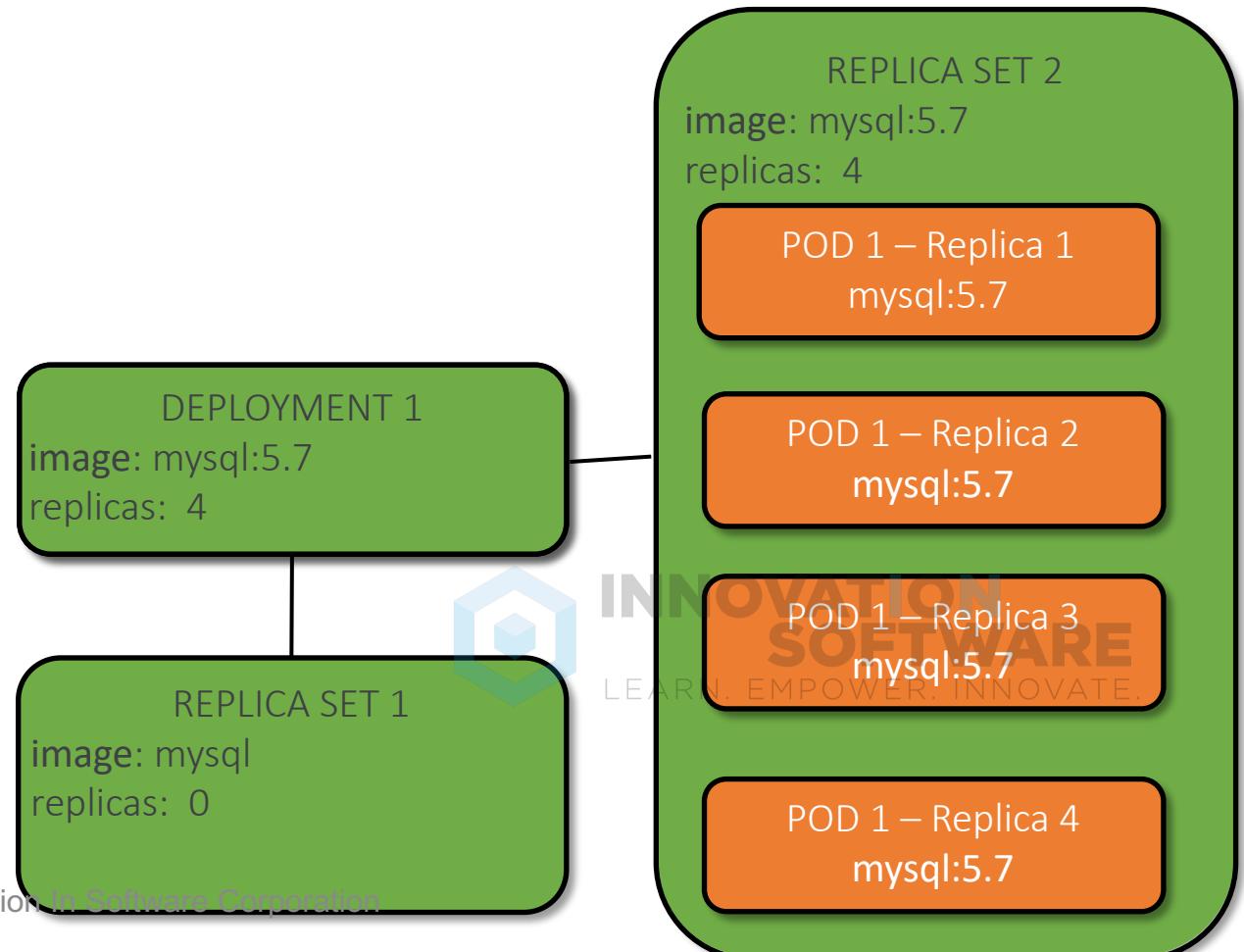
- When a change is made to a Deployment's spec, all Pods are removed and then recreated
  - Old Replica Set pods are killed
  - Then, new Replica Set starts pods
- May lead to downtime during the process while new pods are started



# Deployment Strategy: Recreate

*Simplest strategy for deployment updates*

- When a change is made to a Deployment's template, all Pods are removed and then recreated
  - Old Replica Set pods are killed
  - New Replica Set starts pods
- May cause downtime due to delay between old pods terminating and new pods becoming available



# Deployment Strategy: Recreate

*Strategies are defined in the spec of a Deployment*

- **strategy** parameter in Deployment spec sets the strategy to be used for updates
- If no parameter value is set, the default is **RollingUpdate**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
spec:
  replicas: 3
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        type: webserver
    spec:
      containers:
        - name: tomcat-container
          image: tomcat
          ports:
            - containerPort: 8080
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Deployment Strategy: RollingUpdate



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.



# Deployment Strategy: Rolling Update

***RollingUpdate** is **DEFAULT** strategy for Deployments*

- When a change is made to the Deployment, the old Replica Set pods are scaled down as new pods are created by the new Replica Set
- A minimum number of running Pods is specified, so the Deployment will never be totally out of Pods to respond to service requests
- During the update process, the requested replica count may be temporarily exceeded



# Deployment Strategy: RollingUpdate

*Configure parameters to control the update process*

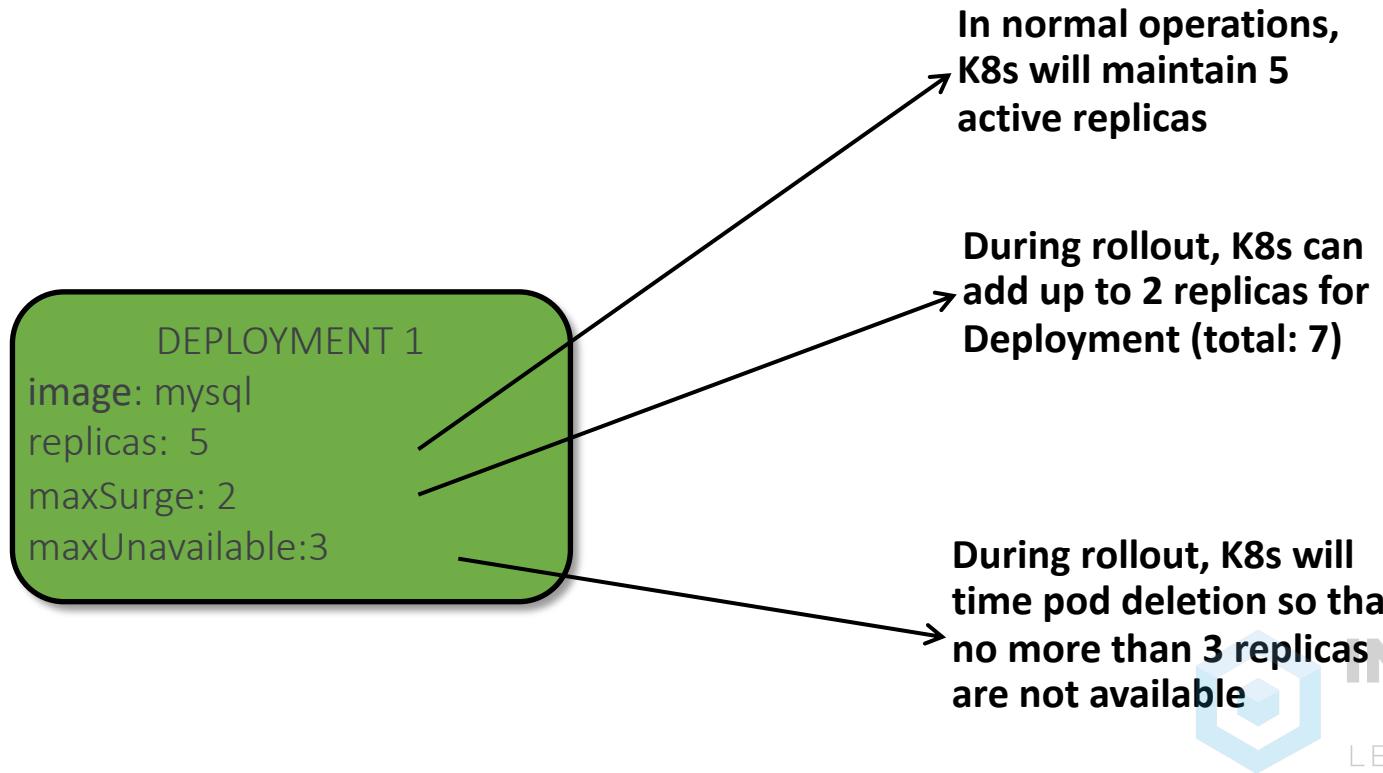
```
...  
  metadata:  
    name: tomcat-deployment  
  spec:  
    replicas: 3  
    strategy:  
      type: RollingUpdate  
      rollingUpdate:  
        maxSurge: 25%  
        maxUnavailable:10%  
    template:  
      metadata:  
        labels:  
          type: webserver  
...  
...
```

- **maxSurge**: number or percentage of additional Pods that can be created exceeding the replica count during update
  - Default value of 25%
- **maxUnavailable**: number of Pods that can be unavailable during the update
  - Default value of 25%



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Deployment Strategy: RollingUpdate

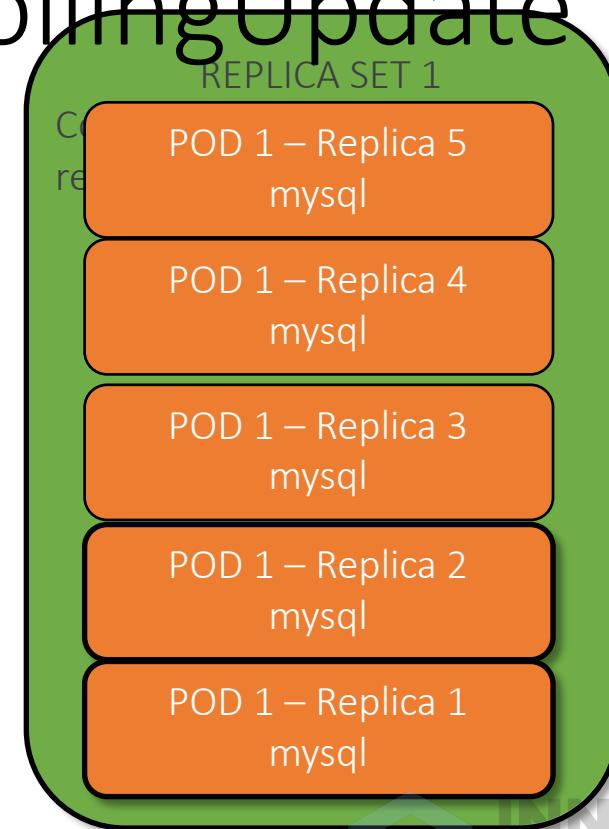


**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Deployment Strategy: RollingUpdate

Initial State

DEPLOYMENT 1  
image: mysql  
Replicas: 5  
maxSurge: 2  
maxUnavailable:3



```
$ vi simpledeployment.yaml
...
  image: mysql:5.7
...
$ kubectl apply -f simpledeployment.yaml
```

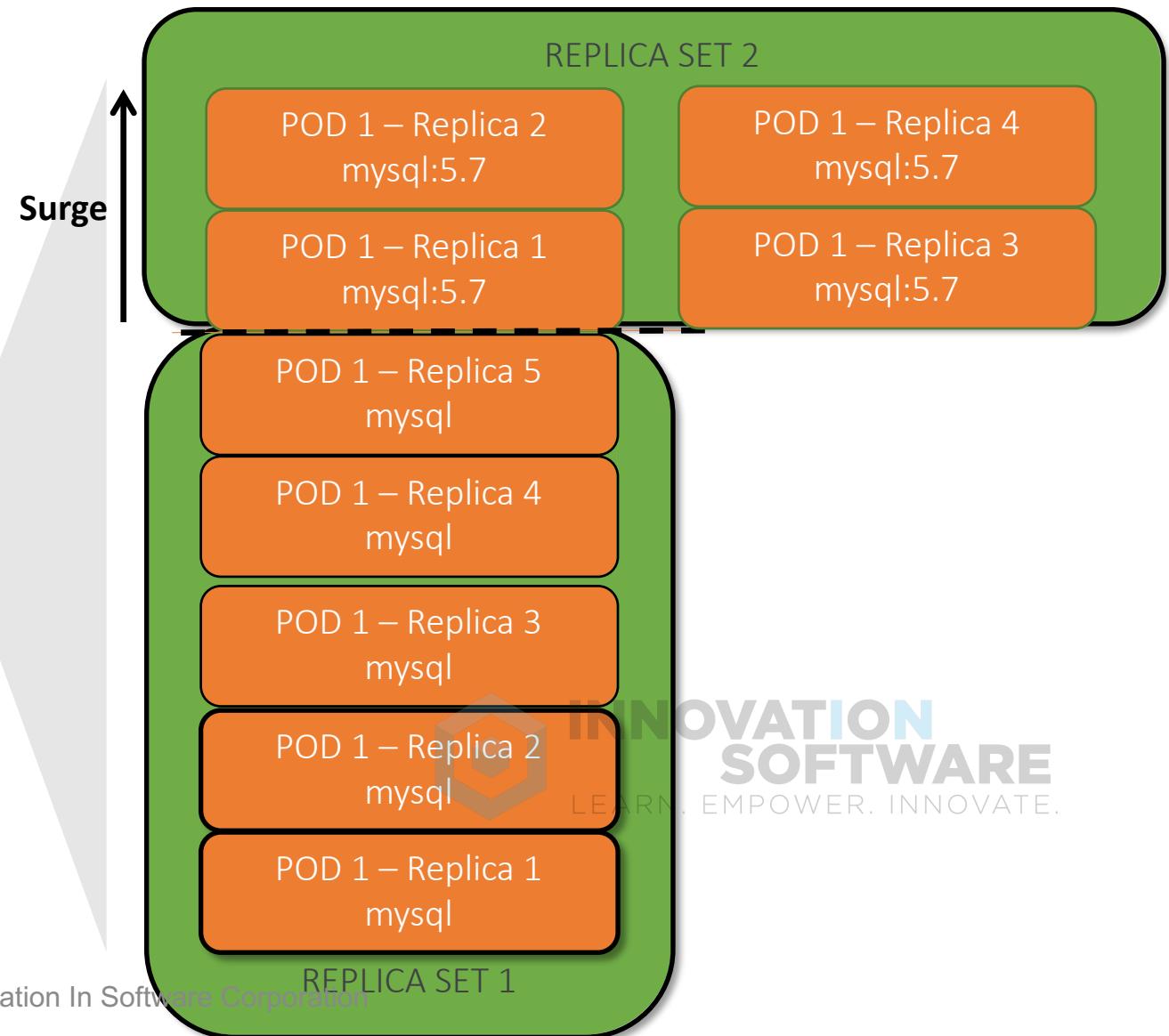


# Deployment Strategy: RollingUpdate

Rollout in progress

DEPLOYMENT 1  
image: mysql:5.7  
Replicas: 5  
maxSurge: 2  
maxUnavailable:3

- Initial surge of new pods on new Replica Set
- Original Replica Set scaled back as new RS scaled out



# Deployment Strategy: RollingUpdate

Rollout complete

DEPLOYMENT 1

image: mysql:5.7  
replicas: 5  
maxSurge: 2  
maxUnavailable:3

- By default, old, inactive Replica Set saved – previous version of the Deployment

REPLICA SET 2

image: mysql:5.7  
replicas: 5

POD 1 – Replica 5  
mysql:5.7

POD 1 – Replica 4  
mysql:5.7

POD 1 – Replica 3  
mysql:5.7

POD 1 – Replica 2  
mysql:5.7

POD 1 – Replica 1  
mysql:5.7

REPLICA SET 1

image: mysql  
replicas: 0

# Updating Using Multiple Deployments



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.



# RollingUpdate using Multiple Deployments

*Controlled testing of new versions in production*

- Assume an application running as a Deployment, exposed as a Service
- To apply a new application version in production, a second Deployment can be used using labels in common with the first Deployment
  - Canary deployment allows for limited testing of new version in production
  - Blue/green deployment



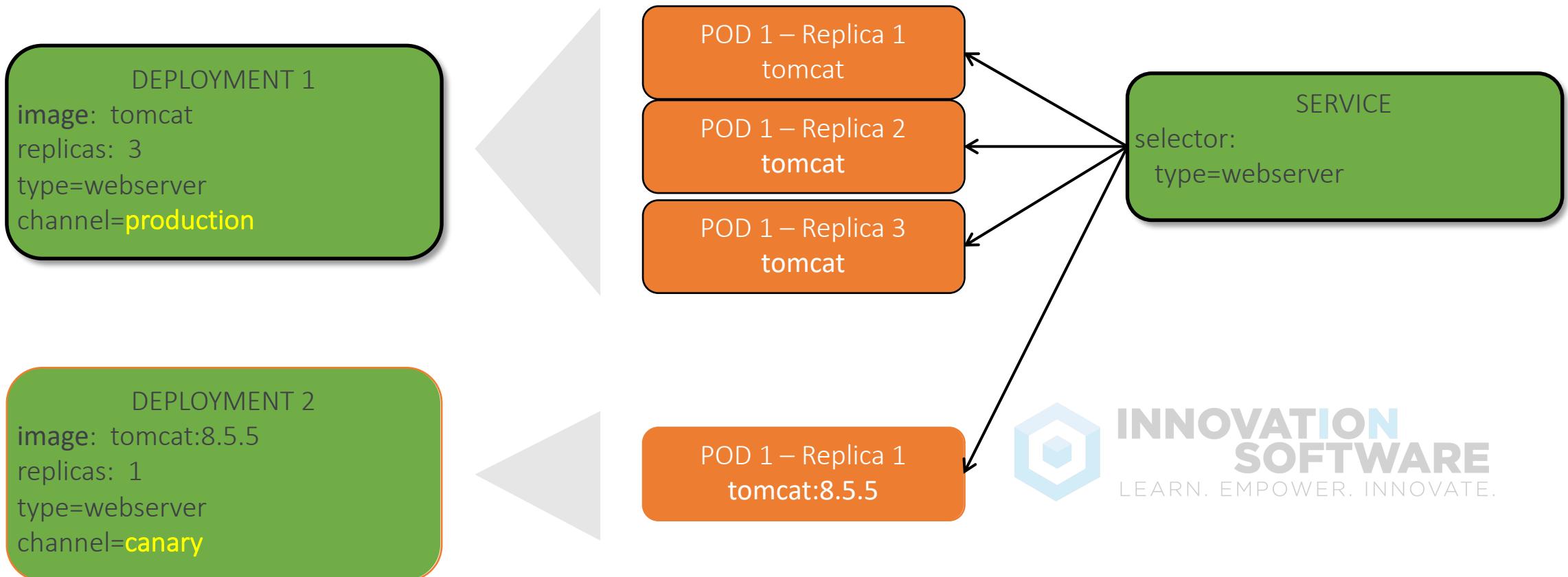
# Strategic Approach: Canary Deployment

*Controlled testing of the update on production*

- Consider a Service selecting pods from a Deployment of application pods
- In a canary deployment, a second Deployment (Canary Deployment) is created with pods for the new version, with labels matching the Service's selector
- Service directs some requests to pods on the Canary, allowing testing of changes in production
- If a malfunction is detected, it will only impact a small portion of the Pods and can be undone.



# Strategic Approach: Canary Deployment



# Strategic Approach: Canary Deployment

*Decisions after running the canary Deployment in production*

- If the application error rate is not increasing and Canary Deployment is stable:
  - The main Deployment can be updated to the newer version (using Rolling Update for example) and then the Canary can be discarded; OR
  - The Canary can be scaled up and reconfigured and the old Deployment can be discarded.
- If the test results in failure, the Canary deployment can be deleted



# Strategic Approach: Blue/Green Deployment

*Complete environment switch from one version to another*

- With a Blue/Green deployment, you create a new full-scale Deployment in addition to the current production Deployment
- Reconfiguring the pod label selector on the application's Service allows choice of directing requests to old Deployment or new Deployment
- Similar to effect of Replace strategy without application downtime



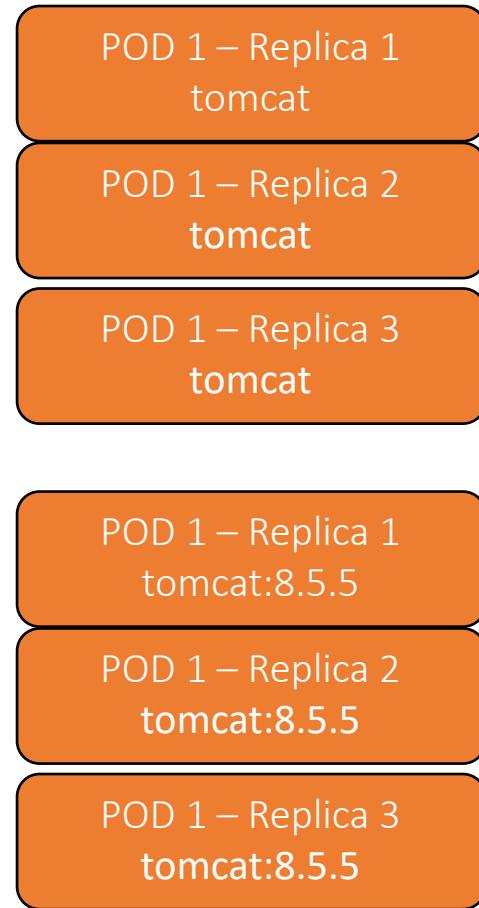
# Strategic Approach: Blue/Green Deployment

DEPLOYMENT 1

```
image: tomcat  
replicas: 3  
type=webserver  
color=blue
```

DEPLOYMENT 2

```
image: tomcat:8.5.5  
replicas: 3  
type=webserver  
color=green
```



SERVICE

```
selector:  
type: webserver  
color=blue
```



**INNOVATION SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Strategic Approach: Blue/Green Deployment

DEPLOYMENT 1

image: tomcat  
replicas: 3  
type: webserver  
color=blue

DEPLOYMENT 2

image: tomcat:8.5.5  
replicas: 3  
type: webserver  
color=green



SERVICE

selector:  
type: webserver  
color=green



# Scaling Deployments



INNOVATION  
SOFTWARE

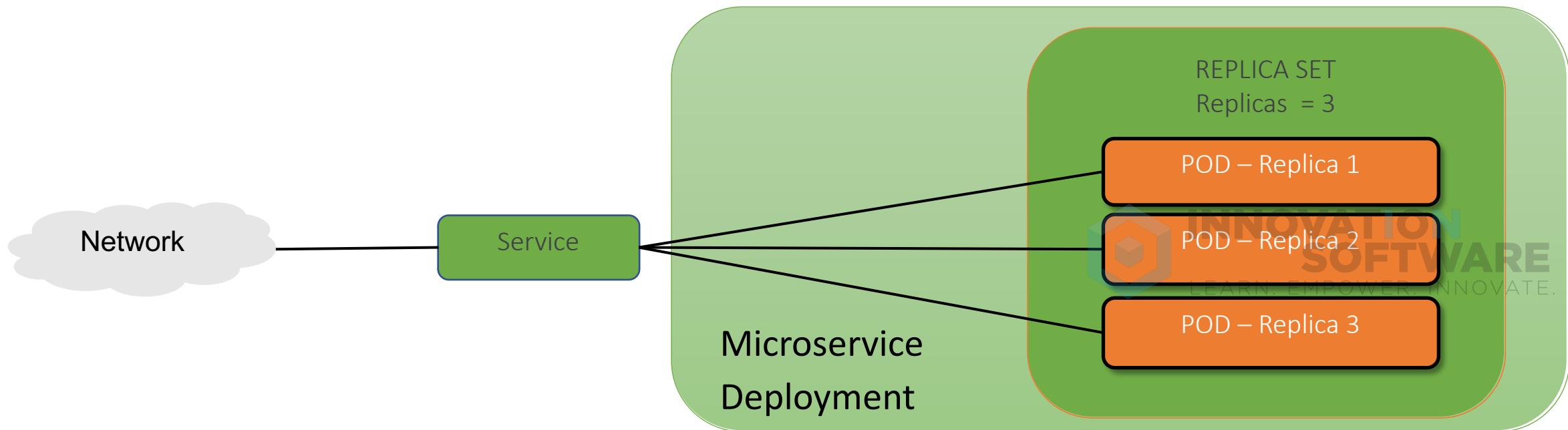
LEARN. EMPOWER. INNOVATE.



# Kubernetes Deployments for Microservices

*Kubernetes is perfect for Microservices management*

- Applications built as services or collections of microservices easily modeled using Kubernetes Deployment and Service resources
- Key property is that application or component can be scaled horizontally



# Application Scaling Strategies

*Several ways to achieve scalability*

- **Pre-defined application scale:** application deployed with a predefined, static number of resources => not the Kubernetes way
- **Manual scaling:** deployment scale reconfiguration driven by operator
- **Auto-scaling:** automatic scaling of resources based on a defined trigger (number of hits, CPU usage, etc)

# Application Scaling Strategies

## *Manual scaling*

- K8s supports manual scaling of Deployment to adjust replica count
- Operator just needs to adjust declaration of how many replicas are desired, and K8s system will manage to that number
  - Pod creation and placement completely automatic
  - Pods automatically re-created if nodes fail

```
$ kubectl scale --replicas=2 deployment/tomcat
```

```
$ kubectl scale --replicas=8 deployment/tomcat
```

```
$ kubectl scale --replicas=4 deployment/tomcat
```

```
$ kubectl edit deployment/tomcat
```



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.

```
$ kubectl apply -f simpledeployment.yaml
```

# Application Scaling Strategies

## *Auto-scaling in Kubernetes*

- Kubernetes has a controller object for automatic scaling of Deployments (or Replica Sets or ReplicationControllers)
- HorizontalPodAutoscaler is control loop to adjust scale of pod set depending on one or more metrics, evaluated at configurable interval (default 30s)
  - Requires Heapster to be deployed on the cluster to provide metric data
  - Metrics include CPU and RAM utilization for pods
- Typical scenario: increase Deployment replica count when average Pod CPU utilization is above threshold value for specified period of time
- Note: application must support horizontal scaling!

# Automatic Scaling of a Deployment

## *Creating a HorizontalPodAutoscaler resource*

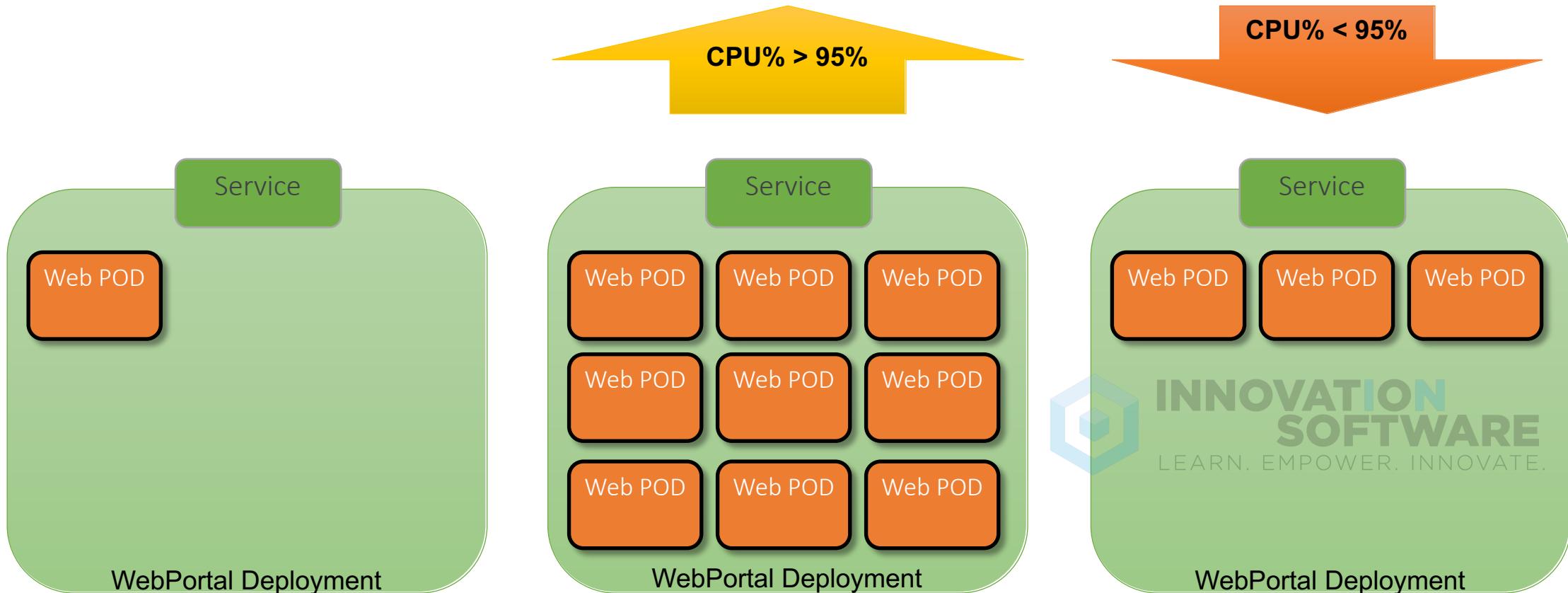
- The hpa is created similarly to any other Kubernetes resource
  - Use a manifest file for source tracking
  - Can create directly with *kubectl*
- 30 seconds is the default for considering the threshold
- Scaling is metrics-driven, either resource metrics from Heapster or custom metrics API

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: tomcat-autoscaler
spec:
  maxReplicas: 10
  minReplicas: 2
  scaleTargetRef:
    kind: Deployment
    name: tomcat-deployment
  targetCPUUtilizationPercentage: 75
```

*simplescaler.yaml*

# Auto-Scaling of a Deployment

*HorizontalPodAutoscaler with a 95% CPU usage threshold*



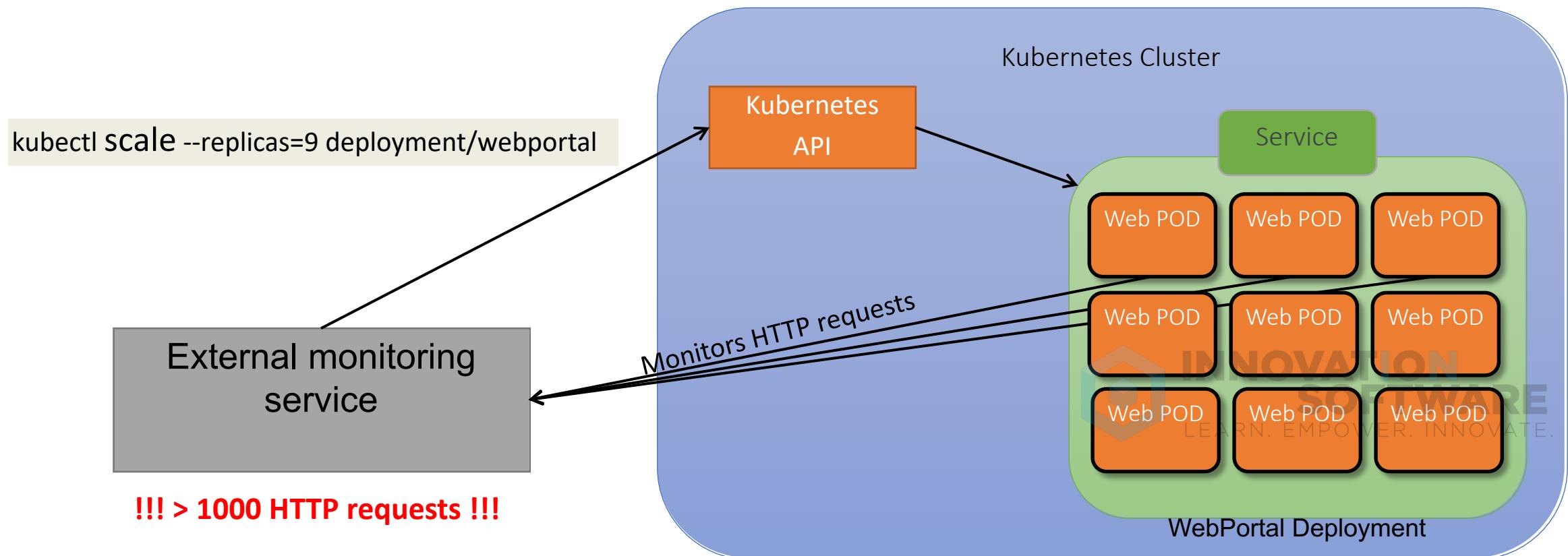
# Application Scaling Strategies

## *Externally-driven auto-scaling*

- Kubernetes API can be used to change Deployment replica counts remotely
- Any monitoring application that can send REST calls (e.g. Nagios) could be used to drive automatic scaling externally
- Very flexible approach, but more complex to implement
- **Needs robust testing before going into production**

# Application Scaling Strategies

*Auto-scaling driven by external system*

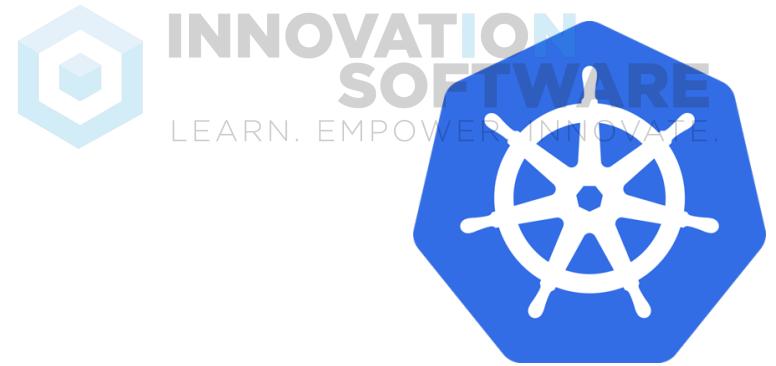




INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.

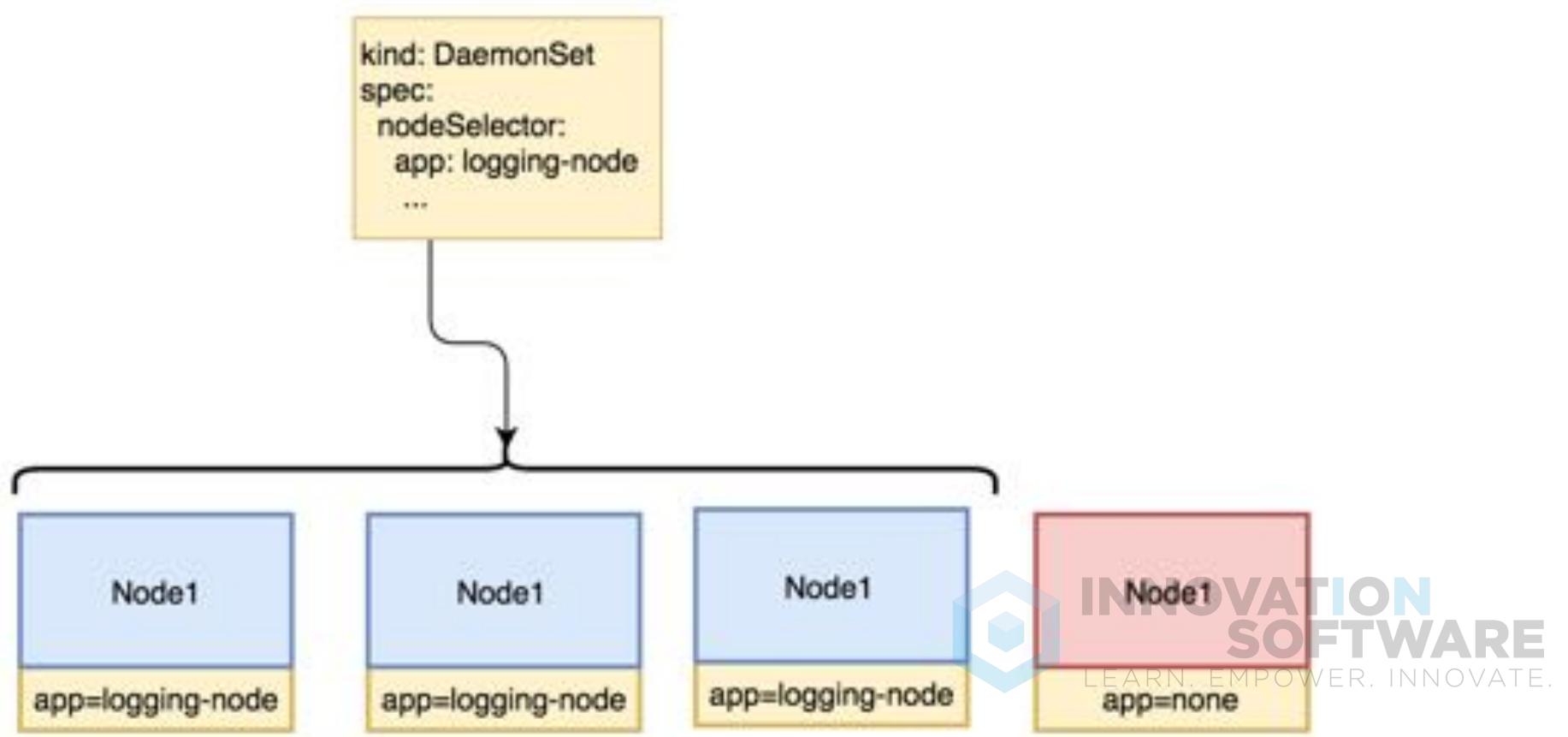
# Lab: Deployments



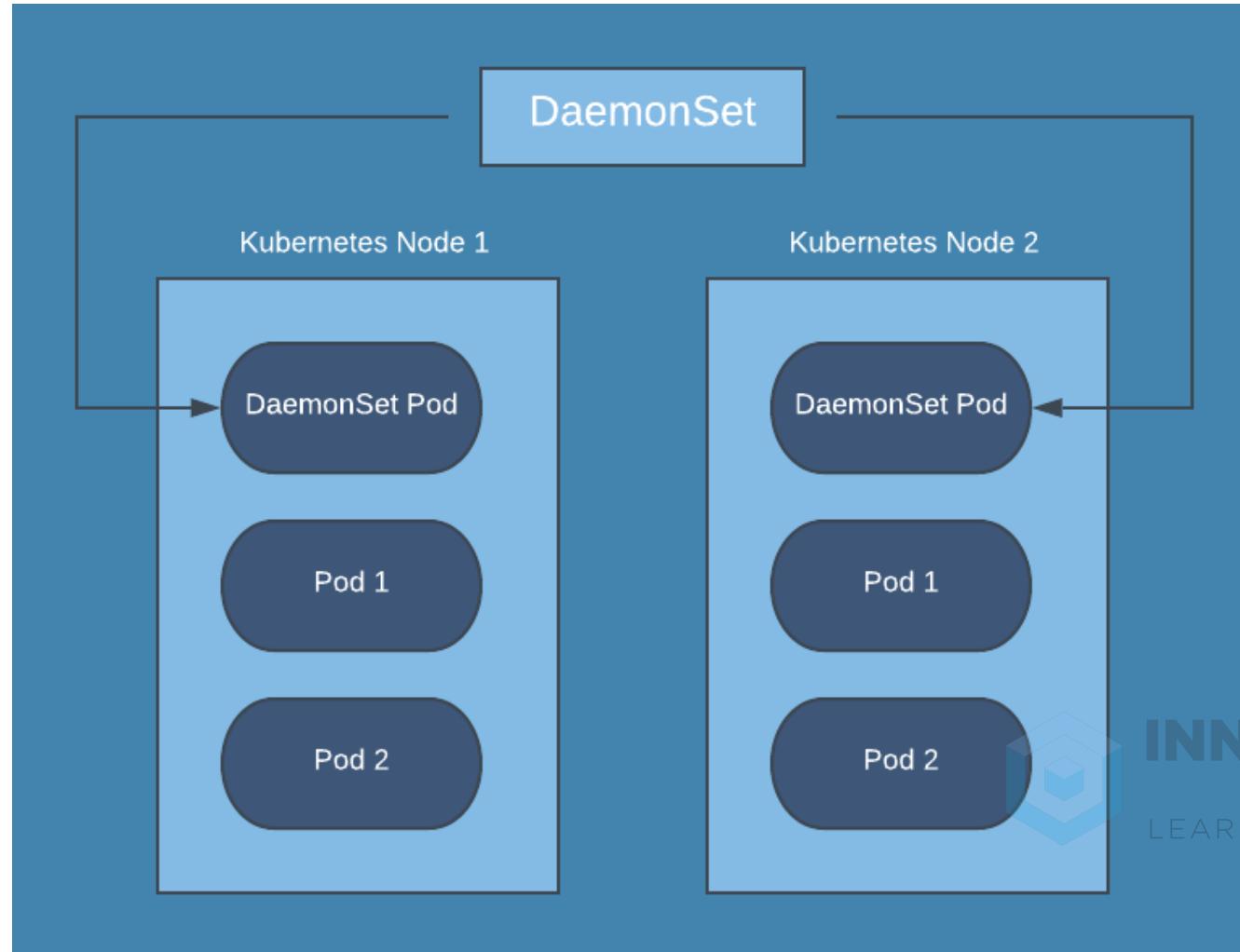
# DaemonSets



# DaemonSets



# DaemonSets



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# DaemonSets

## *Configure DaemonSet*

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: logging
spec:
  selector:
    matchLabels:
      app: logging-app
  template:
    metadata:
      labels:
        app: logging-app
  spec:
    containers:
      - name: webserver
        image: nginx
        ports:
          - containerPort: 80
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# DaemonSets

## *Configure DaemonSet*

```
....  
template:  
  metadata:  
    labels:  
      app: logging-app  
spec:  
  nodeSelector:  
    type: prod  
  containers:  
    - name: webserver  
      image: nginx  
      ports:  
        - containerPort: 80
```

- **nodeSelector:** which node(s) the DaemonSet runs on.



# Jobs & CronJobs



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.



# Non-Parallel Job Config

*Configure non-parallel job*

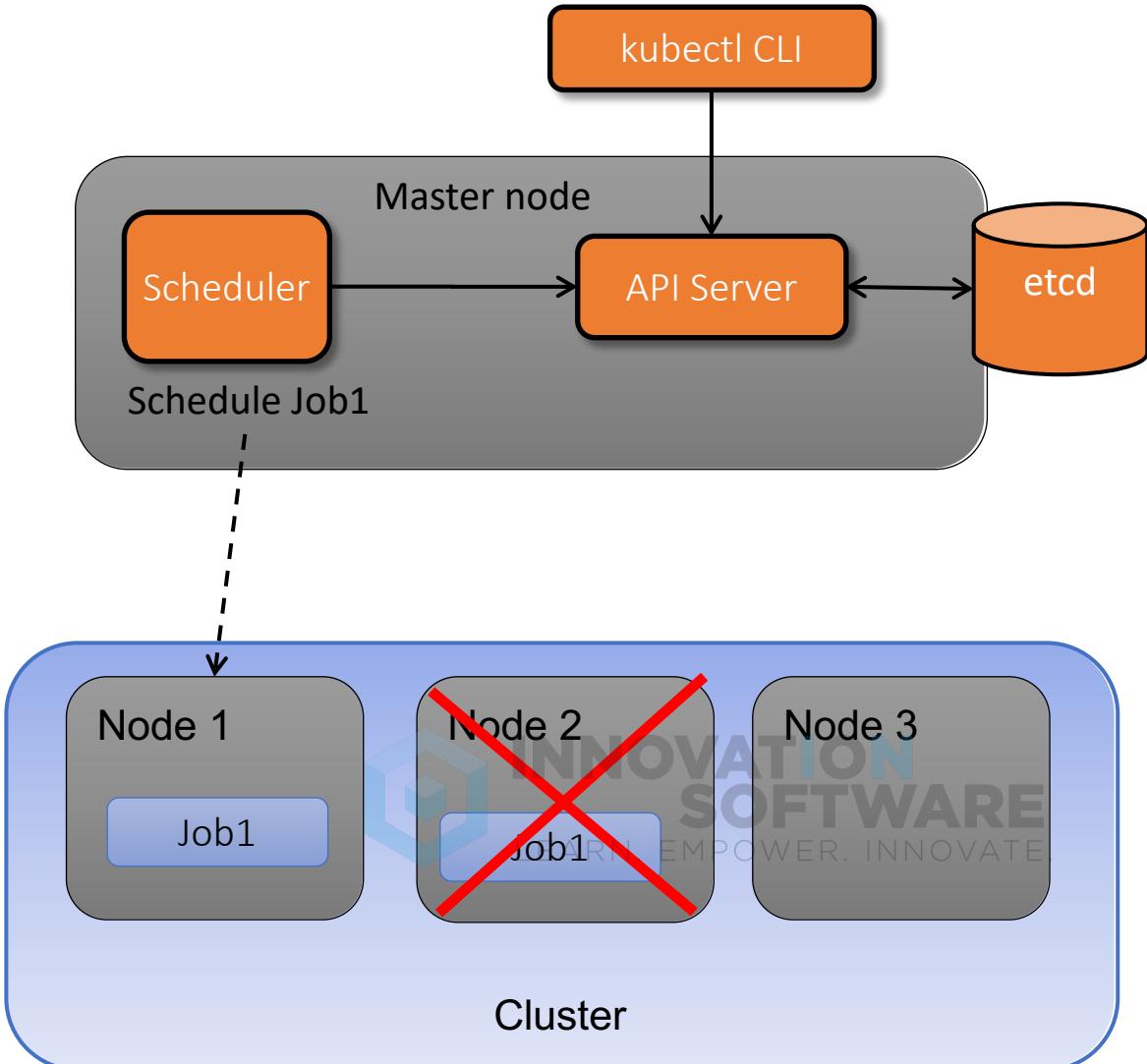
```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
    backoffLimit: 4
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Non-Parallel Job

- Unlike other Kubernetes controllers, Jobs manage a task to its completion rather than to an open-ended desired state. In a way, the desired state of a job is its completion.
- Jobs can run in non-parallel or parallel mode.



# Parallel Job Config

*Configure parallel job*

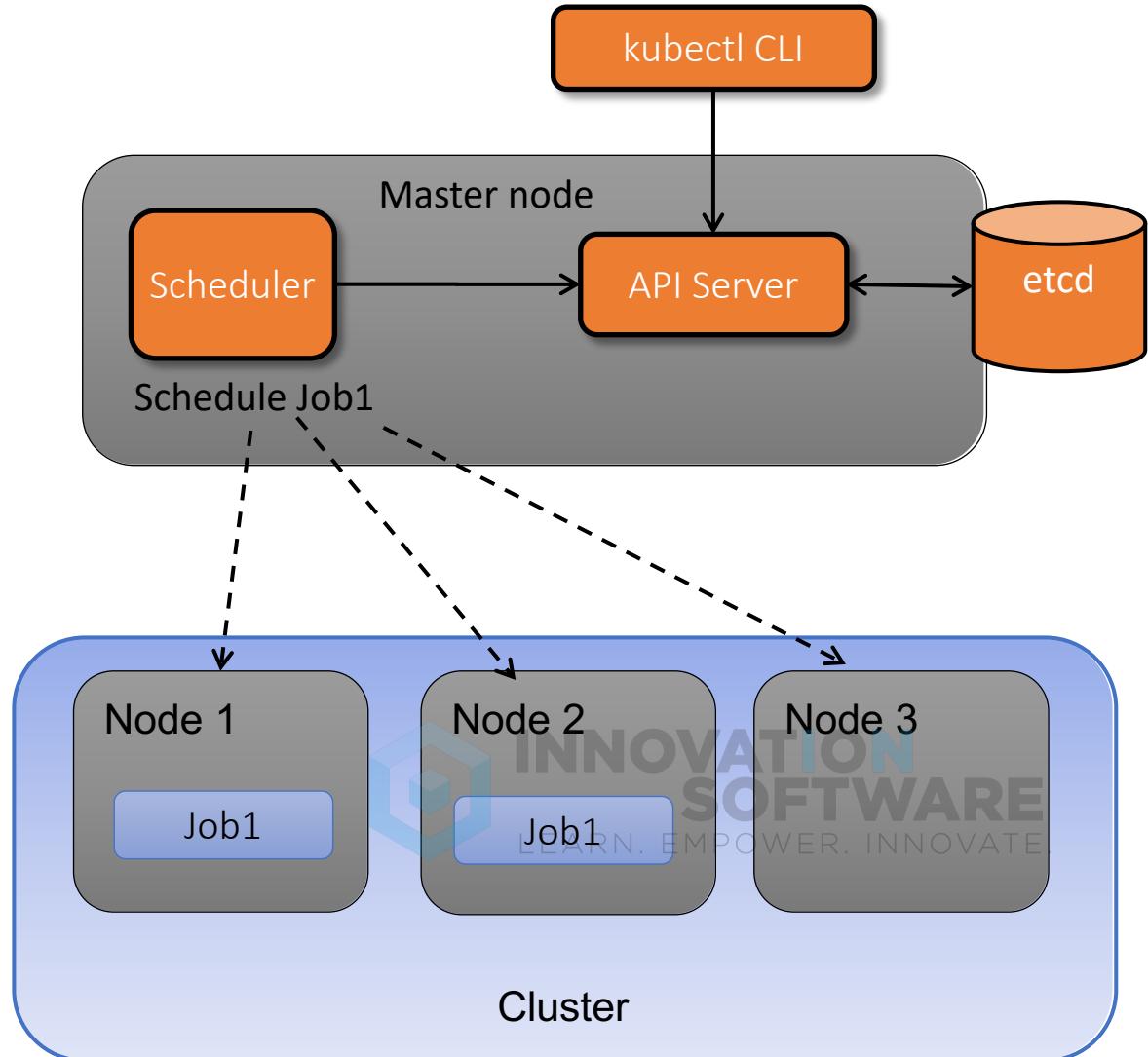
```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 3
  completions: 2
  template:
    spec:
      containers:
        - name: pi
          image: perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Parallel Job

- Unlike other Kubernetes controllers, Jobs manage a task to its completion rather than to an open-ended desired state. In a way, the desired state of a job is its completion.
- Jobs can run in non-parallel or parallel mode.



# Parallel Queue Job Config

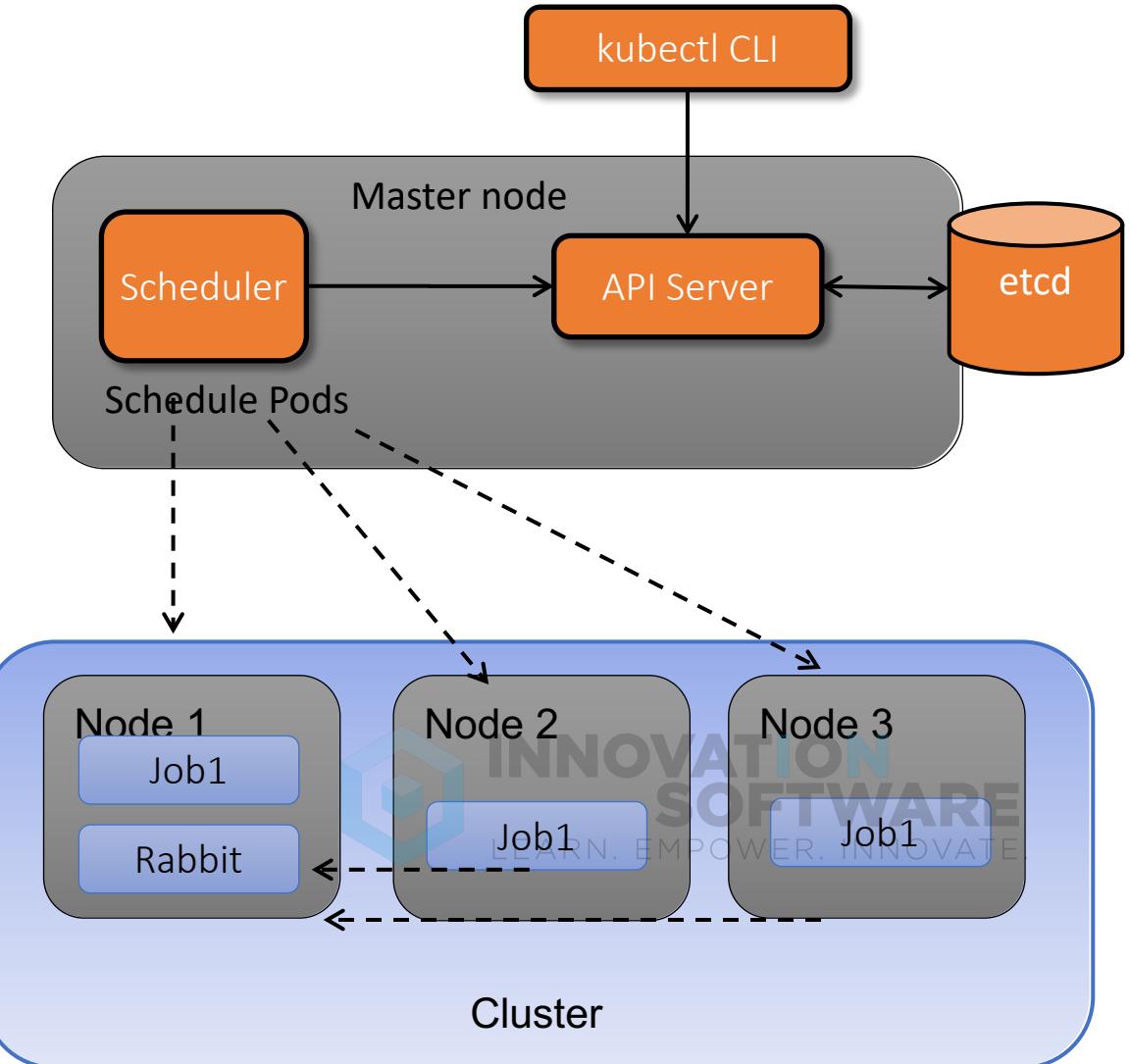
*Configure parallel queue job*

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-queue-1
spec:
  parallelism: 3
  completions: 8
  template:
    metadata:
      name: job-queue-1
    spec:
      containers:
        - name: c
          image: gcr.io/project/job-queue-1
          env:
            - name: BROKER_URL
              value amqp://guest:guest@rabbitmq-service:5672
            - name: QUEUE
              value: job1
      restartPolicy: Never
```



# Parallel Queue Job

- Use this when you know the number of messages in the queue.
- Consume messages from a queue until *completions* is reached.



# Parallel Queue Job Config

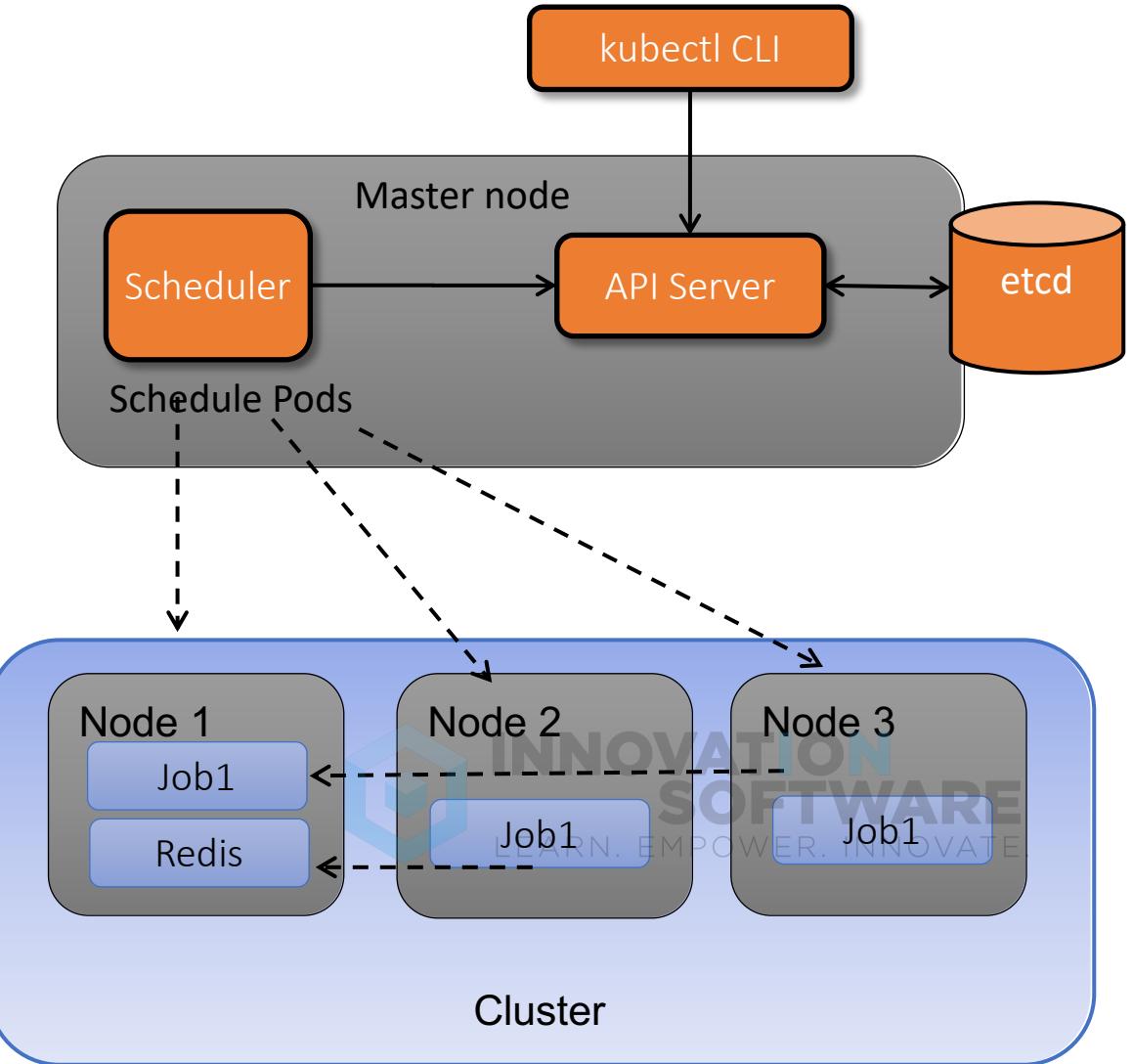
*Configure parallel queue job*

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-queue-2
spec:
  parallelism: 3
  template:
    metadata:
      name: job-queue-2
    spec:
      containers:
      - name: c
        image: gcr.io/project/job-queue-2
  restartPolicy: Never
```



# Parallel Queue Job

- Use this when you have an undefined number of tasks that need to be done.
- Consume messages from a queue until it is empty.



# CronJob

## *Configure cronjob*

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
  restartPolicy: OnFailure
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# CronJob Scheduling

## *Configure cronjob*

```
# └───────── minute (0 - 59)
#   └──────── hour (0 - 23)
#     └────────── day of the month (1 - 31)
#       └────────── month (1 - 12)
#         └─────────── day of the week (0 - 6) (Sunday to Saturday;
#           7 is also Sunday on some systems)
#
#
# * * * * *
```

**Entry**  
@yearly (or @annually)  
@monthly  
@weekly  
@daily (or @midnight)  
@hourly

**Description**  
Run once a year at midnight of 1 January  
Run first day of month at midnight  
Run once a week at midnight on Sunday morning  
Run once a day at midnight  
Run once an hour at the beginning of the hour

**Equivalent to**  
0 0 1 1 \*  
0 0 1 \* \*  
0 0 \* \* 0  
0 0 \* \* \*  
0 \* \* \* \*



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

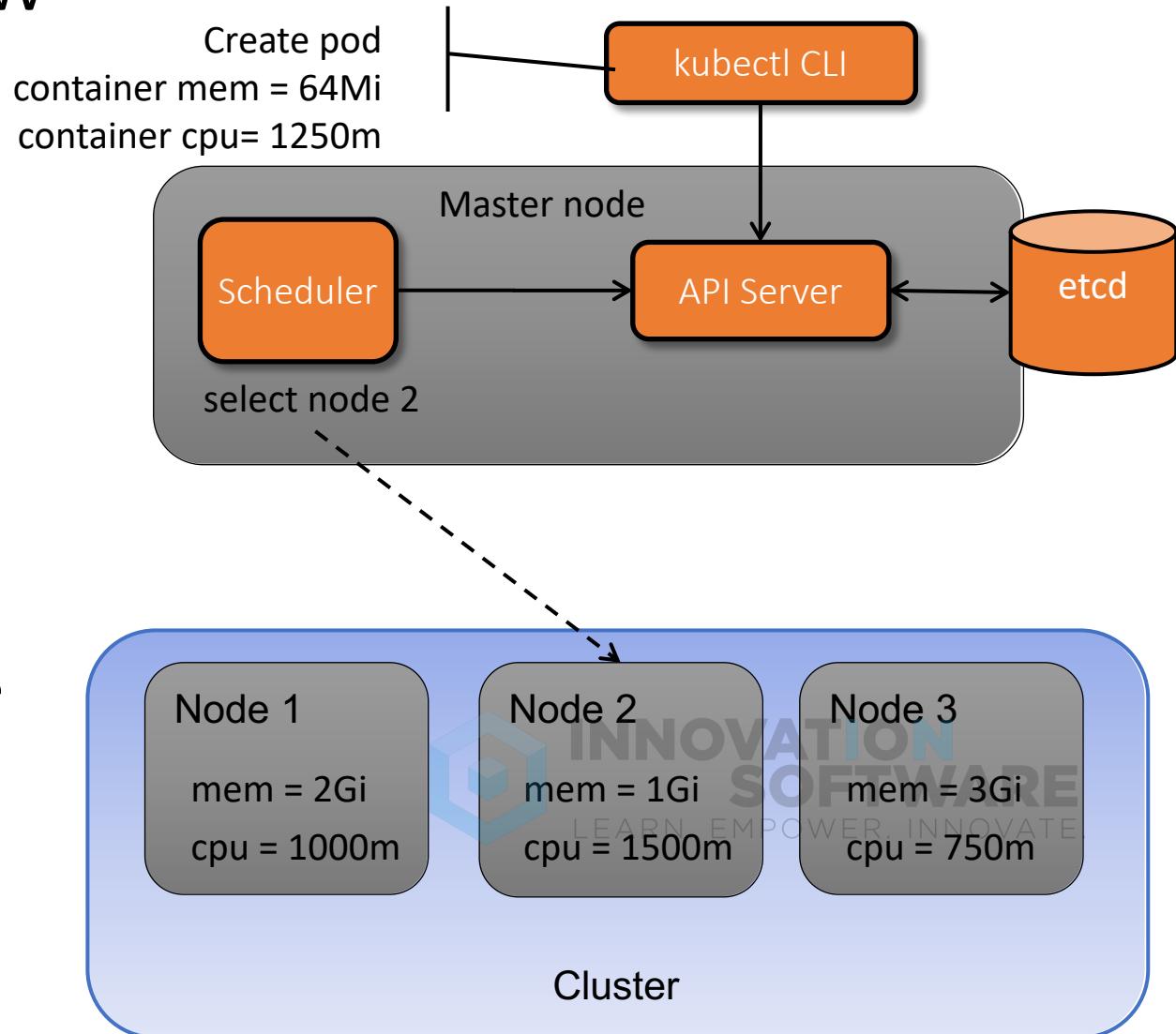
# Kubernetes POD Scheduling



INOVATION  
LEARN. EMPOWER. INNOVATE.

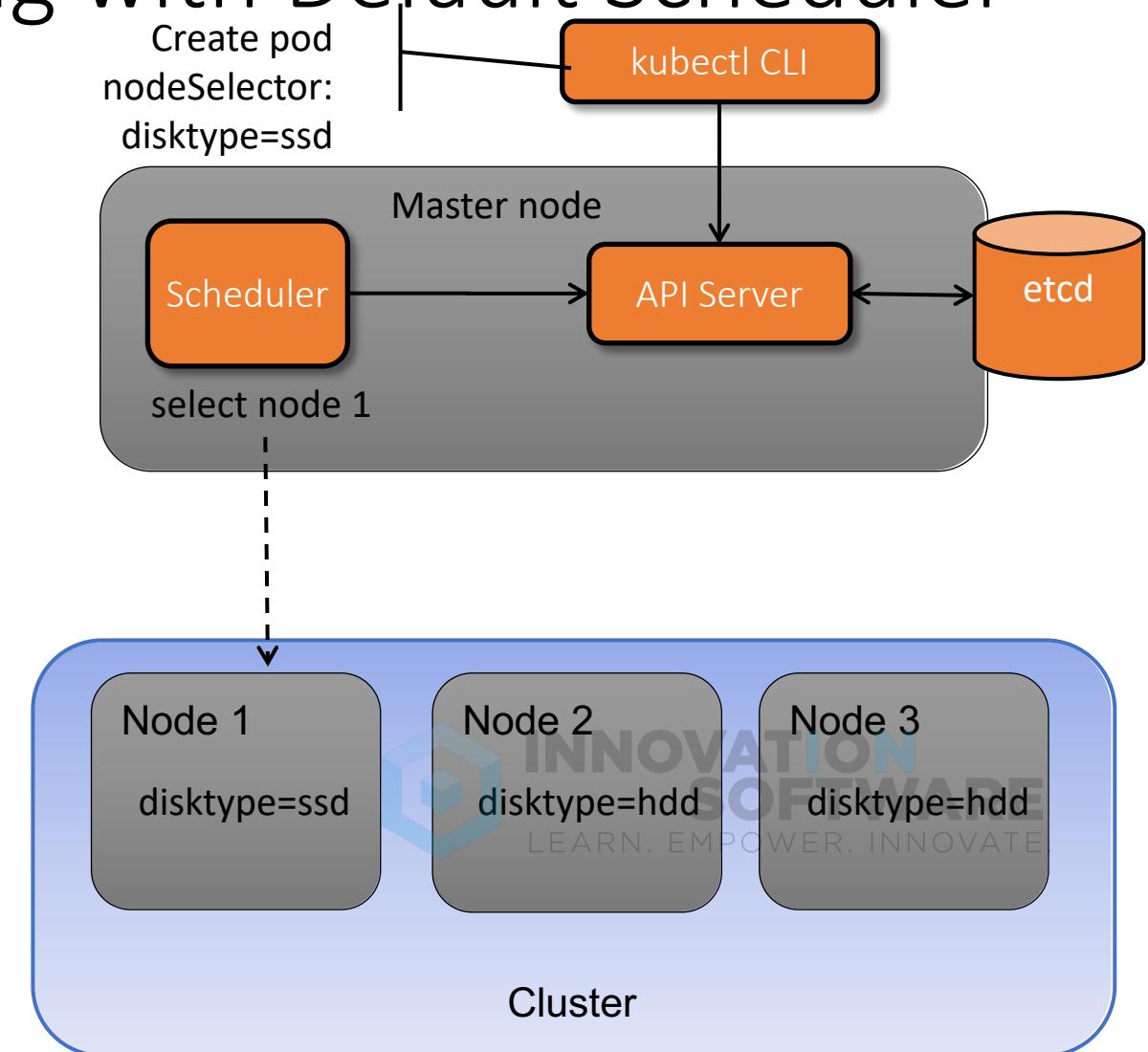
# Pod Scheduling Overview

- kube-scheduler on the master node assigns new pod requests to appropriate worker nodes
- Default scheduler takes account of
  - Available node CPU/RAM
  - Resource requests from new pod – sum of resource requests of pod containers
- Default scheduler will automatically
  - Schedule pod on node with sufficient free resources
  - Spread pods across nodes in the cluster
- Can specify custom schedulers for pods



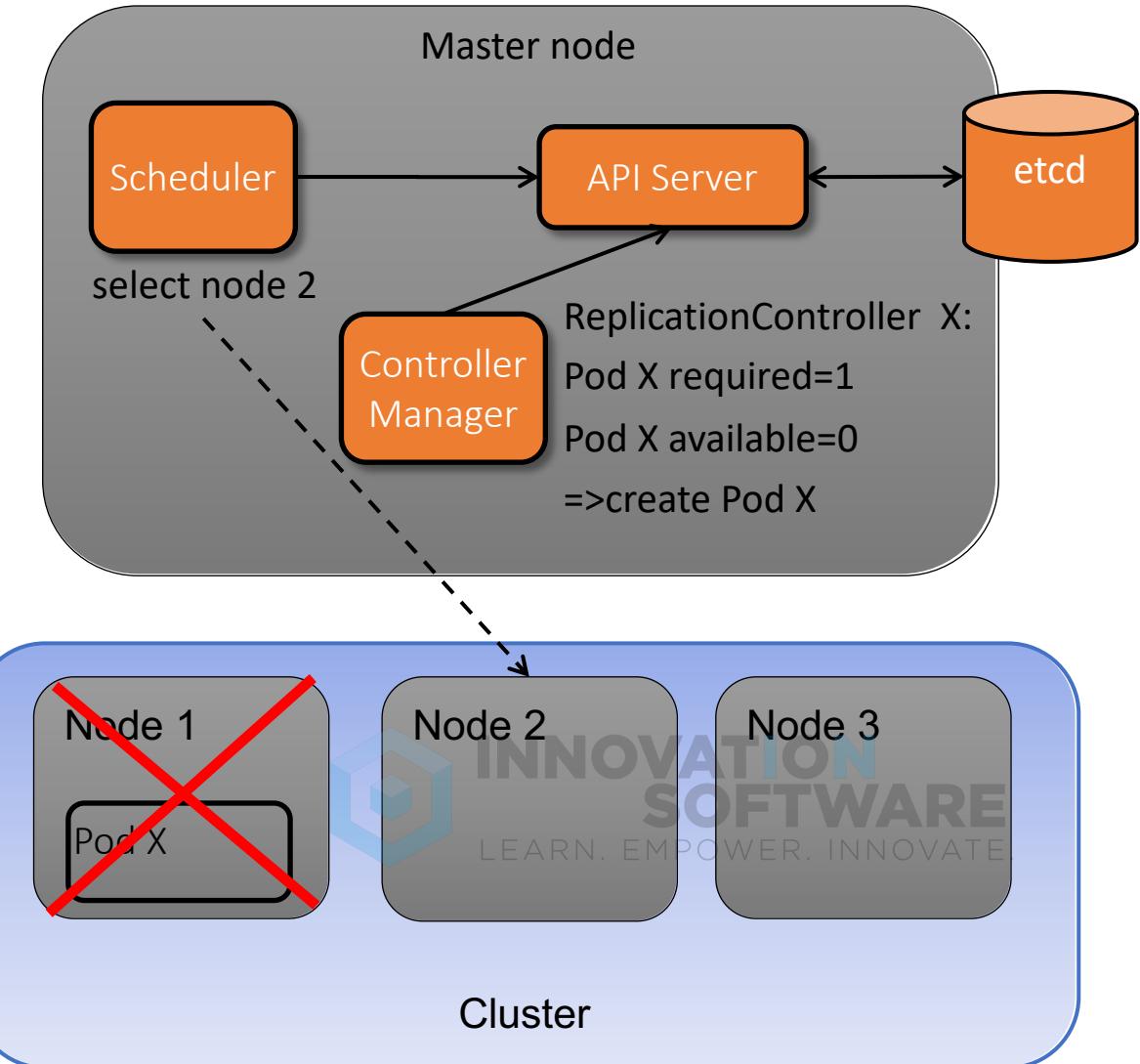
# Controlling Pod Scheduling with Default Scheduler

- Nodes carry labels indicating topology and other resource notes
- Users can require pods to be scheduled on nodes with specific label(s) via a nodeSelector in container spec



# Scheduling Pod Re-creation Driven by Control Loops

- kube-scheduler performs same node selection operation when new pod created due to e.g. node loss
- kube-controller-manager runs controllers like ReplicationController managing number of pod instances available
- kube-controller-manager will initiate request for new pods as needed, which will be scheduled by kube-scheduler per pod/container spec



# Kubernetes Scheduling

## Node Selector

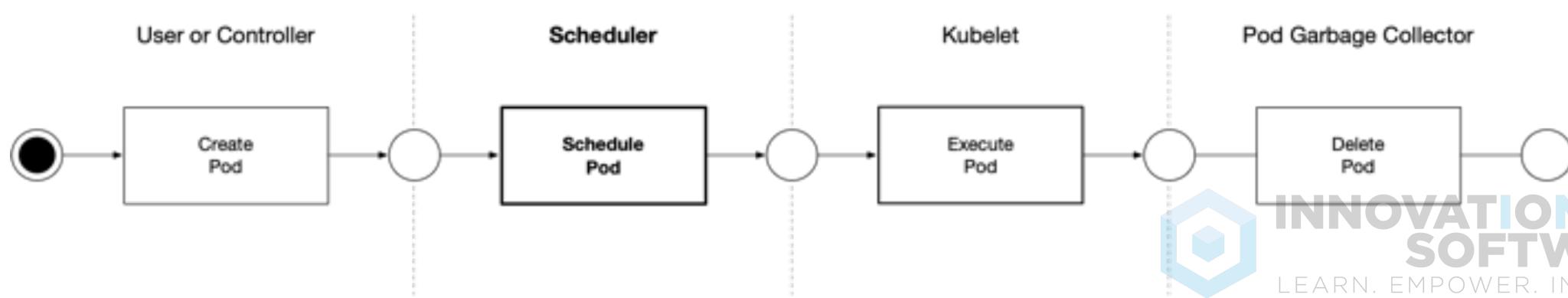
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  container:
  - name: nginx
    image: nginx
    nodeSelector:
      disktype: ssd
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

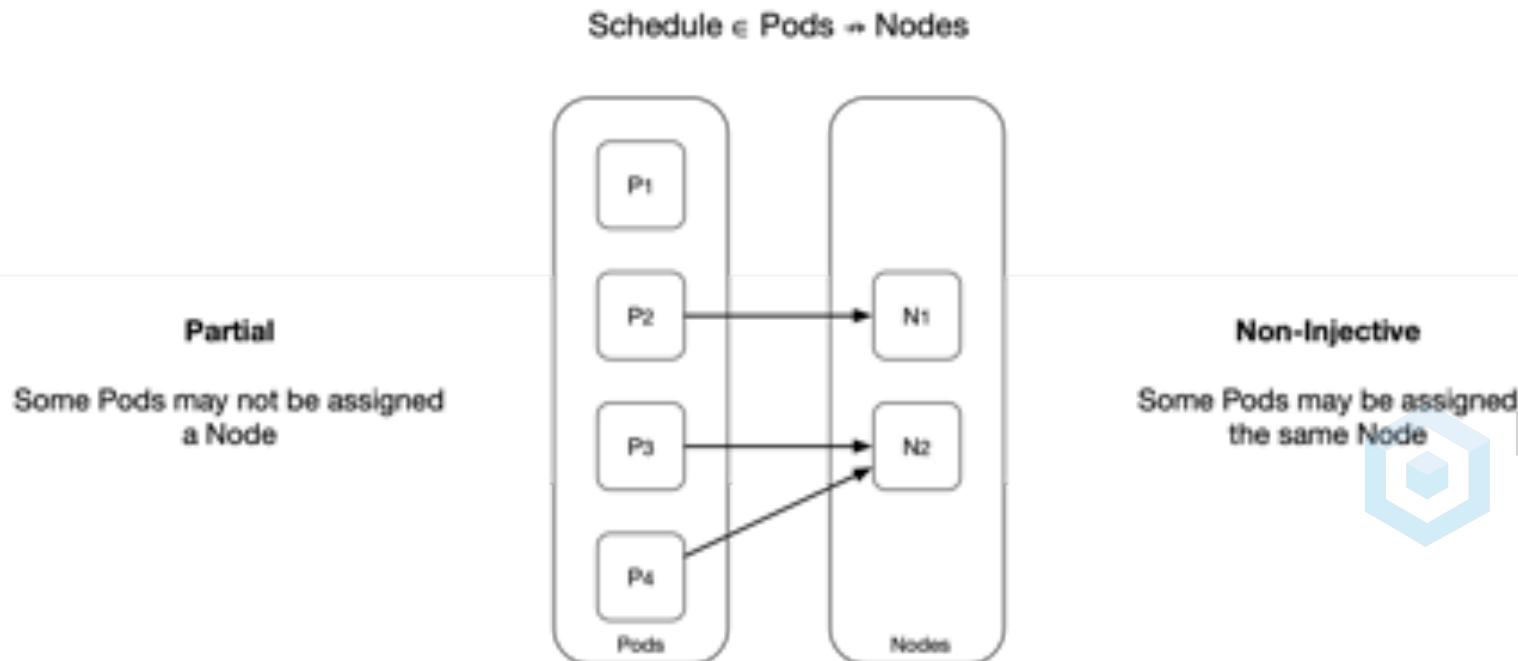
# Scheduling flow

- **Scheduler (kube-scheduler)**: selects nodes for newly created pods to run on, this is called a "placement"
  - Placement: a partial, non-injective assignment of a set of Pods to a set of Nodes.



# Scheduler details

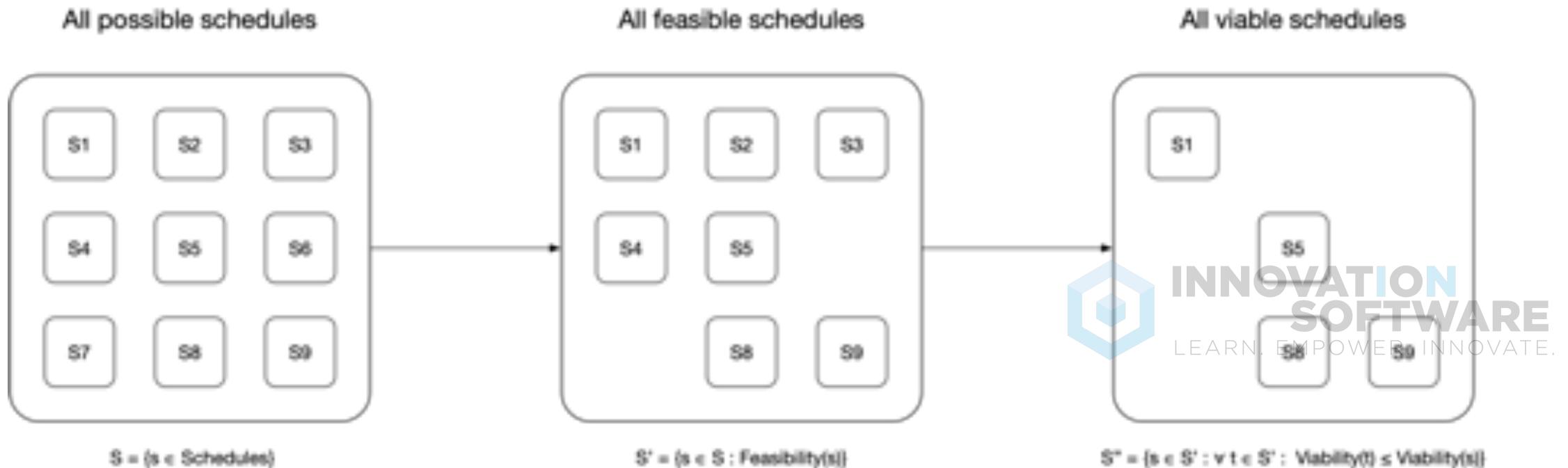
- **Scheduler (kube-scheduler)**: selects nodes for newly created pods to run on, this is called a "placement"
  - Placement: a partial, non-injective assignment of a set of Pods to a set of Nodes.



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

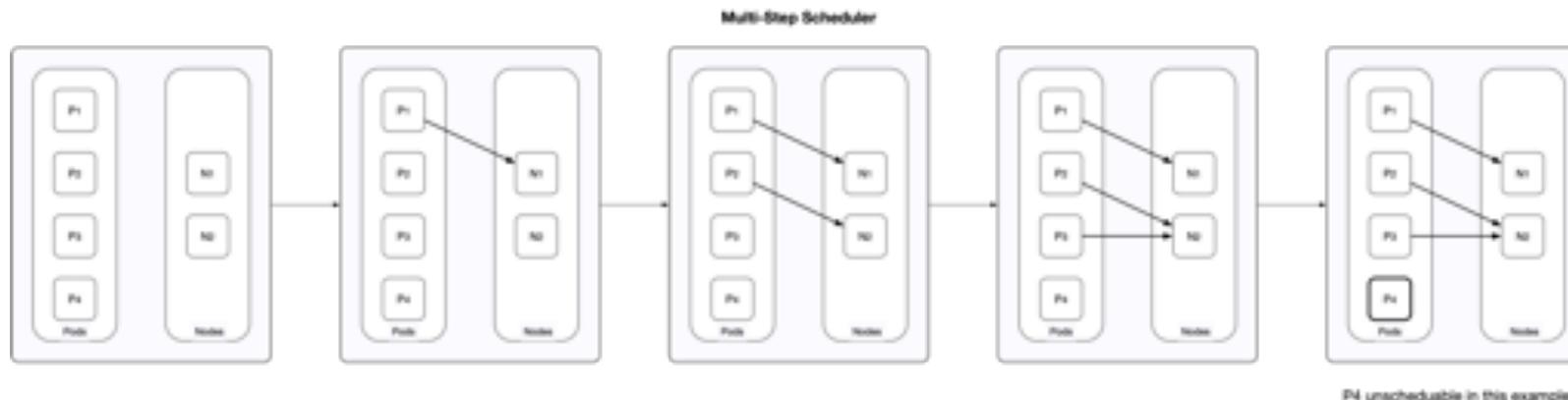
# Scheduler decisions

- **Scheduler (kube-scheduler)**: selects nodes for newly created pods to run on, this is called a "placement"
  - Placement: a partial, non-injective assignment of a set of Pods to a set of Nodes.



# Scheduler decisions

- **Multi-step scheduler**
  - local optimum instead of global optimum



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Scheduler preemption



- **Preemption**

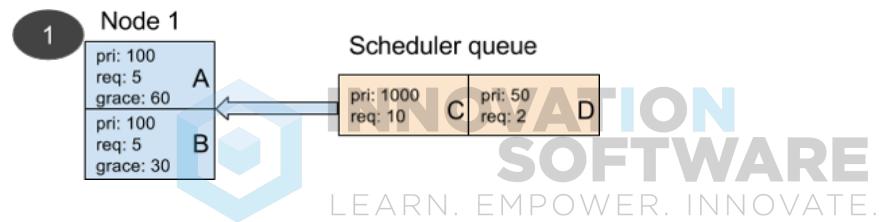
- Lower priority Pods are destroyed to free up resources for higher priority Pods.

# Scheduler preemption



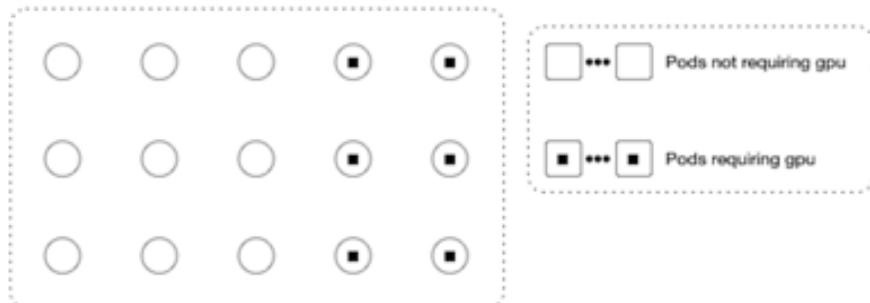
- **Example**

- 1 node in the cluster with capacity 10 units
- 2 pods (A,B) running on the node with priority 100 and each using 5 units
- Scheduler has 2 pods (C,D) in queue. Pod C needs 10 units and priority is 1000. Pod D needs 2 units and it's priority is 50
- Scheduler determines that Pod C has highest priority and destroys (A,B) so it can schedule.



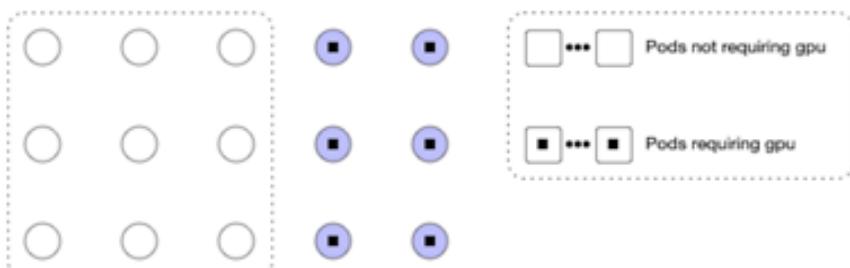
# Scheduler placements

## Initial State



- Non-GPU Pods eligible to run on any Node
- GPU Pods eligible to run on any Node

## With Node Taints



- Non-GPU Pods eligible to run on any Non-GPU Node
- GPU Pods eligible to run on any Non-GPU Node

- **Case Study**

- 9 nodes without GPUs
- 6 nodes with GPUs
- We need to schedule Pods that require GPUs only on nodes with GPUs and Pods that do not require GPUs on nodes without GPUs.

- **Node Taints**

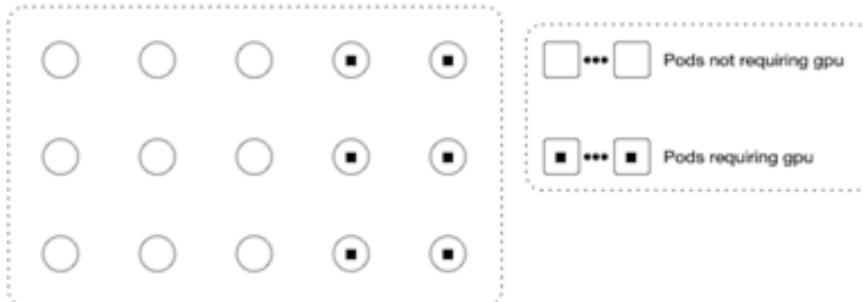
- An option is to use taints which will allow you to stop Pods from being deployed on certain nodes.



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

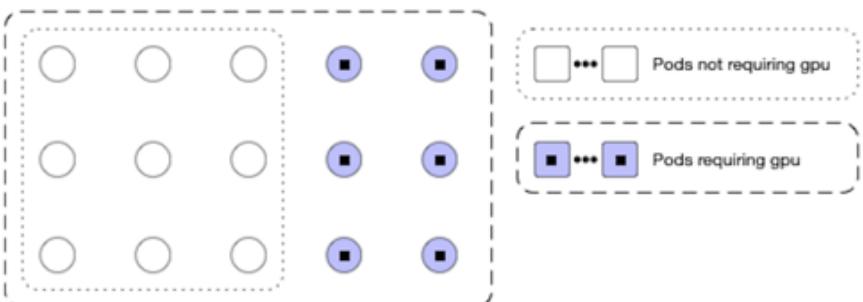
# Scheduler placements

## Initial State



- Non-GPU Pods eligible to run on any Node
- GPU Pods eligible to run on any Node

## With Pod Tolerations



- Non-GPU Pods eligible to run on any Non-GPU Node
- GPU Pods eligible to run on any Node

- **Case Study**

- 9 nodes without GPUs
- 6 nodes with GPUs
- We need to schedule Pods that require GPUs only on nodes with GPUs and Pods that do not require GPUs on nodes without GPUs.

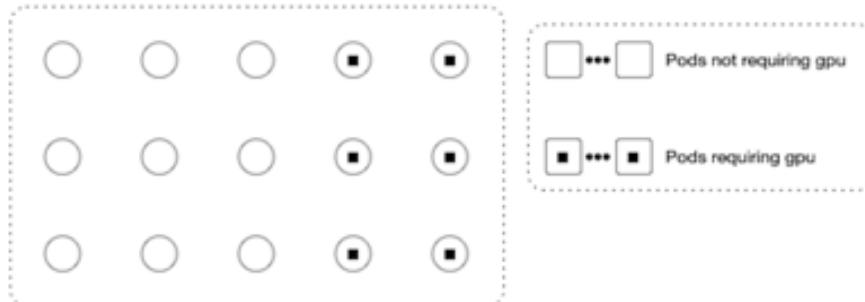
- **Tolerations**

- Tolerations allow us to say specific Pods can run on tainted nodes.



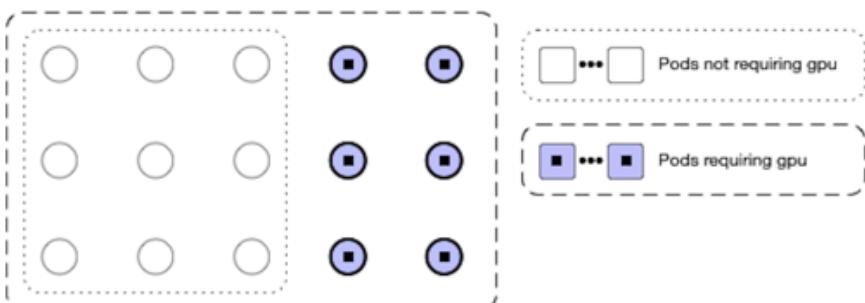
# Scheduler placements

## Initial State



- Non-GPU Pods eligible to run on any Node
- GPU Pods eligible to run on any Node

## With Node Labels



- Non-GPU Pods eligible to run on any Non-GPU Node
- GPU Pods eligible to run on any Node

- **Case Study**

- 9 nodes without GPUs
- 6 nodes with GPUs
- We need to schedule Pods that require GPUs only on nodes with GPUs and Pods that do not require GPUs on nodes without GPUs.

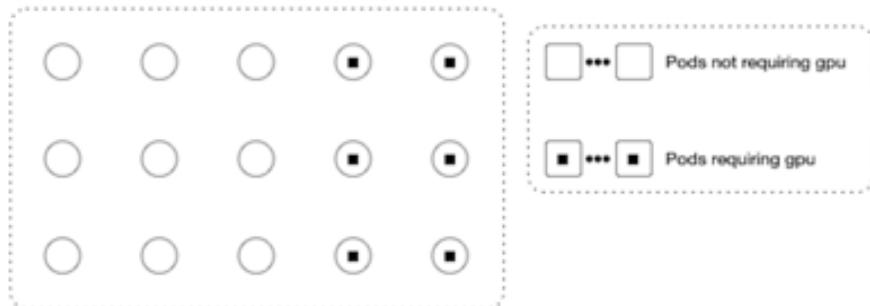
- **Node labels**

- Labels tell Pods which nodes they should be scheduled on.



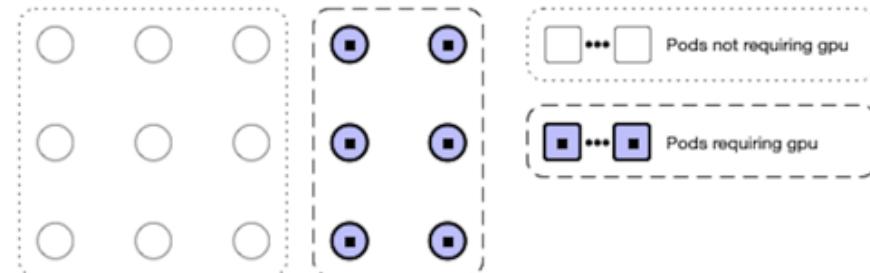
# Scheduler placements

## Initial State



- Non-GPU Pods eligible to run on any Node
- GPU Pods eligible to run on any Node

## With Node Affinity



- Non-GPU Pods eligible to run on any Non-GPU Node
- GPU Pods eligible to run on any GPU Node

- **Case Study**

- 9 nodes without GPUs
- 6 nodes with GPUs
- We need to schedule Pods that require GPUs only on nodes with GPUs and Pods that do not require GPUs on nodes without GPUs.

- **Node affinity**

- Similar to node labels but more expressive
  - In/NotIn/Exists/DoesNotExist/Lt,Gt
  - Required or preferred
  - Schedule based on Pods running on other nodes.

# Built-in Node Labels

- kubernetes.io/hostname
- failure-domain.beta.kubernetes.io/zone
- failure-domain.beta.kubernetes.io/region
- beta.kubernetes.io/instance-type
- beta.kubernetes.io/os
- beta.kubernetes.io/arch

# Kubernetes Advanced POD Scheduling



LEARN - EMPOWER - INNOVATE

# Kubernetes Scheduling

- Node Selector
- Node Affinity
- Pod Affinity
- Pod Anti Affinity
- 3rd party schedulers



# Kubernetes Scheduling

- K8s 1.6 adds more advanced scheduling options, including
  - Node affinity/anti-affinity
    - generalizing the nodeSelector feature
  - Node taints
    - prevent scheduling of pods to nodes unless pod ‘tolerates’ the taint
  - Pod affinity/anti-affinity
    - control relative placement of pods

# Node Affinity

The affinity/anti-affinity feature greatly expands the types of constraints you can express.

The key enhancements are:

- More expressive language (not just “AND of exact match”)
- Rule is “soft”/“preference” so if it doesn’t match Pods are still scheduled.

# Kubernetes Scheduling

## Node Affinity

```
affinity:  
  nodeAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
      nodeSelectorTerms:  
        - matchExpressions:  
          - key: "failure-domain.beta.kubernetes.io/zone"  
            operator: In  
            values: ["us-central1-a"]
```

# Kubernetes Scheduling

## Node Affinity

```
affinity:  
  nodeAffinity:  
    preferredDuringSchedulingIgnoredDuringExecution:  
      nodeSelectorTerms:  
        - matchExpressions:  
          - key: "failure-domain.beta.kubernetes.io/zone"  
            operator: In  
            values: ["us-central1-a"]
```

# Kubernetes Scheduling

Weight example

```
affinity:  
nodeAffinity:  
  preferredDuringSchedulingIgnoredDuringExecution:  
    - weight: 5  
      preference:  
        - matchExpressions:  
          - key: env  
            operator: In  
            values:  
              - dev  
  preferredDuringSchedulingIgnoredDuringExecution:  
    - weight: 1  
      preference:  
        - matchExpressions:  
          - key: team  
            operator: In  
            values:  
              - engineering
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Node Affinity

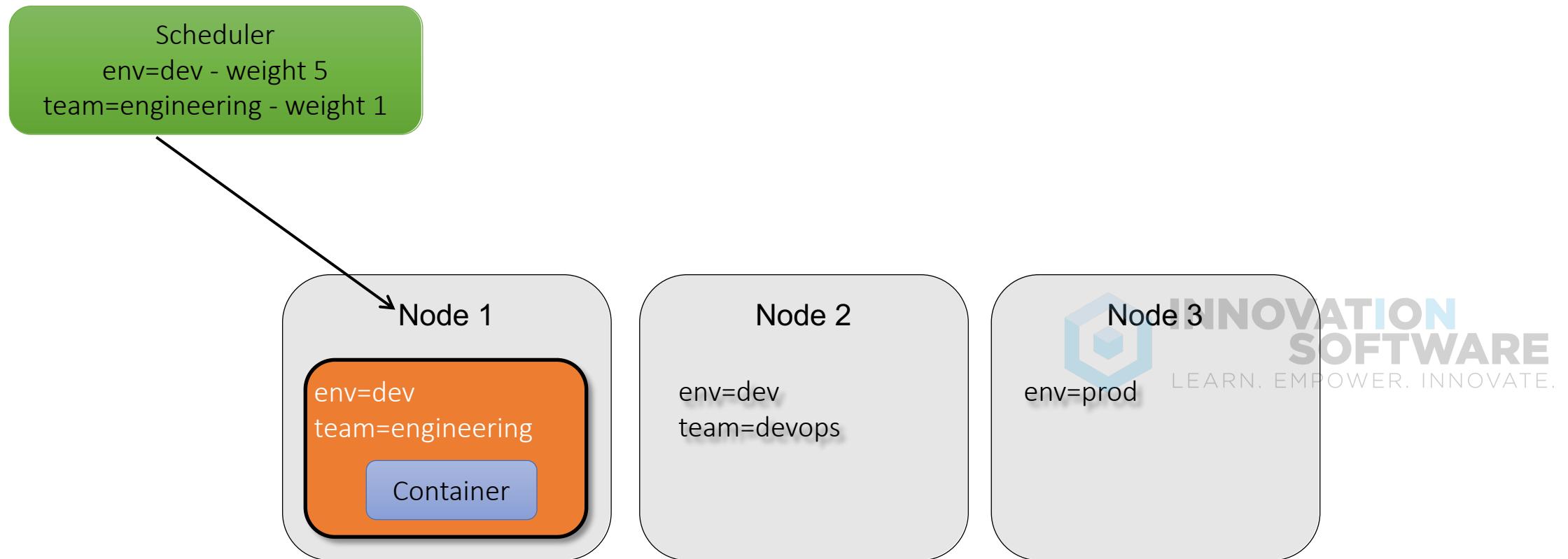
Kubernetes supports "weighted scheduling"

- The higher this weighting the more weight is given to that rule.
- Scoring is done by summarizing weighting per node.
  - 2 rules weights 1 and 5
  - If both match = score of 6
  - If only 1 rule matches = score of 1
- Node with highest total weight score receives Pod.



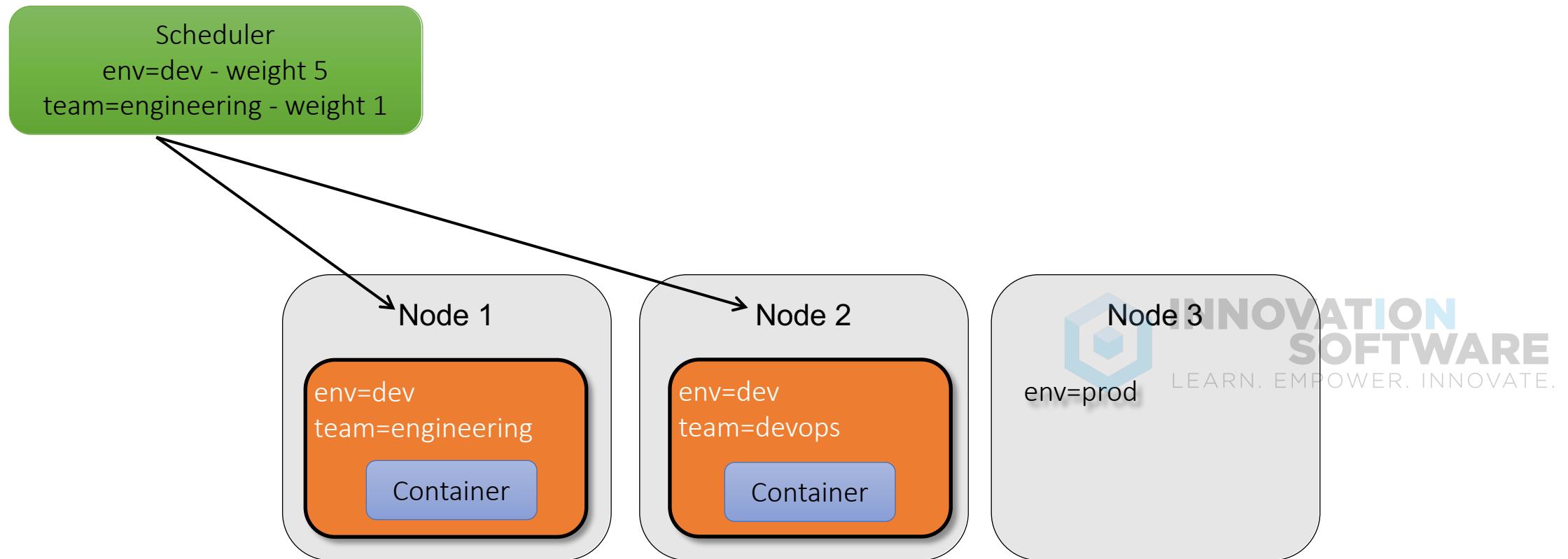
# Node Affinity: Weight

- Schedule Pods onto nodes that have labels
  - env = dev
  - team = engineering



# Node Affinity: Weight

- Schedule Pods onto nodes that have labels
  - env = dev
  - team = engineering



# Kubernetes Scheduling

## Node Anti-Affinity

```
affinity:  
  nodeAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
      nodeSelectorTerms:  
        - matchExpressions:  
          - key: "failure-domain.beta.kubernetes.io/zone"  
            operator: NotIn  
            values: ["us-central1-a"]
```

# Operators

- Valid operators
  - In
  - NotIn
  - Exists
  - DoesNotExist
  - Gt (Greater Than)
  - Lt (Less Than)

# Node Management



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE



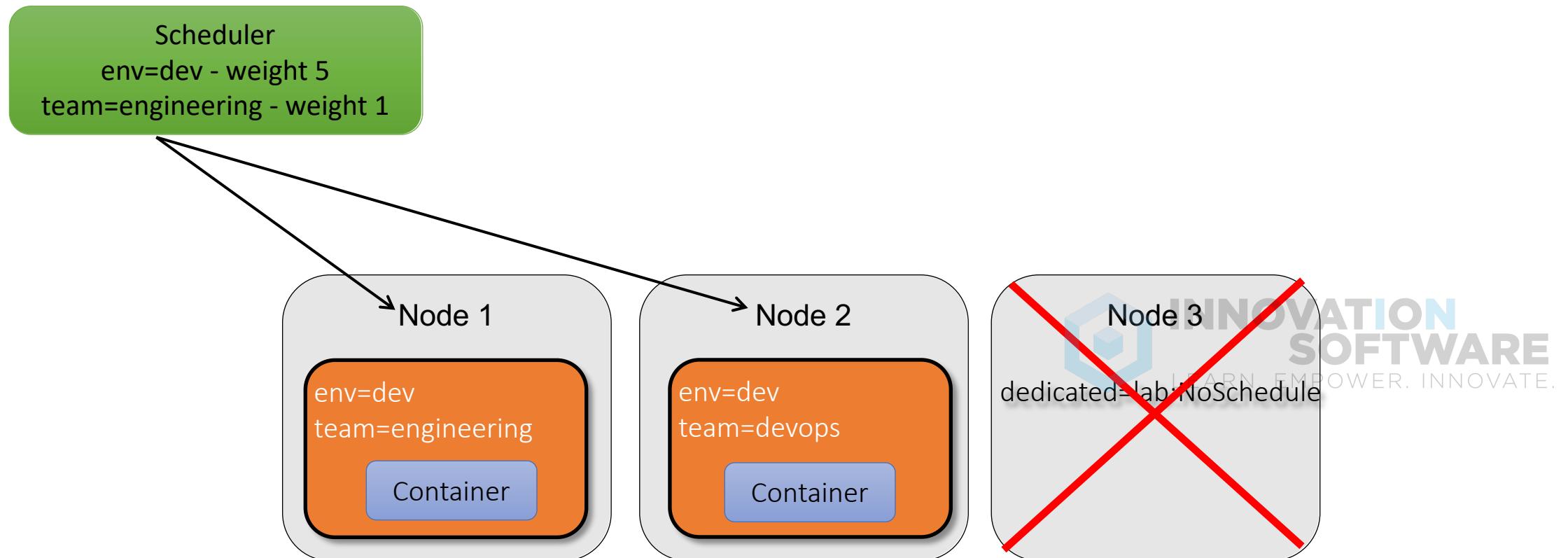
# Node Taints

- Allows you to mark (“taint”) a node
  - No pods can be scheduled onto tainted nodes
  - Useful when all or most PODs should not be scheduled on a node
  - Mark your master node as schedulable only by Kubernetes system components
  - Keep regular pods away from nodes that have special hardware so as to leave room for pods that need the special hardware.



# Node Management: Taint

- Scheduler will not deploy pods to Node 3
  - dedicated=lab:NoSchedule
  - team = engineering



# Node Tolerations

- To allow a POD to be scheduled onto a ‘tainted’ node it must have:

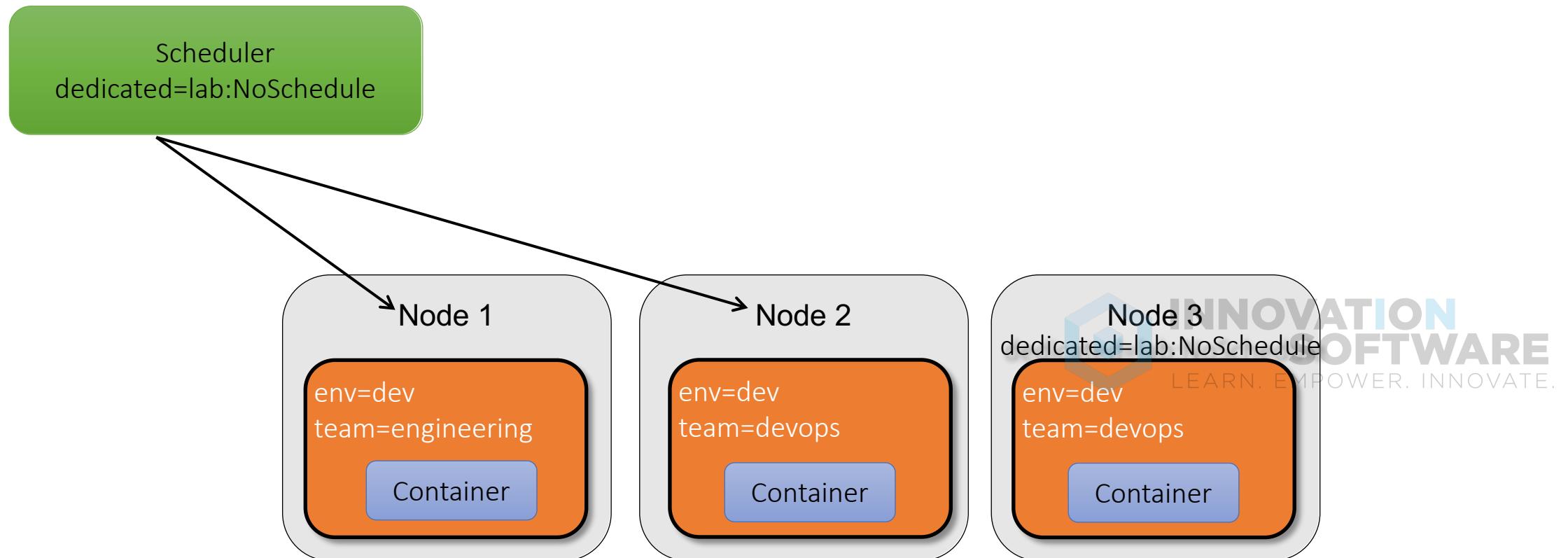
```
tolerations:  
- key: "key"  
  operator: "Equal"  
  value: "value"  
  effect: "NoSchedule"
```



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Node Management: Toleration

- Apply toleration to Node 3
  - dedicated=lab:NoSchedule

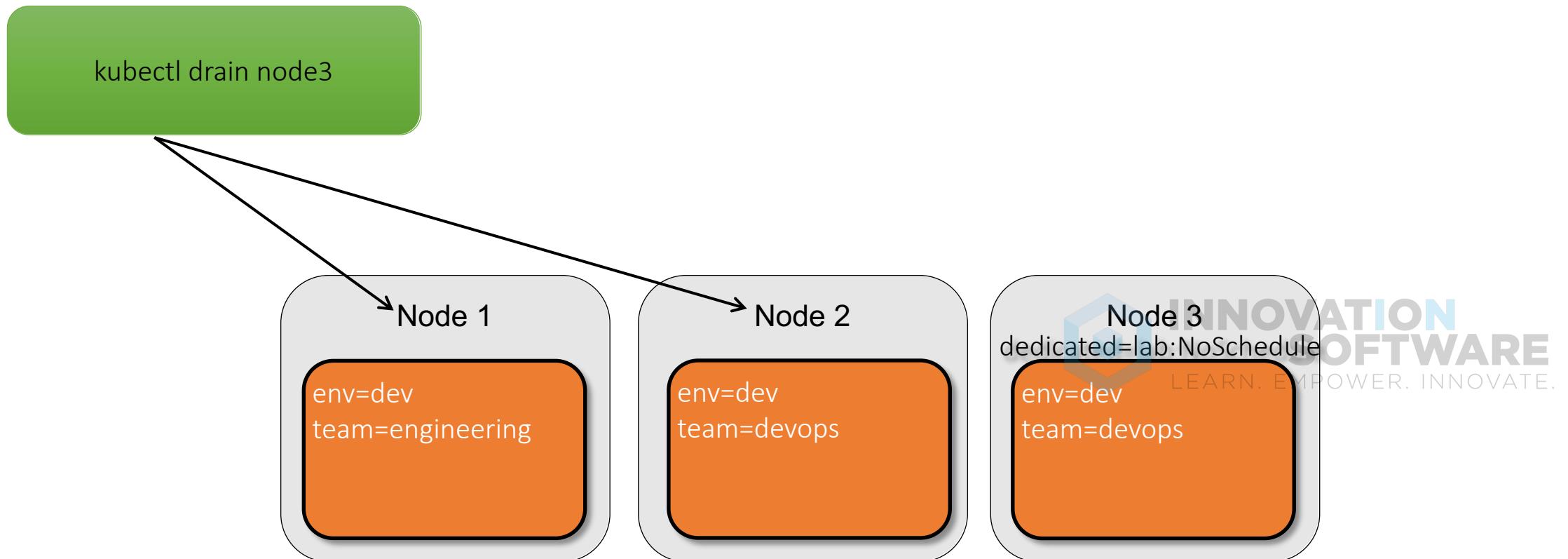


# Drain Node(s)

- Evict all pods from node(s)
  - Pods managed by controller (ReplicaSet, StatefulSet, DaemonSet) are recreated on active nodes if specified in configuration.
  - No pods are scheduled to node(s) while in 'cordon' mode.
  - Respects PodDisruptionBudget
  - Uncordon node when ready to schedule pods to it.

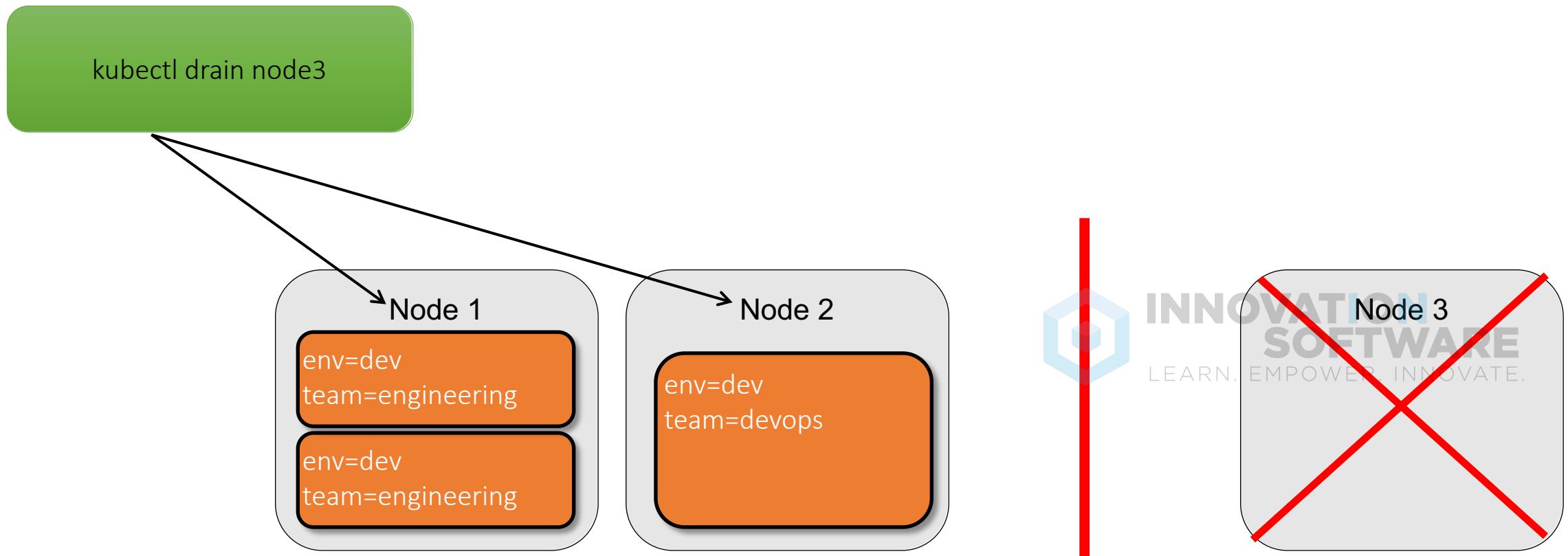
# Node Management: Drain

- Evict pods from Node 3



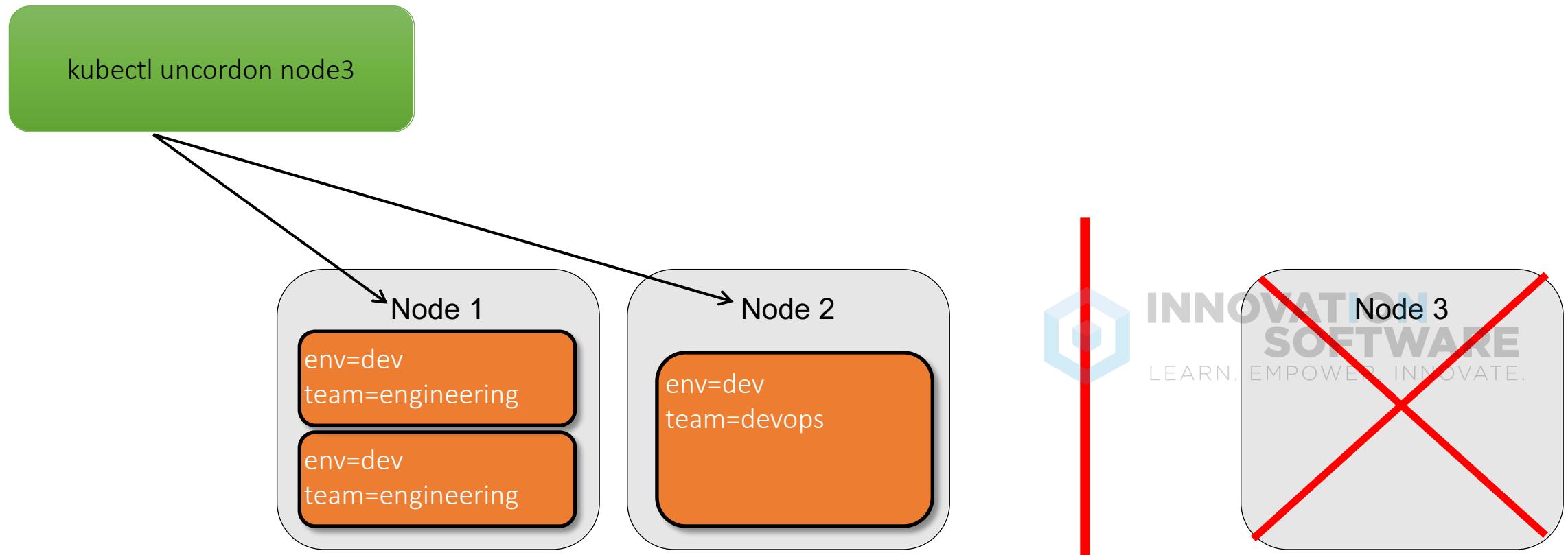
# Node Management: Drain

- Evict pods from Node 3



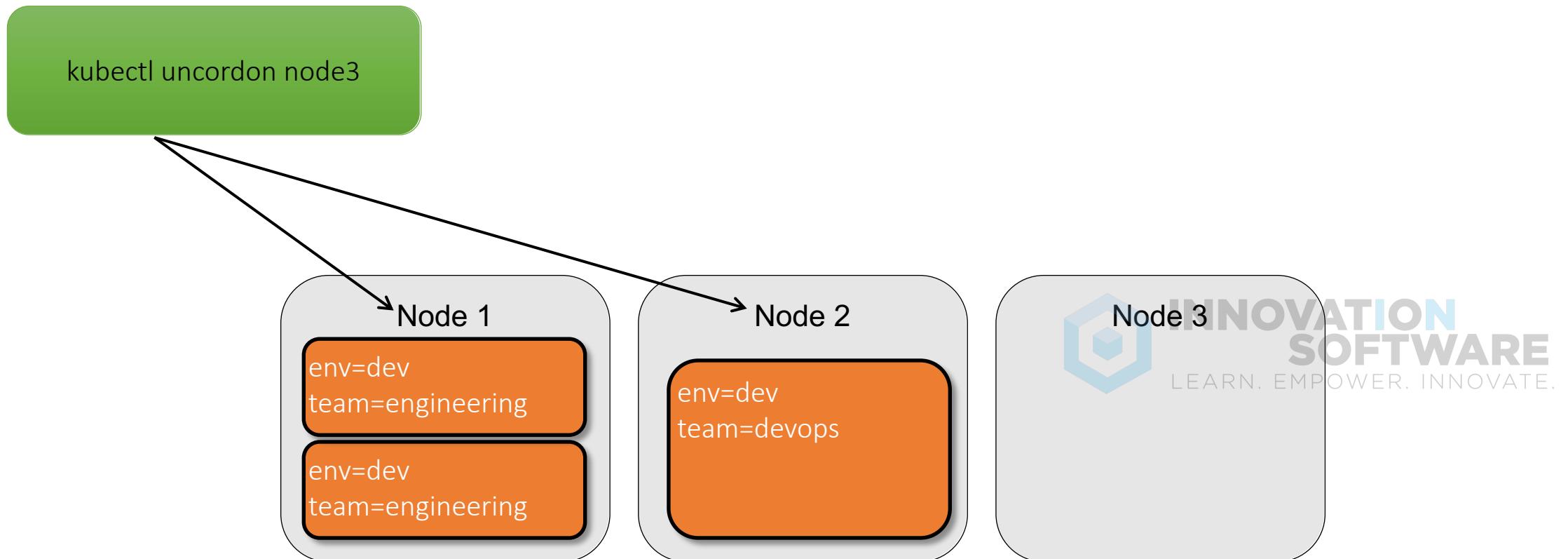
# Node Management: Uncordon

- Return Node 3 to active state



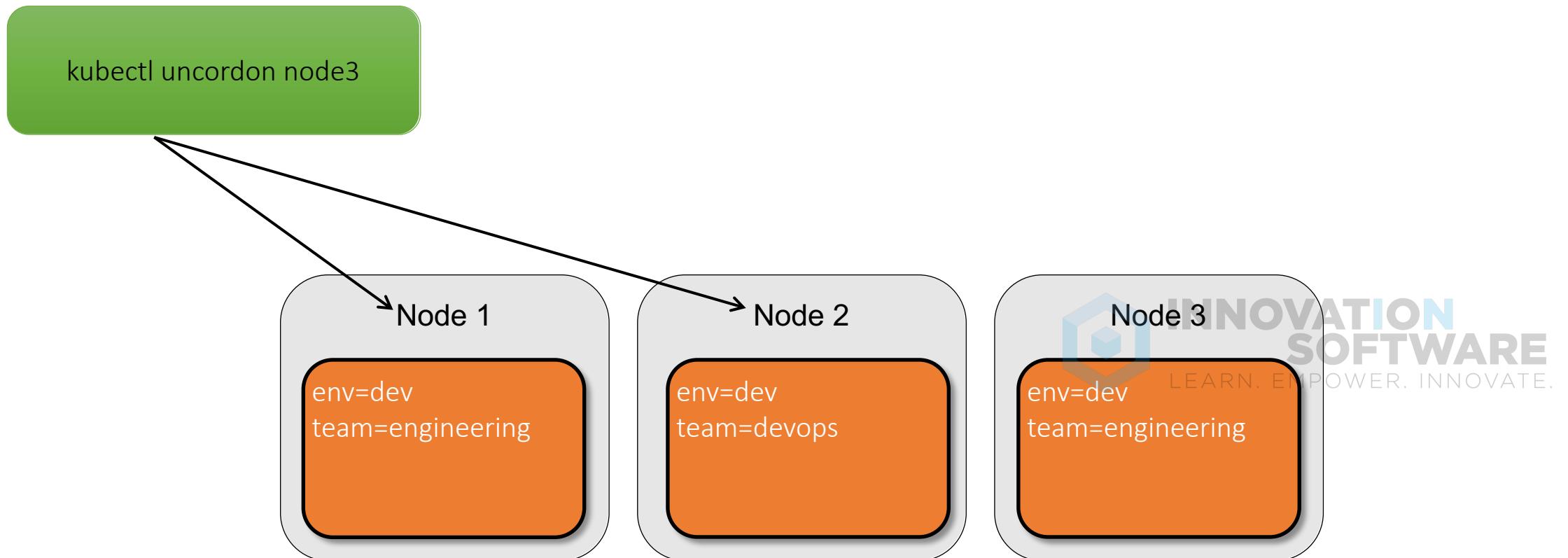
# Node Management: Uncordon

- Return Node 3 to active state



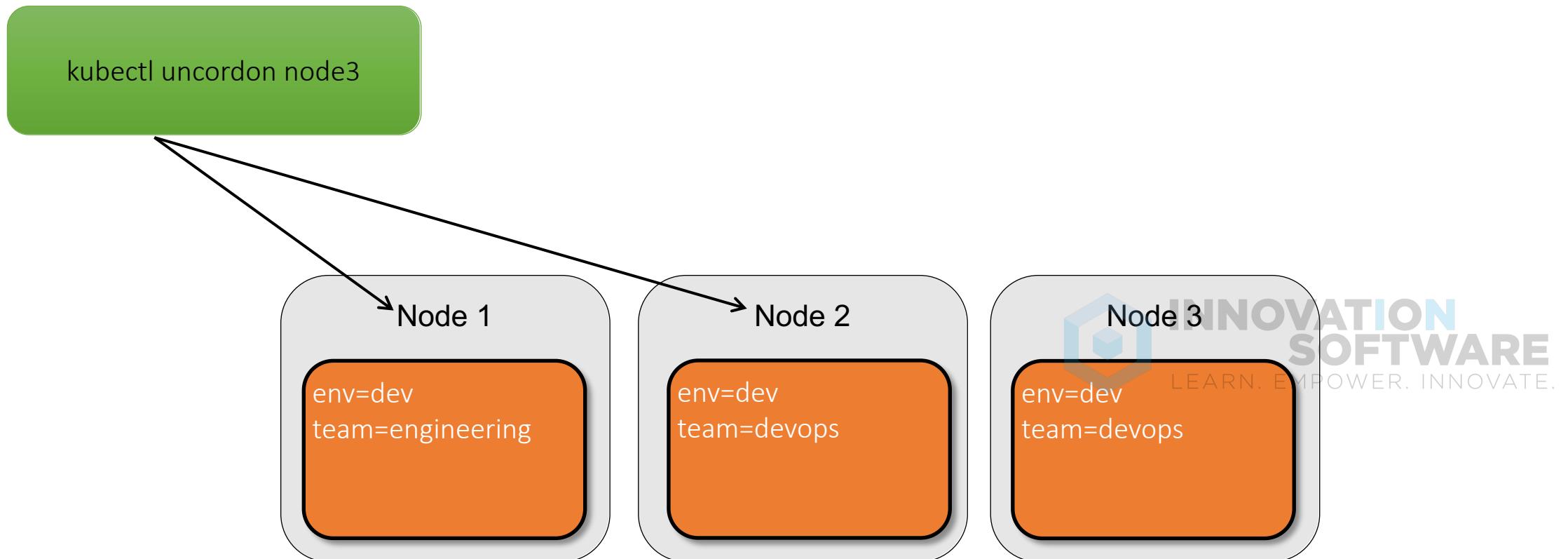
# Node Management: Uncordon

- Return Node 3 to active state



# Node Management: Uncordon

- Evict pods from Node 3



# Pod Affinity

- Define how pods should be placed relative to one another
- Spread or pack pods within a service or relative to pods in other services

```
affinity:  
  podAffinity:  
    requiredDuringSchedulingIgnoredDuringExecution:  
      - labelSelector:  
          matchExpressions:  
            - key: service  
              operator: In  
              values: ["S1"]  
    topologyKey: failure-domain.beta.kubernetes.io/zone
```



# Pod Affinity

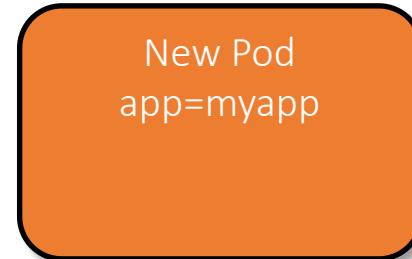
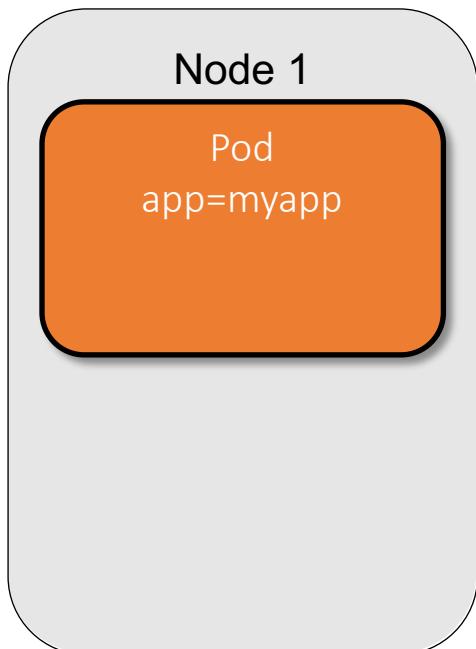
- Define how pods should be placed relative to one another
- Spread or pack pods within a service or relative to pods in other services

```
affinity:  
  podAffinity:  
    preferredDuringSchedulingIgnoredDuringExecution:  
      - labelSelector:  
          matchExpressions:  
            - key: service  
              operator: In  
              values: ["S1"]  
    topologyKey: failure-domain.beta.kubernetes.io/zone
```



# Pod Affinity

- Schedule Pods onto nodes that have same labels
  - app=myapp
  - operator: In
  - kubernetes.io/hostname



```
affinity:  
podAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "kubernetes.io/hostname"
```

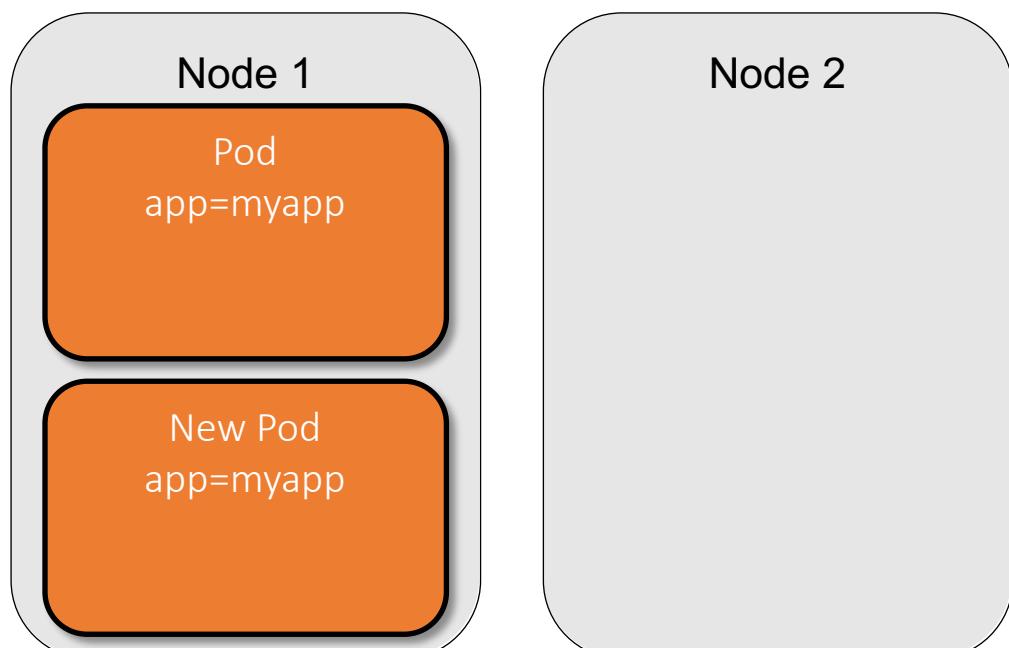


INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

The text block contains the YAML configuration for pod affinity, including the requiredDuringSchedulingIgnoredDuringExecution section, a label selector for the "app" key with value "myapp", and a topology key of "kubernetes.io/hostname". The Innovation Software logo is located at the bottom right of the text block.

# Pod Affinity

- Schedule Pods onto nodes that have same labels
  - app=myapp
  - operator: In
  - kubernetes.io/hostname



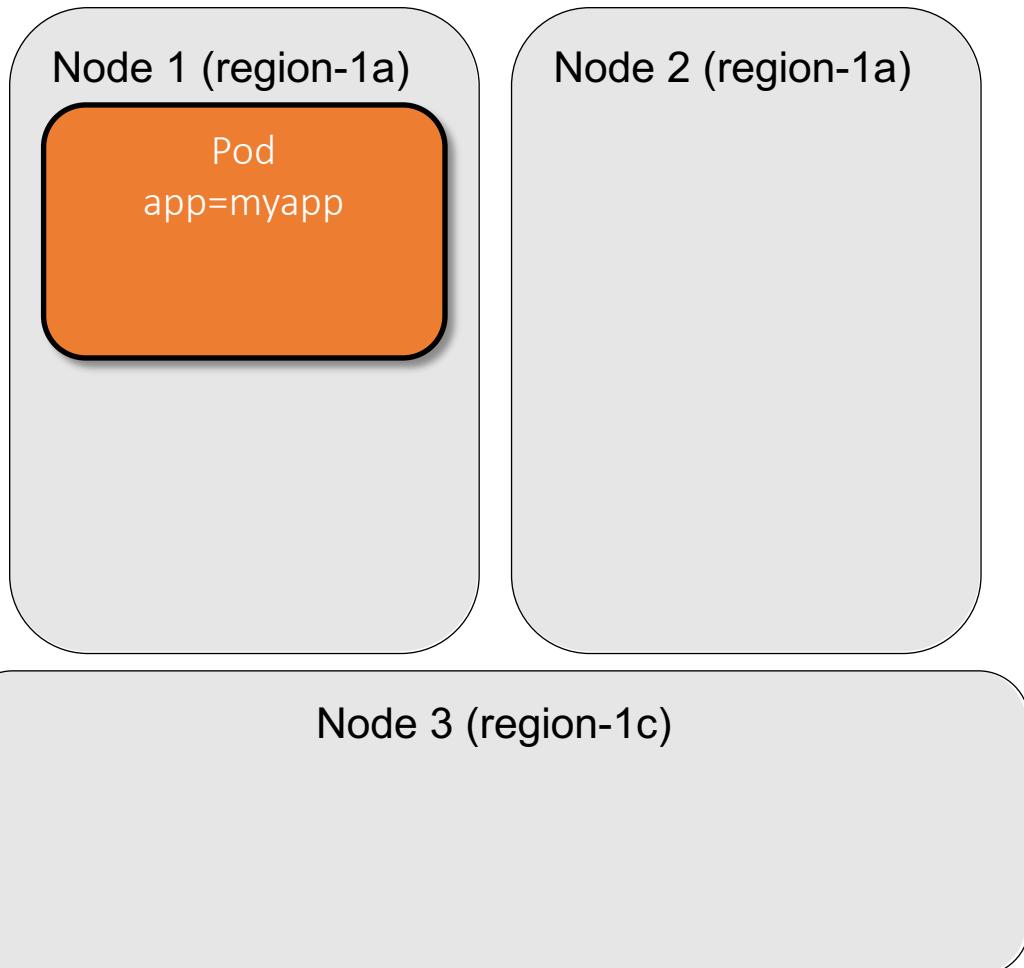
```
affinity:  
podAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "kubernetes.io/hostname"
```



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Pod Affinity

- Schedule Pods onto nodes in same region
  - app=db
  - failure-domain.beta.kubernetes.io/zone

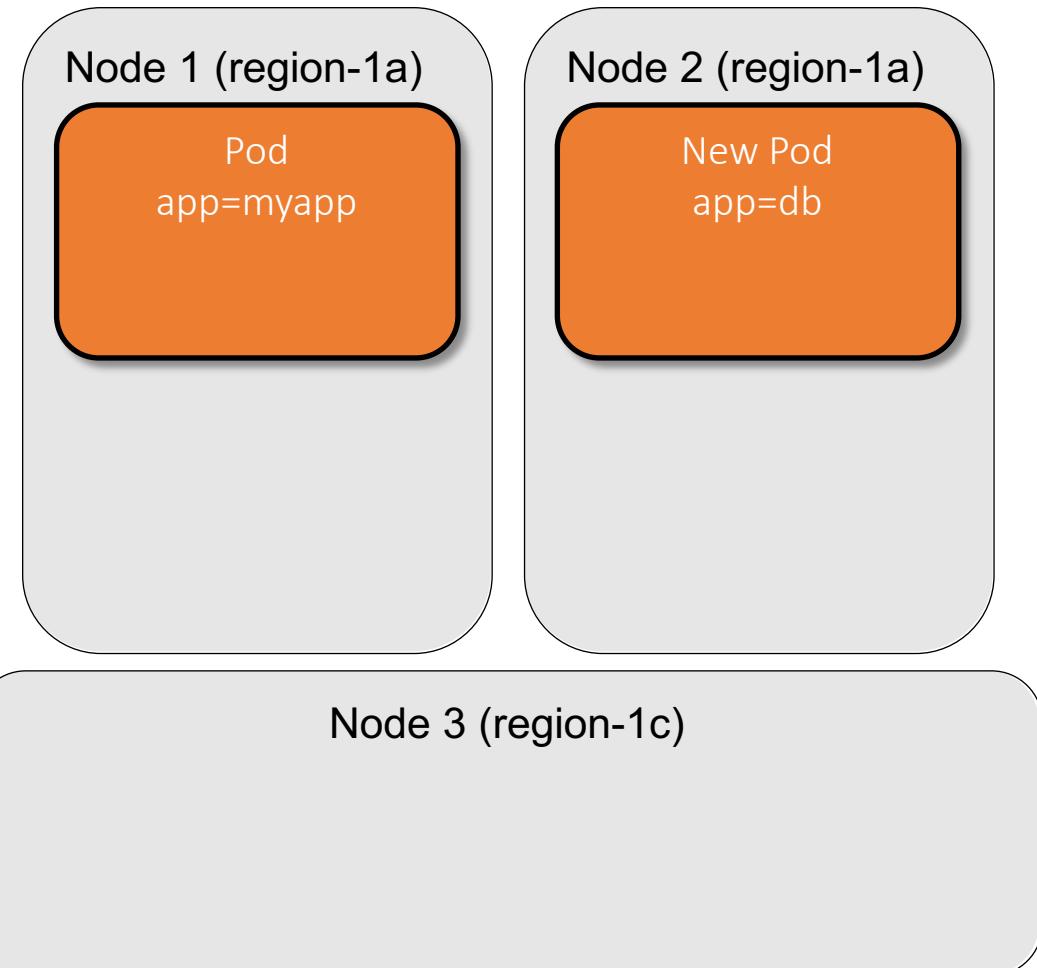


```
affinity:  
podAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "failure-domain.beta.kubernetes.io/zone"
```



# Pod Affinity

- Schedule Pods onto nodes in same region
  - app=db
  - failure-domain.beta.kubernetes.io/zone



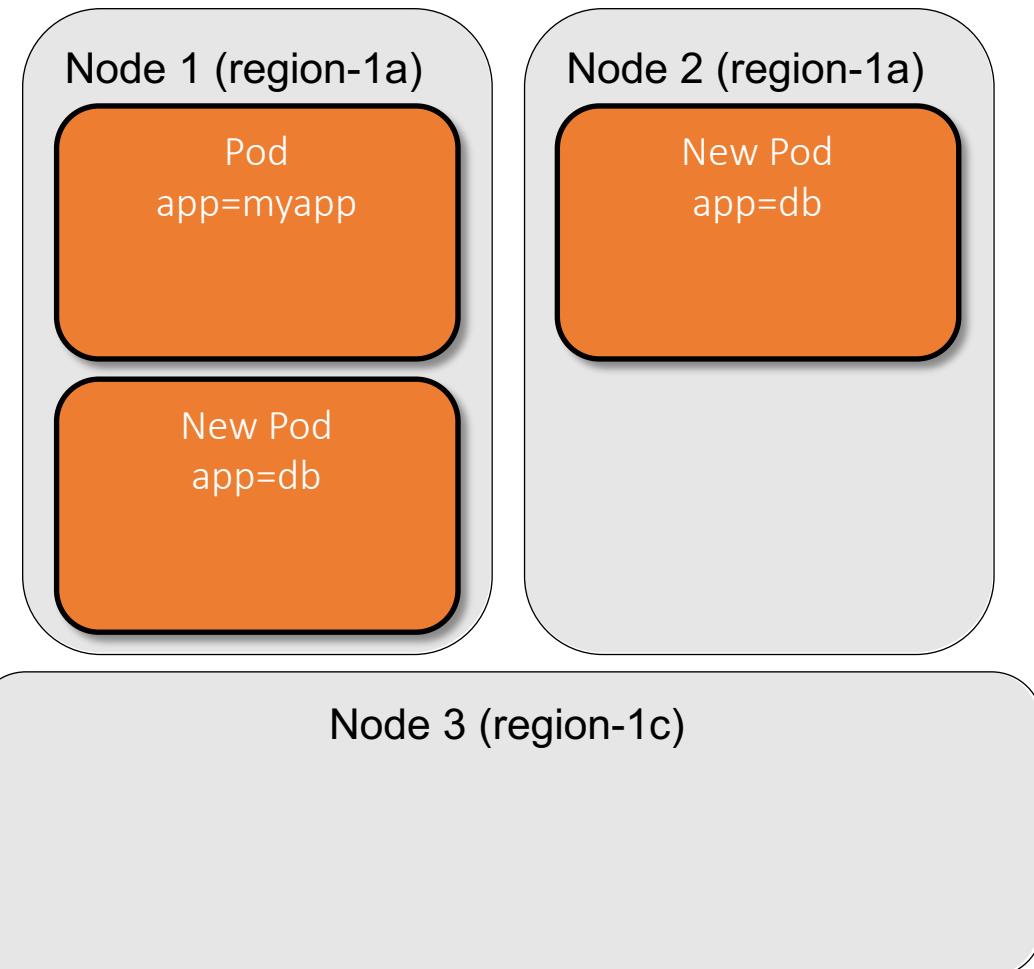
```
affinity:  
podAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "failure-domain.beta.kubernetes.io/zone"
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Pod Affinity

- Schedule Pods onto nodes in same region
  - app=db
  - failure-domain.beta.kubernetes.io/zone



```
affinity:  
podAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "failure-domain.beta.kubernetes.io/zone"
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Pod Anti-Affinity

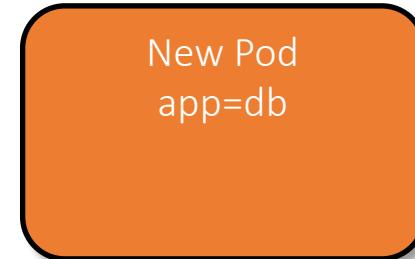
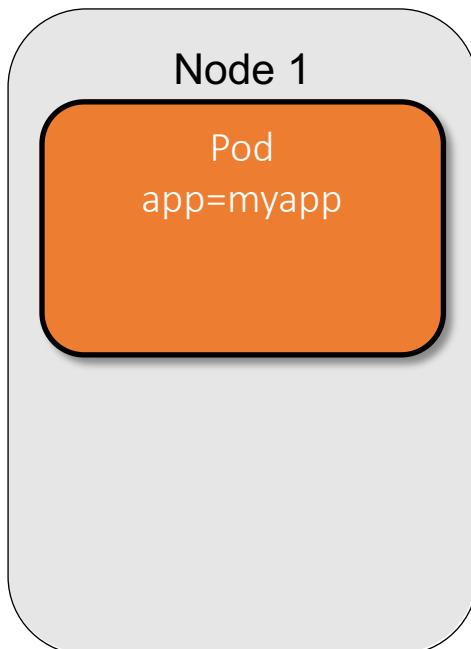
- Define how pods should be placed relative to one another
- PODs of different services run on different nodes.

```
affinity:  
  podAntiAffinity:  
    preferredDuringSchedulingIgnoredDuringExecution:  
      - labelSelector:  
          matchExpressions:  
            - key: service  
              operator: In  
              values: ["S1"]  
        topologyKey: kubernetes.io/hostname
```



# Pod Anti-Affinity

- Do not schedule Pods onto nodes with matching labels.
  - app=db
  - operator: In
  - kubernetes.io/hostname



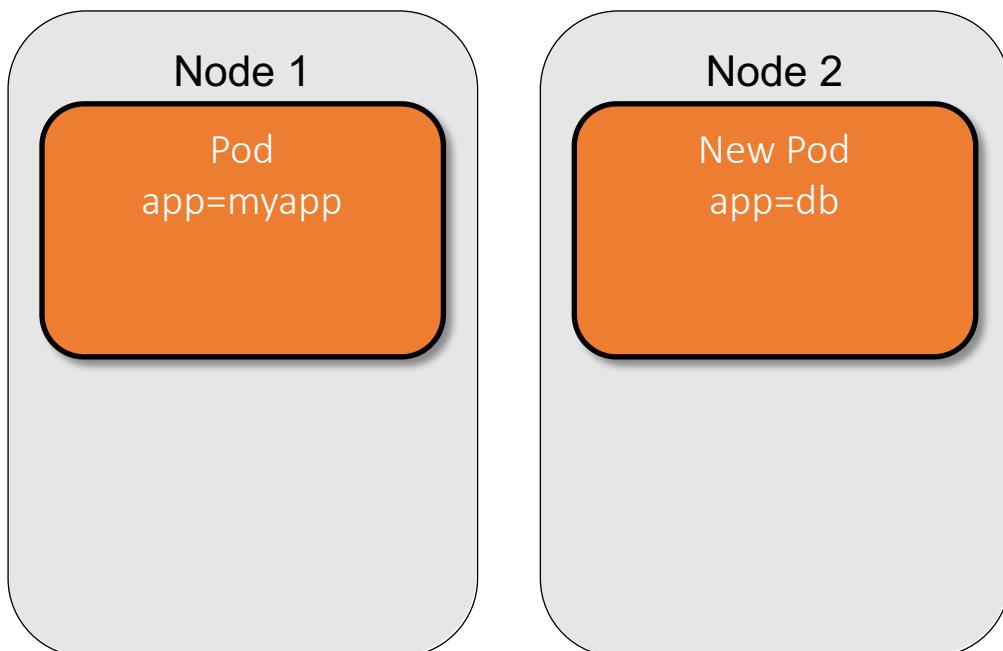
affinity:  
podAntiAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
    - key: "app"  
      operator: In  
      values:  
        - myapp  
topologyKey: "kubernetes.io/hostname"



INNOVATION  
SOFTWARE  
LEARN. EMPOWER. INNOVATE.

# Pod Anti-Affinity

- Do not schedule Pods onto nodes with matching labels.
  - app=db
  - operator: In
  - kubernetes.io/hostname



```
affinity:  
podAntiAffinity:  
requiredDuringSchedulingIgnoredDuringExecution:  
- labelSelector:  
  matchExpressions:  
  - key: "app"  
    operator: In  
    values:  
    - myapp  
topologyKey: "kubernetes.io/hostname"
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Scheduling Lab

- Set label on nodes
- Update POD YAML with nodeSelector
- Apply Affinity/Anti-Affinity YAML
- Confirm PODs are scheduled as expected

# Taint Lab

- Set Taint on node
- Create deployment without toleration
- Update YAML with toleration
- Create deployment with toleration
- Confirm PODs scheduled as expected
- Deploy Job
- Deploy CronJob

# Custom Scheduler(s)

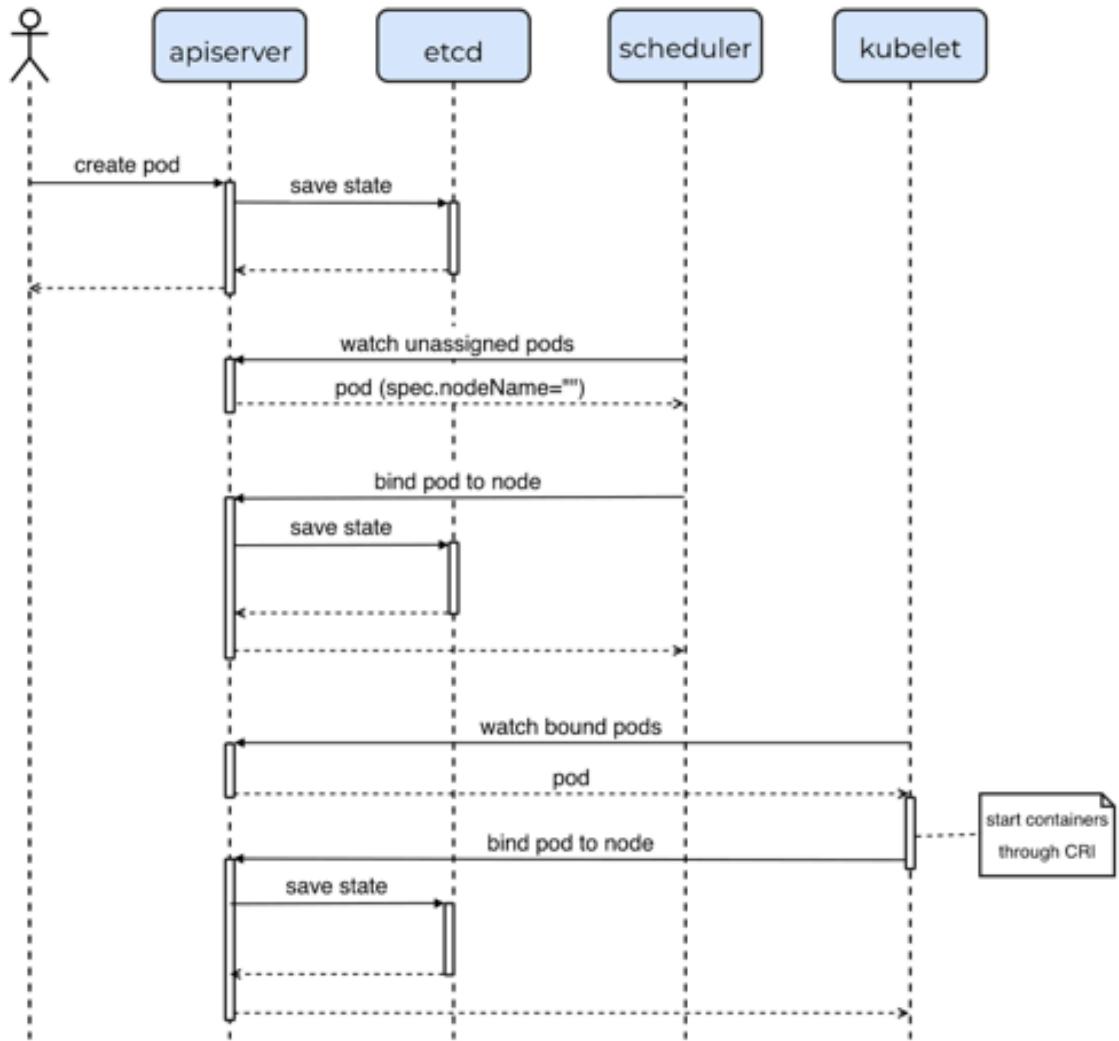


# Custom Schedulers

- Create your own schedulers
- Run alongside or replace default scheduler
- Beta in Kubernetes



# Custom Schedulers



1. Pod created and state saved in etcd without node
2. Scheduler notices new Pod without node
3. Finds node with available resources
4. Tells apiserver to bind pod to node and apiserver updates etcd with node assignment
5. Kubelet sees new pod request and creates it on node.

# Scheduler flow

1. A loop to watch the unbound pods in the cluster through querying the apiserver
2. Custom logic that finds the best node for a pod.
3. A request to the bind endpoint on the apiserver.

# Custom Scheduler Example

- Custom scheduler defined in YAML
- Default scheduler ignores PODs

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  schedulerName: my-scheduler
  containers:
  - name: nginx
    image: nginx:1.10
```



# Custom Schedulers

- Can be written in any language
- Simple or complex
- 3<sup>rd</sup> party schedulers available.



# Custom Scheduler Code (Bash)

```
#!/bin/bash
SERVER='localhost:8001'
while true;
do
    for PODNAME in $(kubectl --server $SERVER get pods -o json | jq '.items[] | select(.spec.schedulerName == "my-scheduler") | select(.spec.nodeName == null) | .metadata.name' | tr -d "'")
    ;
    do
        NODES=$(kubectl --server $SERVER get nodes -o json | jq '.items[].metadata.name' | tr -d "'")
        NUMNODES=${#NODES[@]}
        CHOSEN=${NODES[$RANDOM % $NUMNODES ]}
        curl --header "Content-Type:application/json" --request POST --data '{"apiVersion":"v1", "kind": "Binding", "metadata": {"name": "'$PODNAME'"}, "target": {"apiVersion": "v1", "kind": "Node", "name": "'$CHOSEN'"} }' http://$SERVER/api/v1/namespaces/default/pods/$PODNAME/binding/
        echo "Assigned $PODNAME to $CHOSEN"
    done
    sleep 1
done
```

# Custom Scheduler v1 (Go)

```
watch, err := s.clientset.CoreV1().Pods("").Watch metav1.ListOptions{
    FieldSelector: fmt.Sprintf("spec.schedulerName=%s,spec.nodeName=", schedulerName),
}

...

for event := range watch.ResultChan() {
    if event.Type != "ADDED" {
        continue
    }
    p := event.Object.(*v1.Pod)
    fmt.Println("found a pod to schedule:", p.Namespace, "/", p.Name)
}

}
```



# Custom Scheduler v1 (Go)

```
nodes, _ := s.clientset.CoreV1().Nodes().List metav1.ListOptions{}  
return &nodes.Items[rand.Intn(len(nodes.Items))], nil
```

- Find a fitting node (random)
- We are querying the apiserver for list of nodes on every schedule event.
- Bad for performance.



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Custom Scheduler v1 (Go)

```
s.clientset.CoreV1().Pods(p.Namespace).Bind(&v1.Binding{  
    ObjectMeta: metav1.ObjectMeta{  
        Name:      p.Name,  
        Namespace: p.Namespace,  
    },  
    Target: v1.ObjectReference{  
        APIVersion: "v1",  
        Kind:       "Node",  
        Name:       randomNode.Name,  
    },  
})
```

- After finding node we use the 'Bind' function to tell the apiserver so it can update etcd and have the kubelet create the Pod.



# Custom Scheduler v1 (Go)

```
timestamp := time.Now().UTC()
s.clientset.CoreV1().Events(p.Namespace).Create(&v1.Event{
    Count:      1,
    Message:    message,
    Reason:    "Scheduled",
    LastTimestamp: metav1.NewTime(timestamp),
    FirstTimestamp: metav1.NewTime(timestamp),
    Type:      "Normal",
    Source: v1.EventSource{
        Component: schedulerName,
    },
    InvolvedObject: v1.ObjectReference{
        Kind:    "Pod",
        Name:    p.Name,
        Namespace: p.Namespace,
        UID:     p.UID,
    },
    ObjectMeta: metav1.ObjectMeta{
        GenerateName: p.Name + "-",
    },
})
```

- Add scheduled events so we can track when Pod is scheduled.



# Custom Scheduler v2 (Go)

- Improve performance:
  - SharedInformers provide hooks to receive notifications of adds, updates, and deletes for a particular resource. They also provide convenience functions for accessing shared caches.

```
nodeInformer := factory.Core().V1().Nodes()
nodeInformer.Informer().AddEventHandler(cache.ResourceEventHandlerFuncs{
    AddFunc: func(obj interface{}) {
        node := obj.(*v1.Node)
        log.Printf("New Node Added to Store: %s", node.GetName())
    },
})
factory.Start.quit)
return nodeInformer.Lister()
```



# Custom Scheduler v2 (Go)

- Update 'Bind' code so it looks at cached node list.

```
nodes, err := s.nodeLister.List(labels.Everything())
return nodes[rand.Intn(len(nodes))], nil
```

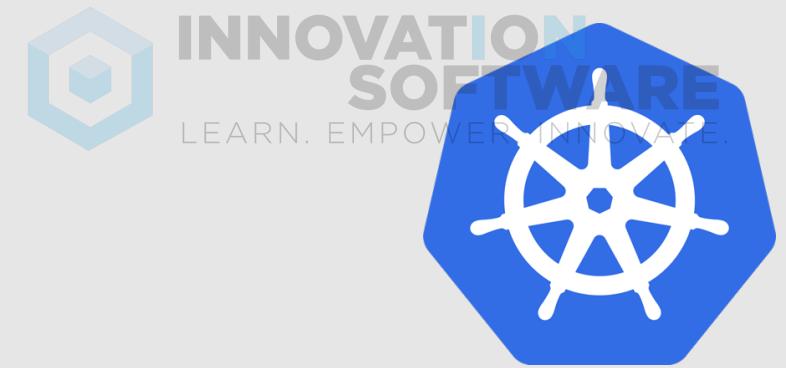


# Custom Scheduler Lab

- Deploy custom Go scheduler
- Update YAML manifest to use custom scheduler
- Deploy application and confirm it uses custom scheduler

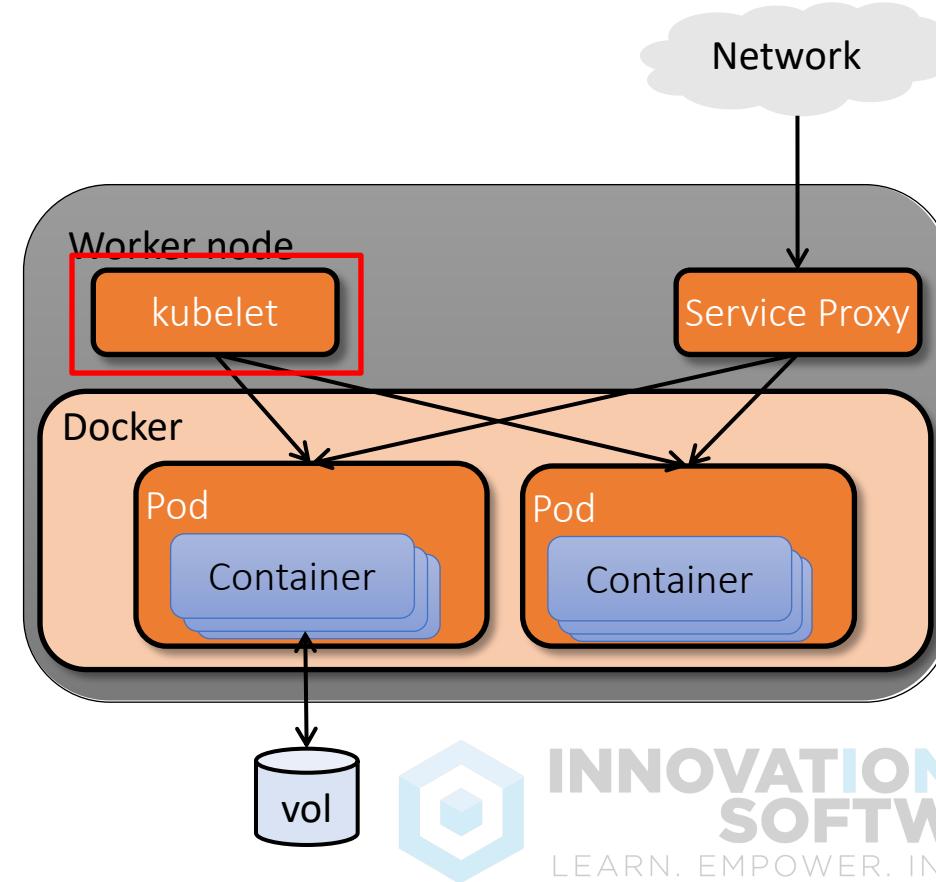


# Kubelet Configuration



# Kubelet Architecture

- **kubelet**: local K8s agent that is responsible for operations on the node, including
  - Watching for pod assignments
  - Mounting pod required volumes
  - Running a pod's containers
  - Executing container liveness probes
  - Reporting pod status to system
  - Reporting node status to system



# Kubelet Configuration

- Kubelet requires PodSpec
- PodSpec: YAML or JSON object describes a POD
- Manages PODs created with Kubernetes



# Kubelet Configuration

There are 4 ways a container manifest can be provided to the Kubelet.

- PodSpec from API server (most common)
- File: Path passed as a flag on the command line.
- HTTP endpoint: passed as a parameter on command line.
- HTTP server: Listen for HTTP and respond to simple API to submit new manifests



# Kubelet Path

```
root      15375  1.9  1.9 427928 79988 ?          Ssl  02:07  0:01 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true --network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin --cluster-dns=10.96.0.10 --cluster-domain=cluster.local --authorization-mode=Webhook --client-ca-file=/etc/kubernetes/pki/ca.crt --cadvisor-port=0 --rotate-certificates=true --cert-dir=/var/lib/kubelet/pki
```



# Kubelet API

```
ubuntu@ip-10-0-100-134:~$ ps auxwww | grep -i [k]ubelet
root      12691  1.7  2.3 513000 93516 ?        Ssl  01:43   2:57 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf
--pod-manifest-path=/etc/kubernetes/manifests --allow-privileged=true --network-plugin=cni --cni-conf-dir=/etc/cni/net.d --cni-bin-dir=/opt/cni/bin --cluster-dns=10.96.0.10 --cluster-domain=cluster.local --authorization-mode=Webhook --client-ca-file=/etc/kubernetes/pki/ca.crt --cadvisor-port=0 --rotate-certificates=true --cert-dir=/var/lib/kubelet/pki
root      13140  2.0  6.7 372616 271444 ?        Ssl  01:43   3:28 kube-apiserver --tls-private-key-file=/etc/kubernetes/pki/apiserver.key --admission-control=Initializers,NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,DefaultTolerationSeconds,NodeRestriction,ResourceQuota --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname --requestheader-group-headers=X-Remote-Group --service-cluster-ip-range=10.96.0.0/12 --client-ca-file=/etc/kubernetes/pki/ca.crt --requestheader-username-headers=X-Remote-User --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --insecure-port=0 --requestheader-extra-headers-prefix=X-Remote-Extra- --secure-port=6443 --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key --enable-bootstrap-token-auth=true --allow-privileged=true --requestheader-allowed-names=front-proxy-client --advertise-address=10.0.100.134 --service-account-key-file=/etc/kubernetes/pki/sa.pub --authorization-mode=Node,RBAC --etcd-servers=http://127.0.0.1:2379
```

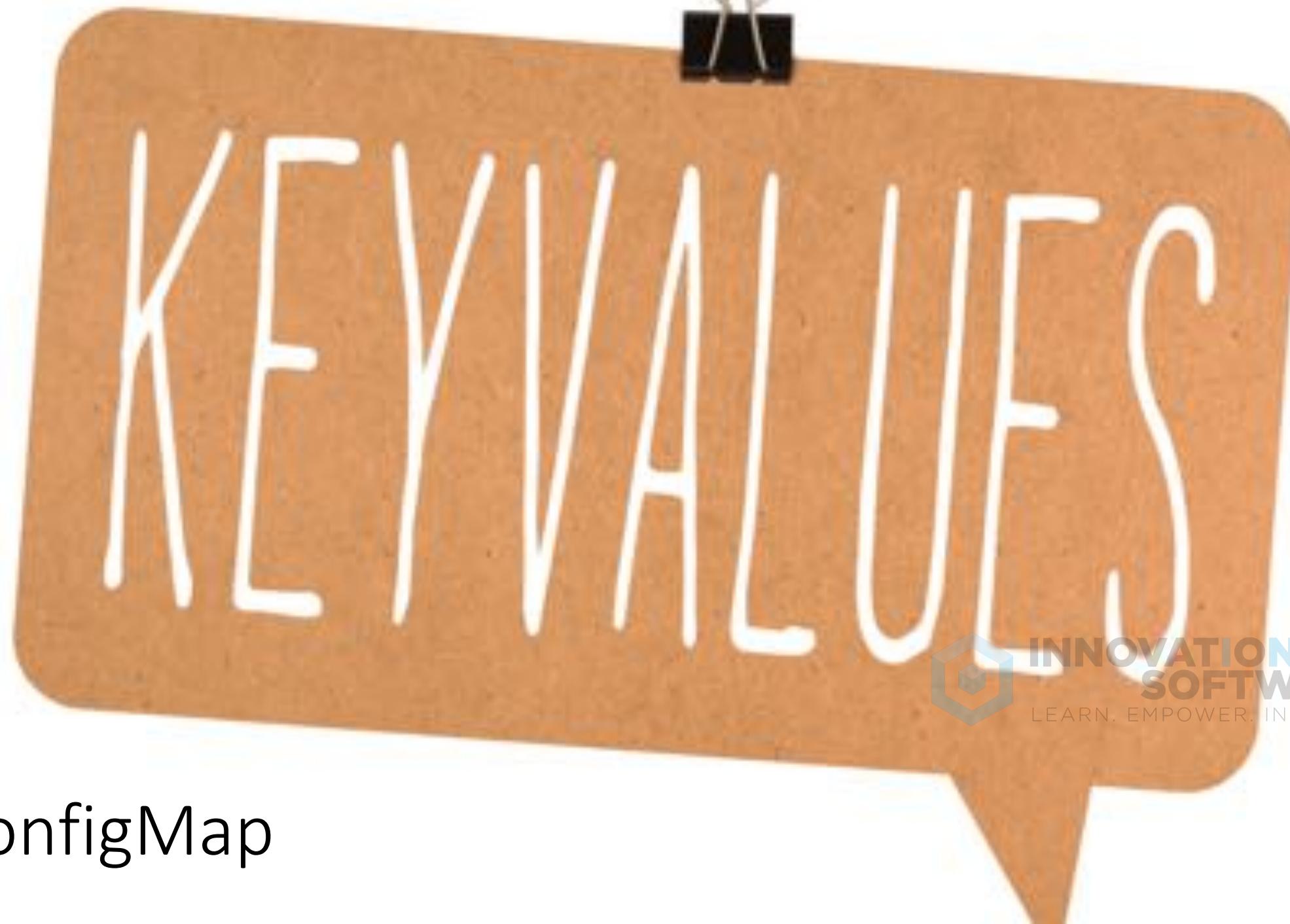


**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# Kubelet Lab

- Install Kubelet in standalone mode
- Deploy v1 application YAML
- Update to v2 application YAML





KEYVALUES



INNOVATION  
SOFTWARE

LEARN. EMPOWER. INNOVATE.

ConfigMap

# ConfigMap

- Many applications require configuration via:
  - Config Files
  - Command-Line Arguments
  - Environment Variables
- These need to be decoupled from images to keep portable
- ConfigMap API provides mechanisms to inject containers with configuration data
- Store individual properties or entire config files/JSON blobs
- Key-Value Pairs

# ConfigMap

- Not meant for sensitive information
- PODs or controllers can use ConfigMaps

1. Populate the value of environment variables
2. Set command-line arguments in a container
3. Populate config files in a volume

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```



# ConfigMap from directory

- 2 files in docs/user-guide/configmap/kubectl
  - game.properties
  - ui.properties

## ui.properties

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

## game.properties

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```



**INNOVATION  
SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# ConfigMap from directory

```
kubectl create configmap game-config --from-file=docs/user-guide/configmap/kubectl
```

```
kubectl describe configmaps game-config
```

```
Name: game-config  
Namespace: default  
Labels: <none>  
Annotations: <none>
```

```
Data
```

```
====
```

```
game.properties: 121 bytes  
ui.properties: 83 bytes
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# ConfigMap from directory

```
kubectl get configmaps game-config -o yaml
```

```
apiVersion: v1
data:
  game.properties: |-  
    enemies=aliens  
    lives=3  
    ...  
  ui.properties: |-  
    color.good=purple  
    ...  
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"-  
  selfLink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```



# ConfigMap from files

```
kubectl get configmaps \  
game-config-2 \  
-o yaml
```

```
kubectl create configmap \  
game-config-2 \  
--from-file=file1 \  
--from-file=file2
```

```
apiVersion: v1  
data:  
  game.properties: |-  
    enemies=aliens  
    lives=3  
    ...  
  ui.properties: |-  
    color.good=purple  
    ...  
kind: ConfigMap  
metadata:  
  creationTimestamp: 2016-02-18T18:52:05Z  
  name: game-config-2  
  namespace: default  
  resourceVersion: "516"-  
  selfLink: /api/v1/namespaces/default/configmaps/game-config-2  
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```



**INNOVATION**  
**SOFTWARE**  
LEARN. EMPOWER. INNOVATE.

# ConfigMap options

- `--from-file=/path/to/directory`
- `--from-file=/path/to/file1 (/path/to/file2)`
- Literal key=value: `--from-literal=special.how=very`

# ConfigMap in PODs

- Populate Environment Variables

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

## OUTPUT

```
SPECIAL_LEVEL_KEY=very
SPECIAL_TYPE_KEY=charm
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: busybox
      command: ["/bin/sh", "-c", "env"]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
  restartPolicy: Never
```



# ConfigMap Lab

- Create a ConfigMap to configure Redis as a cache
- Look at ConfigMap in YAML format
- Create a POD to use ConfigMap
- Confirm POD used ConfigMap settings
- Setup Node.js web app
- Configure Nginx reverse proxy