# Deploying and Maintaining Applications with DaemonSets and Jobs

**Anthony E. Nocentino**
ENTERPRISE ARCHITECT @ CENTINO SYSTEMS

@nocentino www.centinosystems.com

# Course Overview

Using Controllers to Deploy Applications and Deployment Basics

Maintaining Applications with Deployments

Deploying and Maintaining Applications with DaemonSets and Jobs

# Overview

**Working with Controllers in Kubernetes**

- **DaemonSets**
- **Jobs and CronJobs**
- **StatefulSets**

# Controllers in Kubernetes

**DaemonSet**

**Jobs**

**CronJobs**

**StatefulSets**

# Introducing DaemonSet

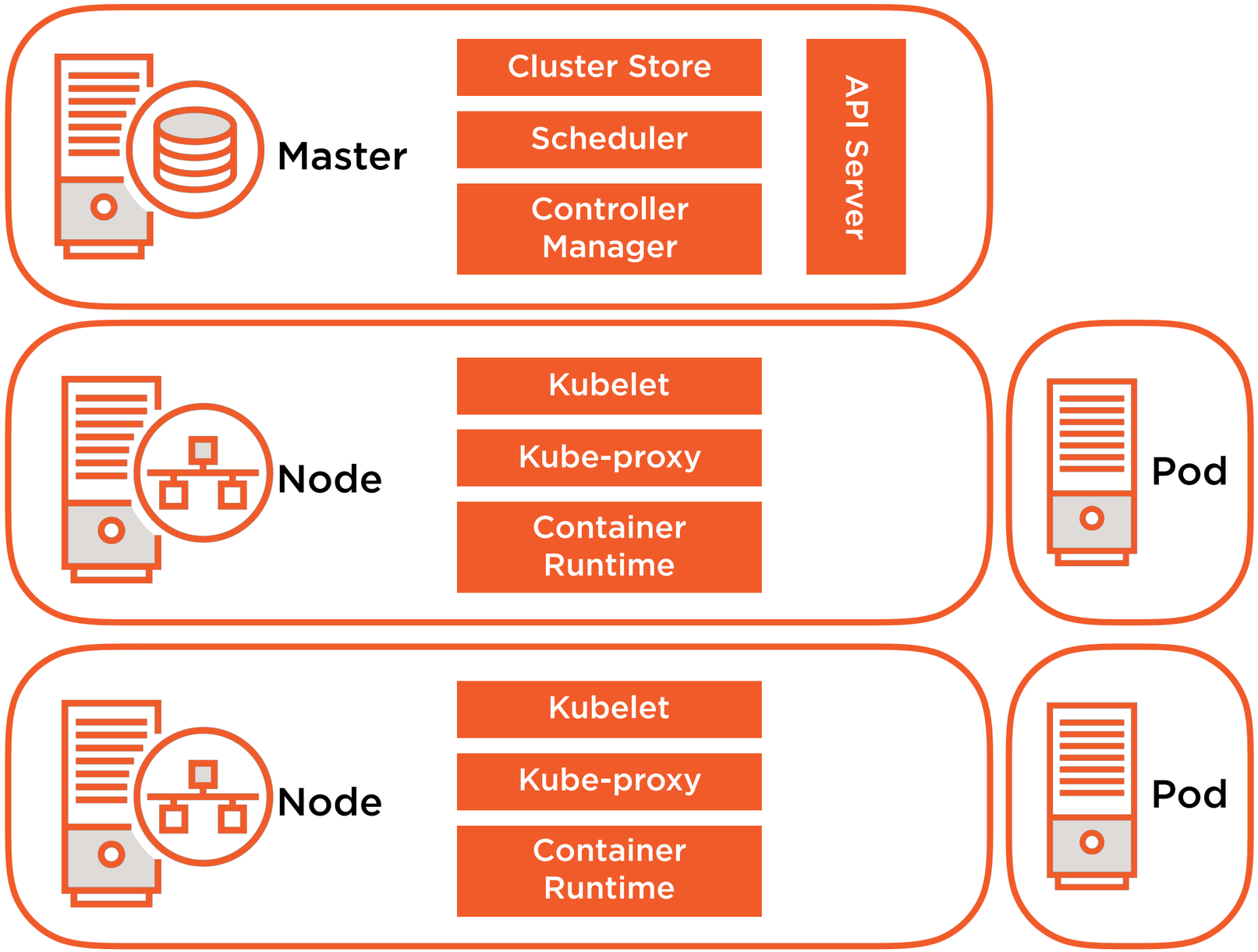
**Ensures that all or some Nodes run a Pod**

**Effectively an init daemon inside your cluster**

**Example workloads**

- `kube-proxy` **for network services**
- **Log collectors**
- **Metric servers**
- **Resource monitoring agents**
- **Storage daemons**

# Controller Operations - DaemonSets



# DaemonSet Pod Scheduling

One Pod will be scheduled to each worker Node in a cluster by the default-scheduler

As Nodes are added to the cluster, they will get a Pod

You can control which Nodes get Pods

- Node Selector
- Labeling the Nodes

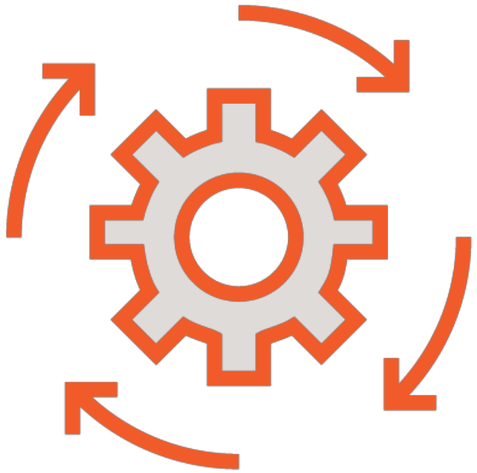
# Defining a DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-world-ds
spec:
  selector:
    matchLabels:
      app: hello-world-app
  template:
    metadata:
      labels:
        app: hello-world-app
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/hello-app:1.0
```

# Defining a DaemonSet with a nodeSelector

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-world-ds
spec:
  selector:
    matchLabels:
      app: hello-world-app
  template:
    metadata:
      labels:
        app: hello-world-app
    spec:
      nodeSelector:
        node: hello-world-ns
      containers:
        - name: hello-world
          image: gcr.io/google-samples/hello-app:1.0
```

# DaemonSet Update Strategy



**RollingUpdate**

**OnDelete**

# Demo

Creating a DaemonSet

- All Nodes
- Subset of Nodes

Updating a DaemonSet

Controllers so far introduced, start up and run Pods continuously...

but what if you  wanted to run a single task?

# Introducing Jobs

Jobs **create one or more Pods**

**Runs a program in a container to completion**

**Ensure that the specified number of Pods complete successfully**

**Workload examples**

- **Ad-hoc**
- **Batch**
- **Data oriented tasks**

# Ensuring Jobs Run to Completion

**Interrupted Execution**

**Non-zero Exit Code**

**Rescheduled**

**`restartPolicy`**

# Jobs Lifecycle

Jobs are tasks that we need to ensure run to completion

When a Job completes successfully

- Its status is set to 'Completed'
- The Job object remains
- The Pods are not deleted
- This way we can keep them around for their logs and other output

It is up to the user to delete the Job when finished, this will delete the Pods

# Defining a Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: hello-world
spec:
  template:
    spec:
      containers:
      - name: ubuntu
        image: ubuntu
        command:
         - "/bin/bash"
         - "-c"
         - "/bin/echo Hello from Pod $(hostname) at $(date)"
      restartPolicy: Never
```

# Controlling Job Execution

**`backoffLimit` - number of Job retries before it's marked failed**

**`activeDeadlineSeconds` - max execution time for the Job**

**`parallelism` - max number of running Pods in a Job at a point in time**

**`completions` - number of Pods that need to finish successfully**

# Introducing CronJobs

`CronJob` **will run a** `Job` **on a given time based schedule**

- **Conceptually similar to UNIX/Linux cron job**
- **Uses the standard cron format**

**Example Workloads**

- **Periodic workloads and scheduled tasks**

`CronJob` **resource is created when the object is submitted to the API Server**

**When it's time, a** `Job` **is created via the** `Job` **template from the** `CronJob` **Object**

# Controlling CronJobs Execution

**schedule - a cron formatted schedule**

**suspend - suspends the `CronJob`**

**`startingDeadlineSeconds` - the `Job` hasn't started in this amount of time mark it as Failed**

**`concurrencyPolicy` - handles concurrent executions of a `Job`. Allow, Forbid or Replace**

# Defining a CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello-world-cron
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: ubuntu
            ...
```

# Demo

**Executing tasks with** `Jobs`

**Failed** `Jobs` **and** `restartPolicy`

**Defining a Parallel** `Job`

**Scheduling tasks with** `CronJobs`

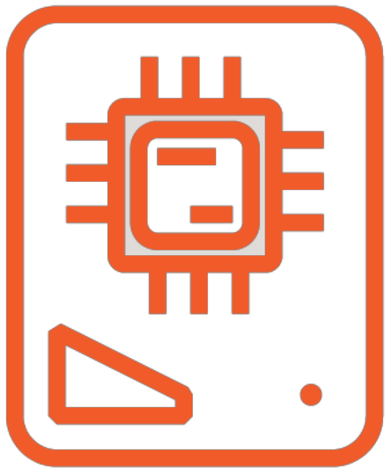
# StatefulSets

**Enables stateful applications to be managed by a controller**

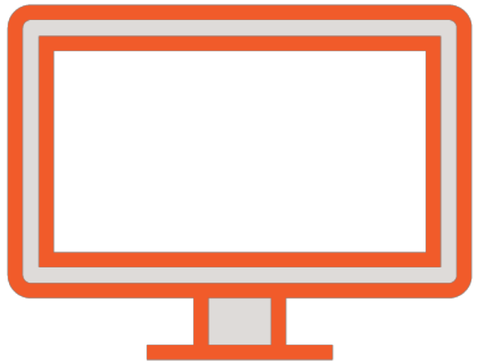
- **Database workloads**
- **Caching servers**
- **Application state for web farms**

# StatefulSet Capabilities

**Naming**



**Storage**



**Headless Service**

# Review

**Working with Controllers in Kubernetes**

- **DaemonSets**
- **Jobs and CronJobs**
- **StatefulSets**

# Thank You!

**@nocentino**

**www.centinosystems.com**