kubernetes

Advanced Concepts

Kubernetes

Course Objectives

In this module, you will learn:

- Commands & Arguments
- Environment variables
- Config Maps & Secrets
- Health Checks
- Jobs/ Cron Jobs
- Application Logs
- Daemon Sets / Stateful Sets

kubernetes

Commands & Arguments

Advanced Concepts

Dockerfile vs Pod YAML

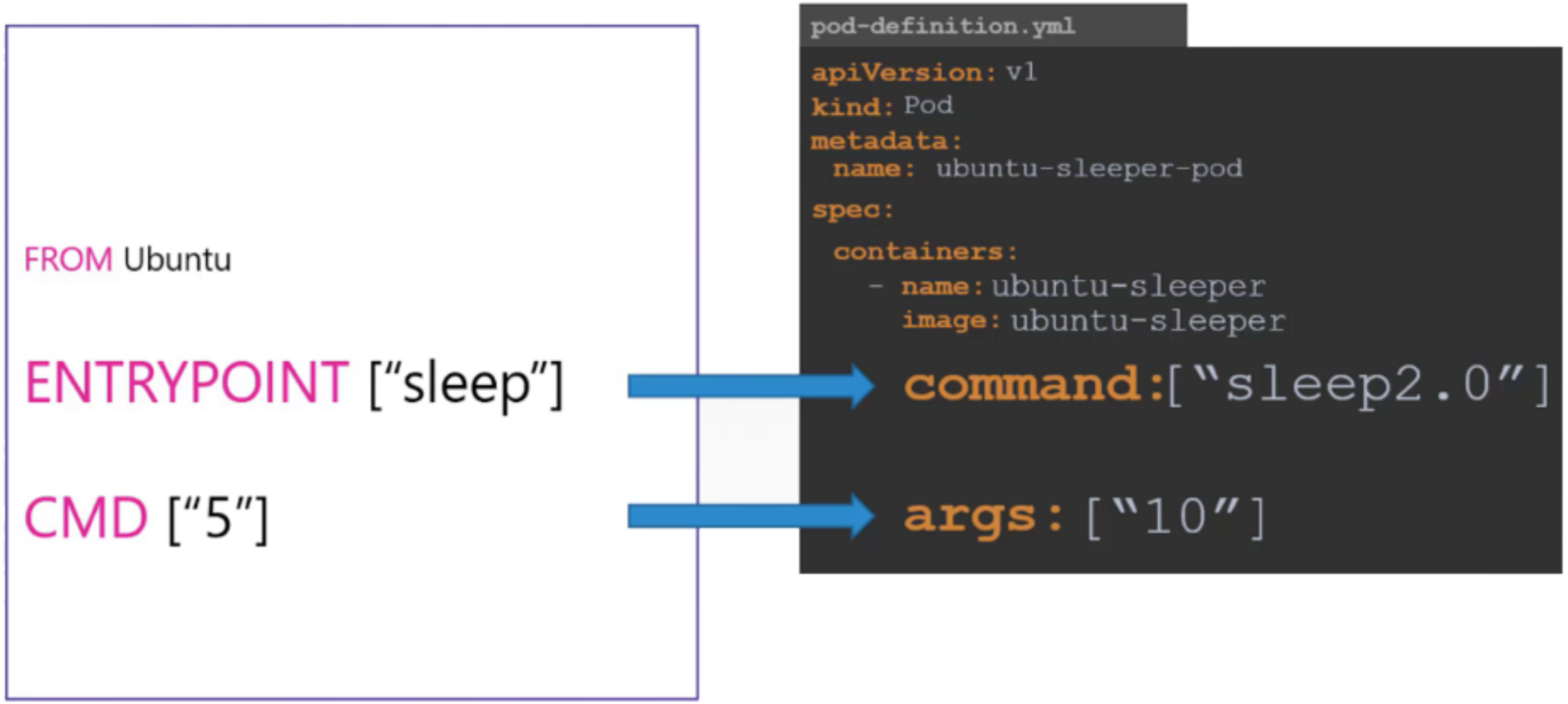
```
FROM Ubuntu

ENTRYPOINT ["sleep"]

CMD ["5"]
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-sleeper-pod
spec:
  containers:
    - name: ubuntu-sleeper
      image: ubuntu-sleeper
      args: ["10"]
```

```
kubectl create -f pod-definition.yml
```

Dockerfile vs Pod YAML



```
FROM Ubuntu

ENTRYPOINT [“sleep”]

CMD [“5”]
```

pod-definition.yml

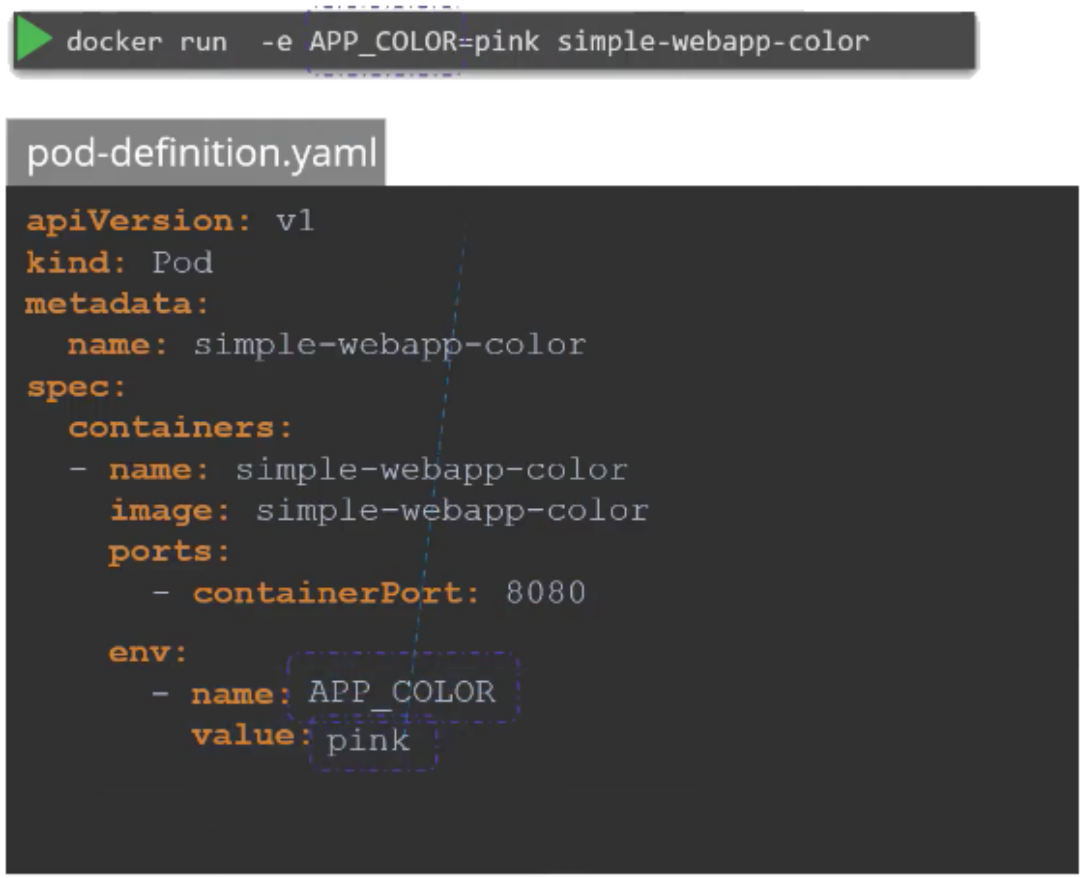
```
apiVersion: v1
kind: Pod
metadata:
 name: ubuntu-sleeper-pod
spec:
 containers:
   - name: ubuntu-sleeper
     image: ubuntu-sleeper
     command:[“sleep2.0”]

     args: [“10”]
```

kubernetes

Environment variables in Applications

Advanced Concepts

ENV variables in Kubernetes



```
docker run  -e APP_COLOR=pink simple-webapp-color
```

pod-definition.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080

    env:
      - name: APP_COLOR
        value: pink
```

ENV Value Types

```
env:
  - name: APP_COLOR
    value: pink
```

1 Plain Key Value

```
env:
  - name: APP_COLOR
    valueFrom:
        configMapKeyRef:
```

2 ConfigMap

```
env:
  - name: APP_COLOR
    valueFrom:
        secretKeyRef:
```

3 Secrets

kubernetes

Configuring Config Maps

Advanced Concepts

ConfigMap

- In programming, we use env files or separate configuration files to store settings, configurations, or variables that are required to execute the program.
- In Kubernetes, we can use ConfigMaps to achieve the same functionality.
- A ConfigMap is a Kubernetes API object that can be used to store data as key-value pairs. Kubernetes pods can use the created ConfigMaps as a:
  - Configuration file
  - Environment variable
  - Command-line argument

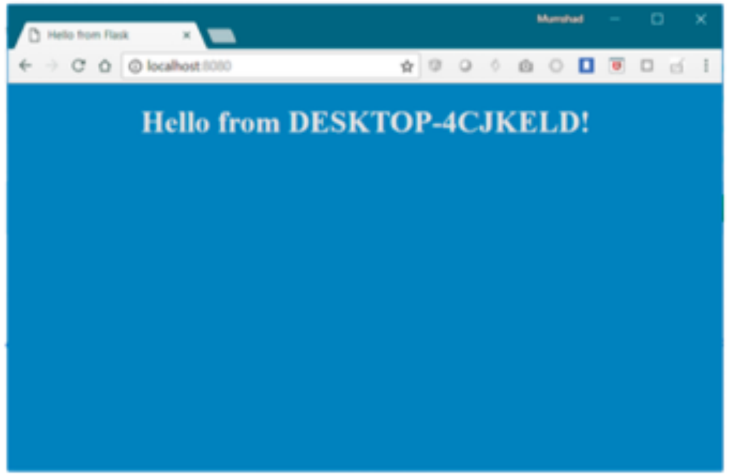
ConfigMap in Pods

pod-definition.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: simple-webapp-color
  labels:
    name: simple-webapp-color
spec:
  containers:
  - name: simple-webapp-color
    image: simple-webapp-color
    ports:
      - containerPort: 8080

    envFrom:
      - configMapRef:
            name: app-config
```

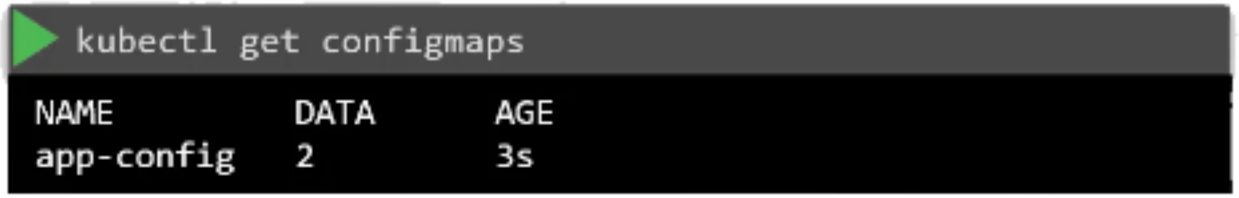
config-map.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  APP_COLOR: blue
  APP_MODE: prod
```

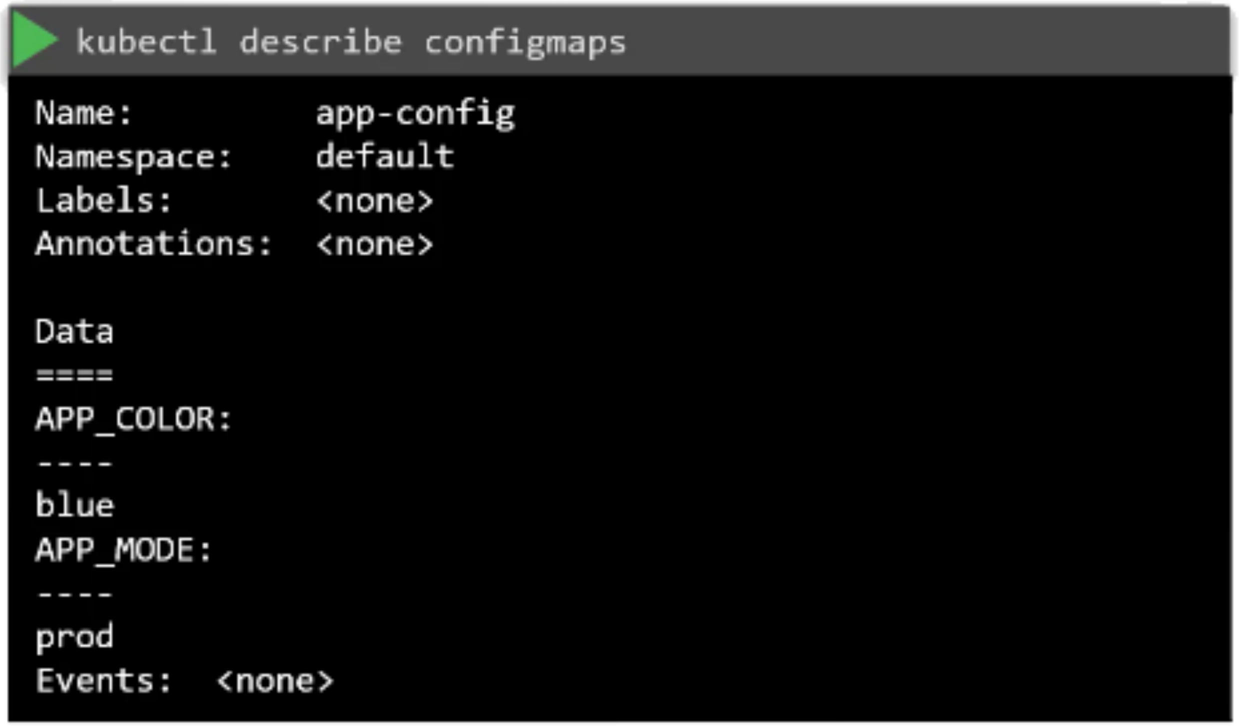

Hello from DESKTOP-4CJKELD!

```
kubectl create -f pod-definition.yaml
```

View ConfigMaps



```
kubectl get configmaps

NAME         DATA      AGE
app-config   2         3s
```



```
kubectl describe configmaps

Name:           app-config
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data
====
APP_COLOR:
----
blue
APP_MODE:
----
prod
Events:   <none>
```

kubernetes

Configuring Secrets

Advanced Concepts

Kubernetes Secrets

- A secret is an object that contains a small amount of sensitive data
  - Such as a password, a token, or a key
- Putting this information in a secret is safer and more flexible than putting it verbatim in a pod definition or in a Docker image
  - A secret is only sent to a node if a pod on that node requires it
  - Not written to disk—stored in a tmpfs on the nodes
  - Deleted once the pod that depends on it is deleted
  - One pod does not have access to the secrets of another pod
  - Secrets are encrypted at the storage layer in etcd

Creating Kubernetes Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Not encrypted, base64 encoded

```
$ kubectl get secret mysecret -o yaml
```

To get the secret

kubernetes

Health Checkers

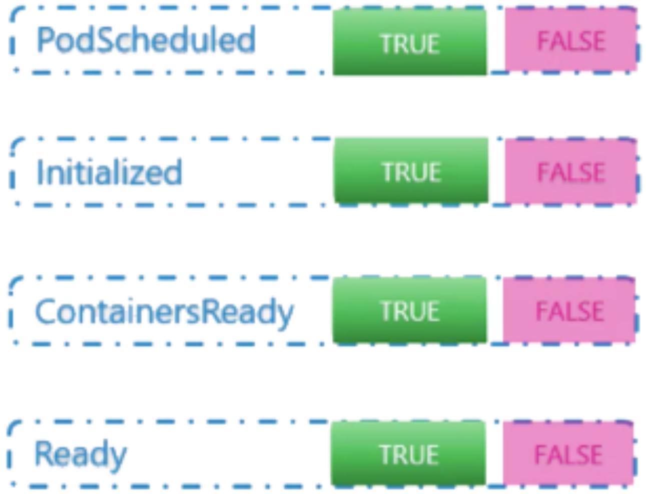
Application Lifecycle Management

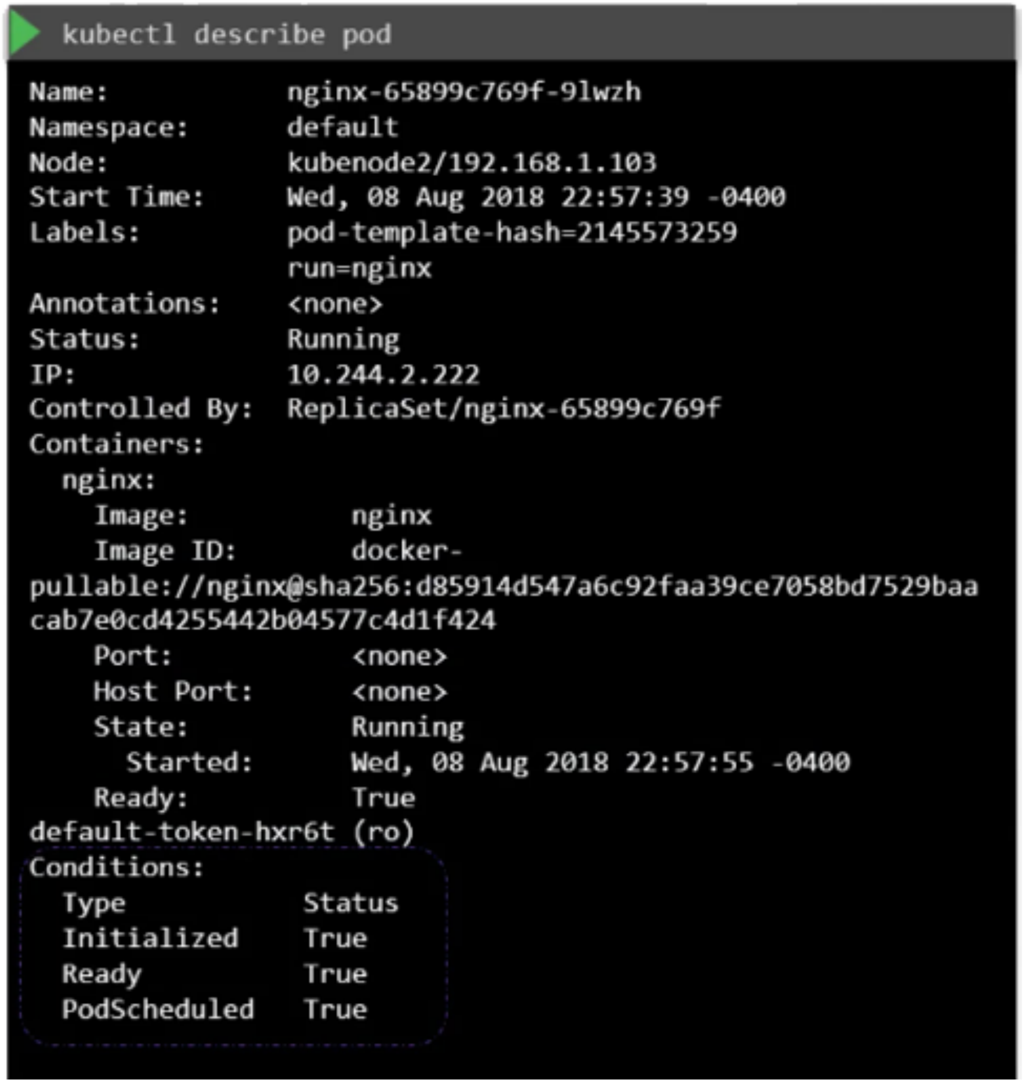
Health Checkers

There are two types of Kubernetes health checks: Liveness and Readiness probes

- If a Liveness probe fails, the pod is deleted and re-created
- If a Readiness probe fails, requests are not sent to the pod until it is back up and running
    - But it is not removed and recreated

POD Conditions

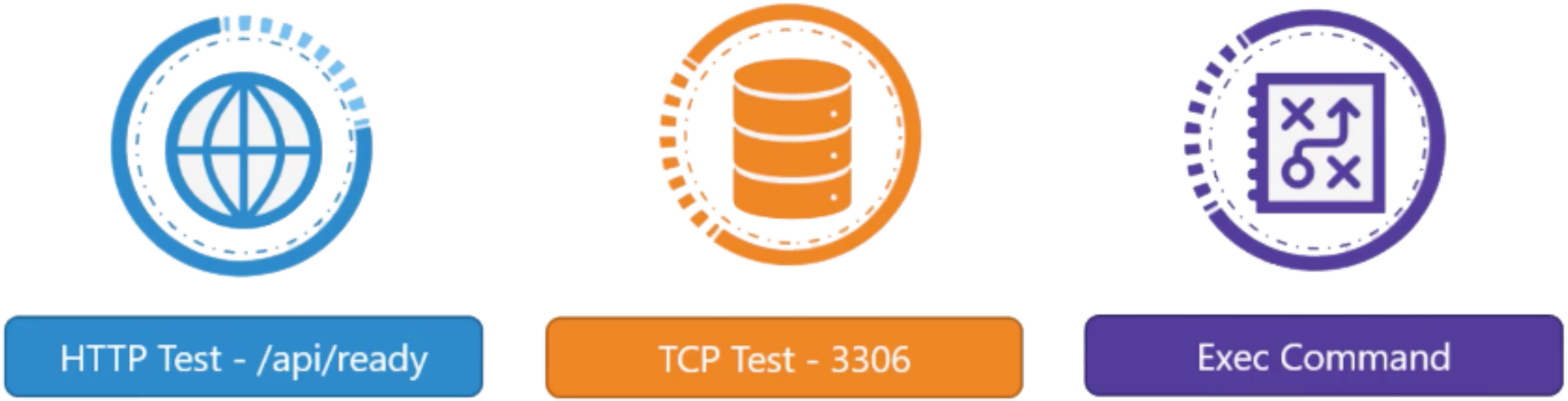




```
kubectl describe pod
```

```
Name:           nginx-65899c769f-9lwzh
Namespace:      default
Node:           kubenode2/192.168.1.103
Start Time:     Wed, 08 Aug 2018 22:57:39 -0400
Labels:         pod-template-hash=2145573259
                run=nginx
Annotations:    <none>
Status:         Running
IP:             10.244.2.222
Controlled By:  ReplicaSet/nginx-65899c769f
Containers:
  nginx:
    Image:          nginx
    Image ID:       docker-
pullable://nginx@sha256:d85914d547a6c92faa39ce7058bd7529baa
cab7e0cd4255442b04577c4d1f424
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Wed, 08 Aug 2018 22:57:55 -0400
    Ready:          True
default-token-hxr6t (ro)
Conditions:
  Type            Status
  Initialized     True
  Ready           True
  PodScheduled    True
```

Readiness/ Liveness Probes



Liveness and Readiness Probe Configuration

```
apiVersion: apps/v1
kind: Deployment
*** CODE OMITTED ***
    spec:
      containers:
      *** CODE OMITTED ***
        readinessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 30
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 15
```

Added to the Containers section of the Deployment

kubernetes

Jobs/ Cron Jobs

Application Lifecycle Management

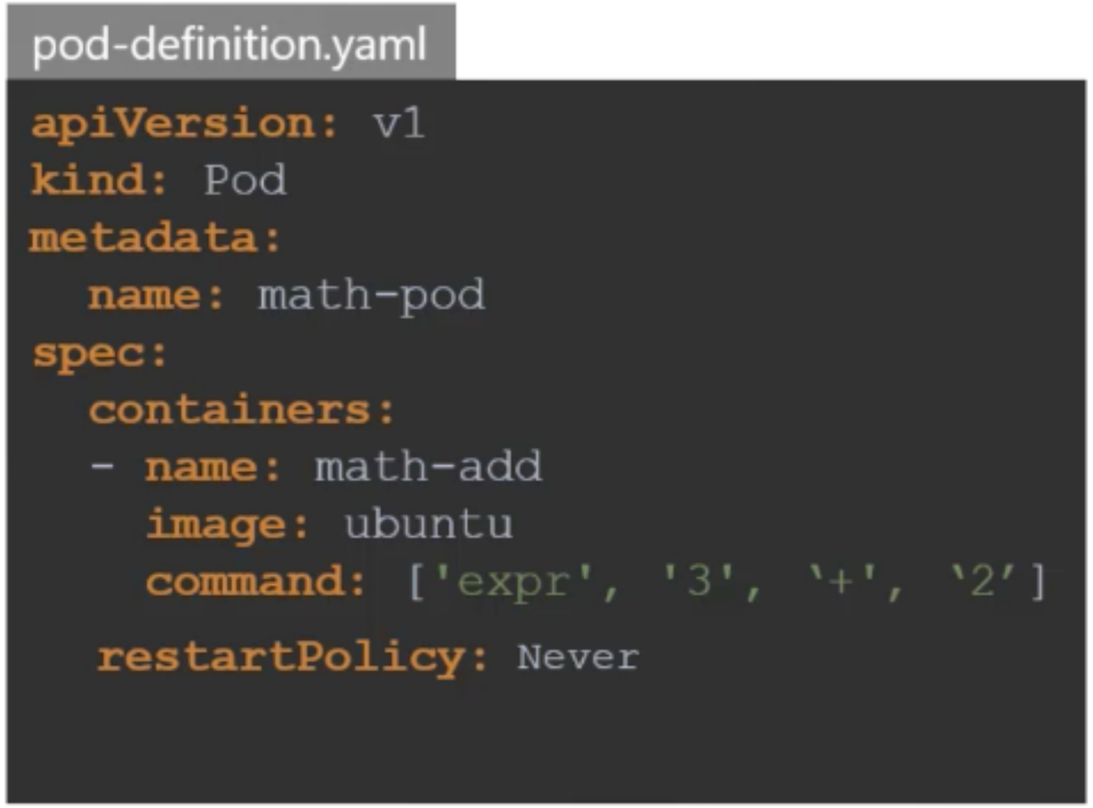
Job

- A Kubernetes Job is a workload controller that represents a finite task.
- Jobs differ from other controller objects in that Jobs manage the task as it runs to completion, rather than managing an ongoing desired state (such as the total number of running Pods).
- When a specified number of pods reach completion, the Job is said to have successfully completed.

Job – Use cases

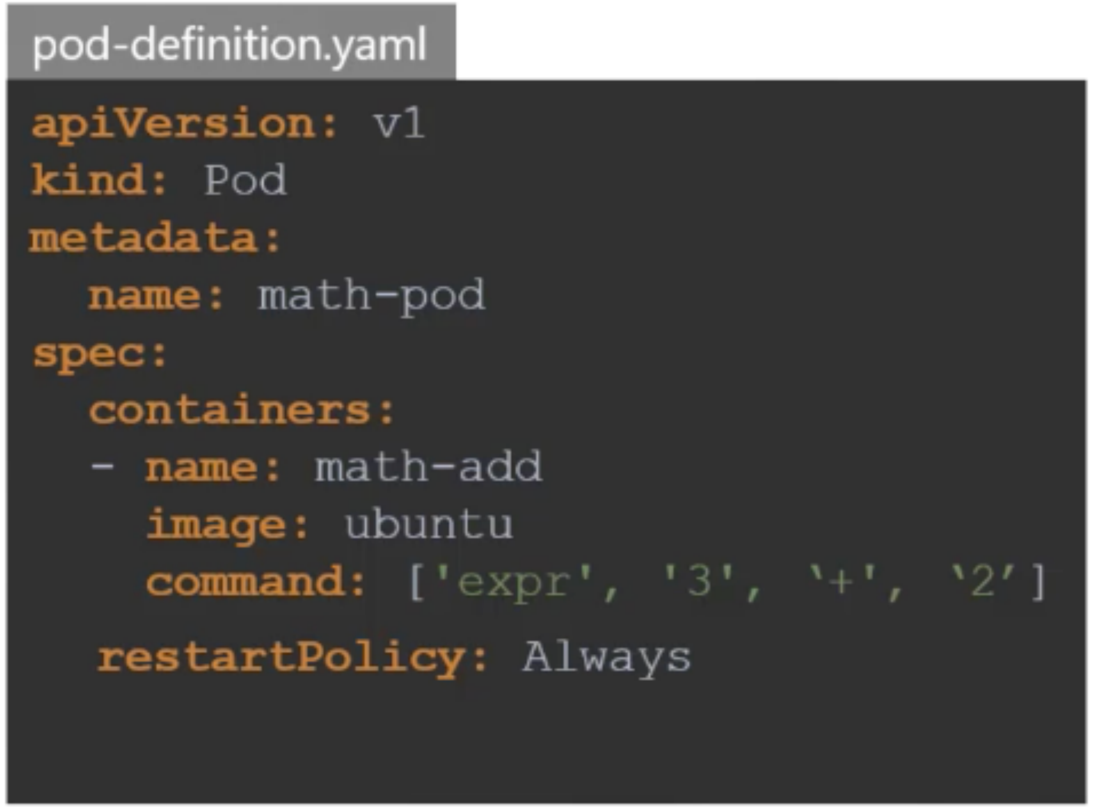
- The best use case for Kubernetes jobs are,
- **Batch processing:** Let’s say you want to run a batch task once a day or during a specific schedule. It could be something like reading files from storage or a database and feed them to a service to process the files.
- **Operations/ad-hoc tasks:** Let’s say you want to run a script/code which runs a database cleanup activity or to even backup a kubernetes cluster itself.

Restart Policy – Never vs Always



pod-definition.yaml

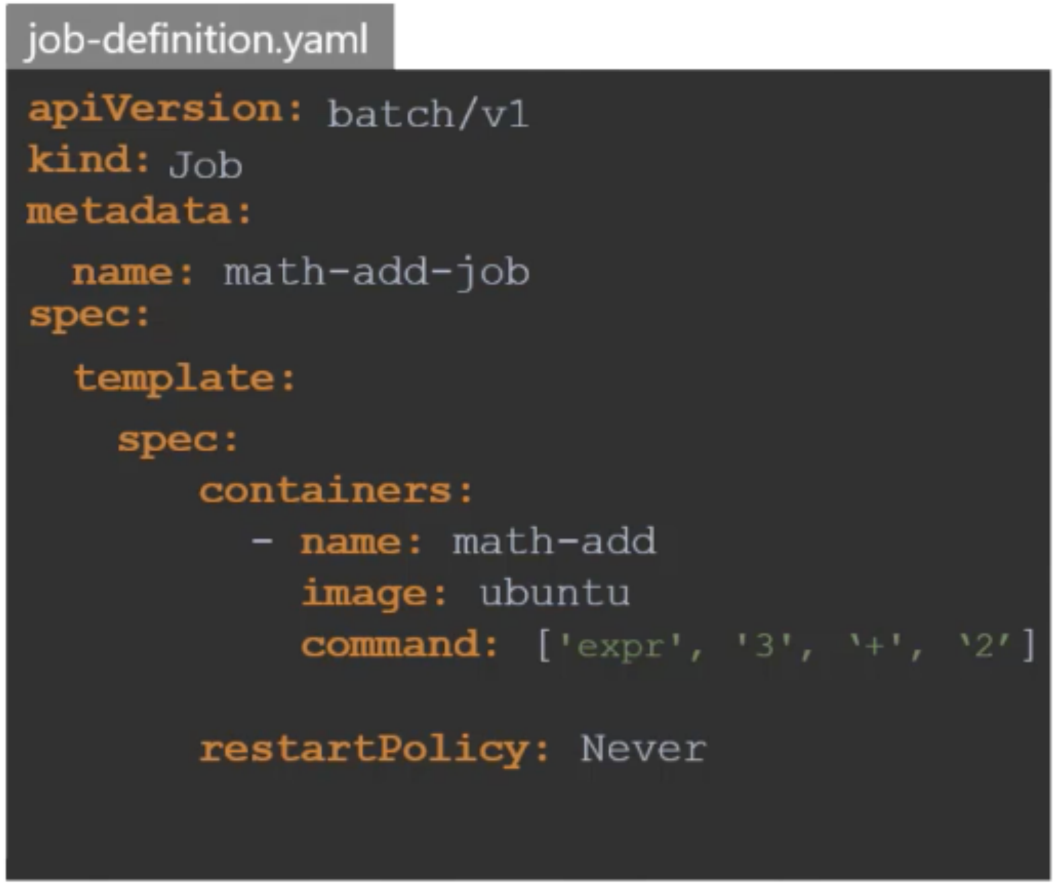
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: math-pod
spec:
  containers:
  - name: math-add
    image: ubuntu
    command: ['expr', '3', '+', '2']
  restartPolicy: Never
```



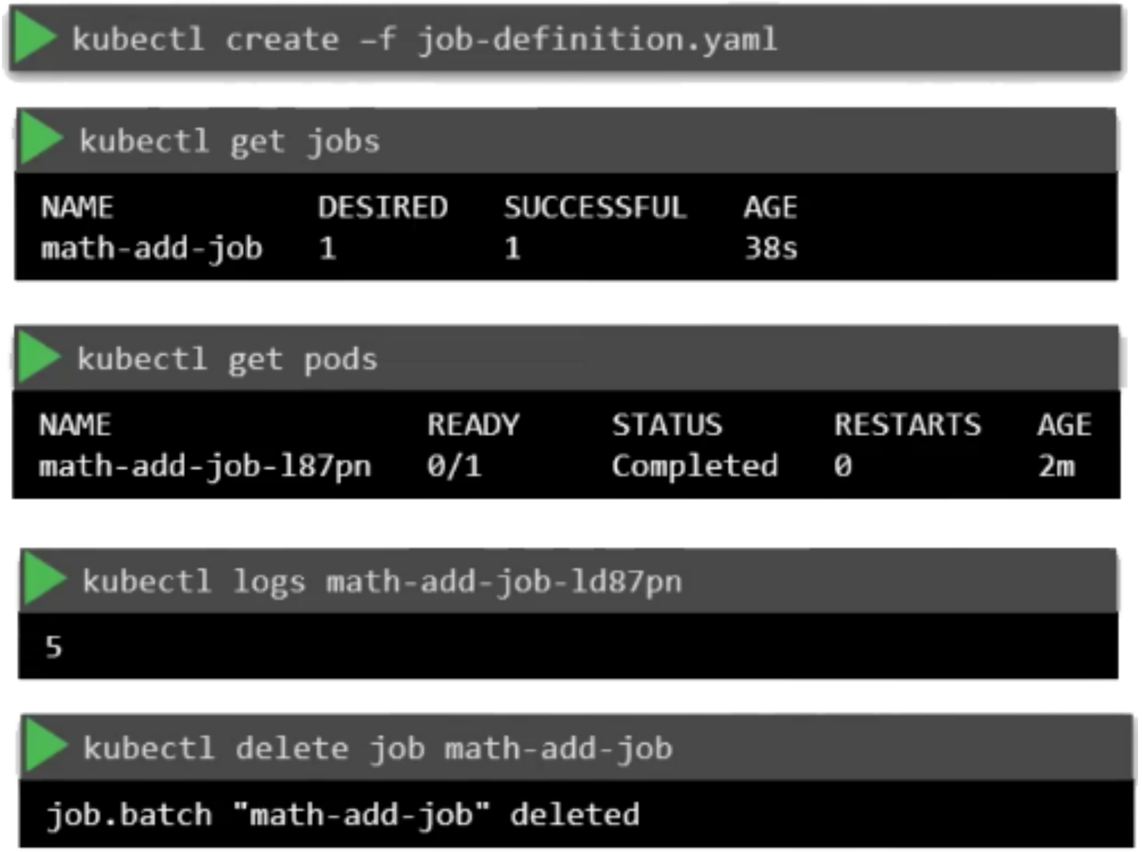
pod-definition.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: math-pod
spec:
  containers:
  - name: math-add
    image: ubuntu
    command: ['expr', '3', '+', '2']
  restartPolicy: Always
```

Create, View & Delete



```yaml
job-definition.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: math-add-job
spec:
  template:
    spec:
      containers:
        - name: math-add
          image: ubuntu
          command: ['expr', '3', '+', '2']

      restartPolicy: Never
```



```
kubectl create –f job-definition.yaml
```

```
kubectl get jobs
NAME           DESIRED   SUCCESSFUL   AGE
math-add-job   1         1            38s
```

```
kubectl get pods
NAME                 READY    STATUS      RESTARTS   AGE
math-add-job-187pn   0/1      Completed   0          2m
```

```
kubectl logs math-add-job-ld87pn
5
```

```
kubectl delete job math-add-job
job.batch "math-add-job" deleted
```

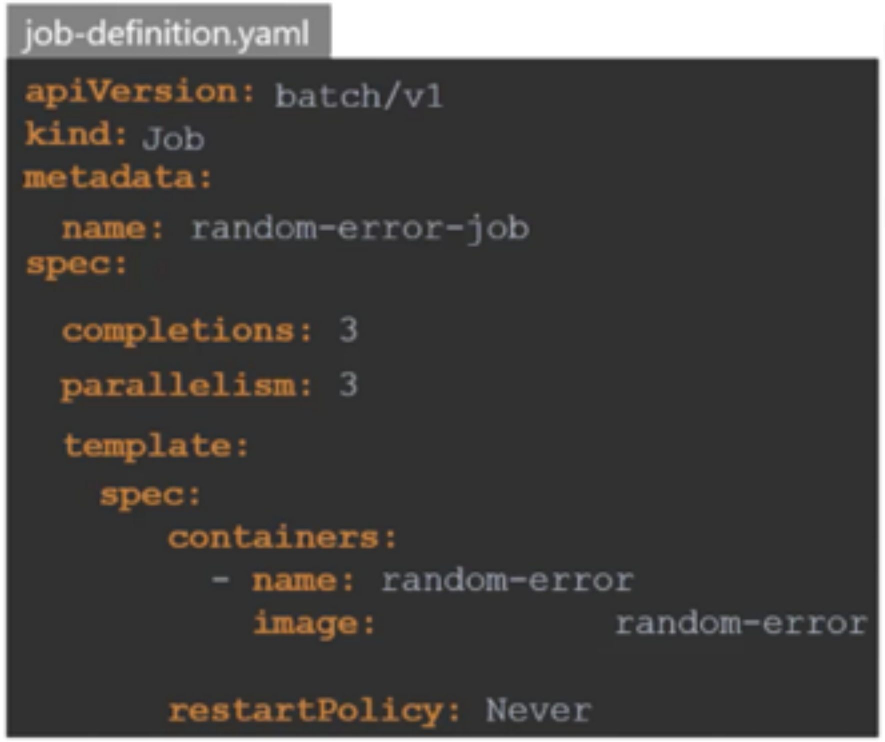
Multiple Job Pods and Parallelism

- When a job is deployed you can make it run on multiple pods with parallelism.
- For example, in a job if you want to run 6 pods and run 2 pods in parallel, you need to add the following two parameters to your job manifest.

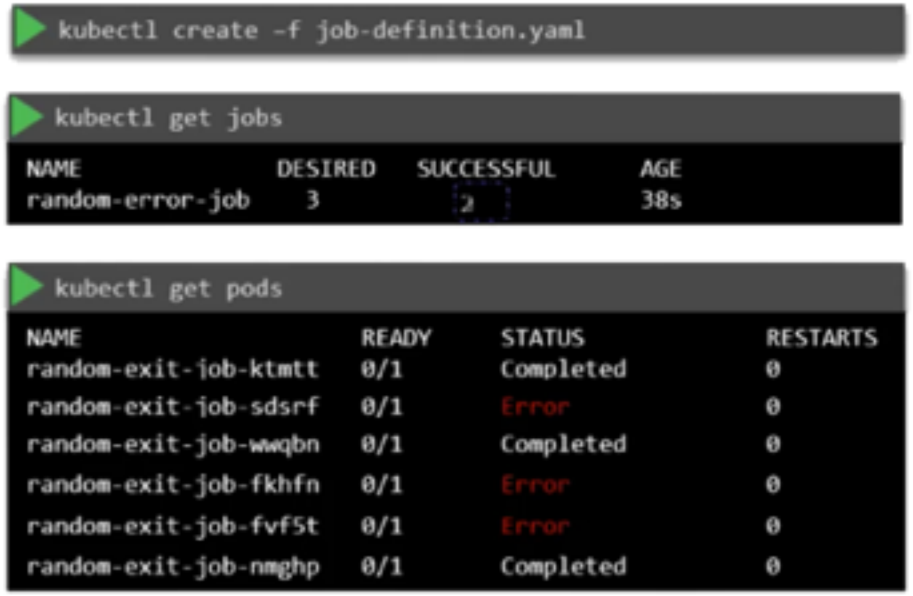
  completions: 3

  parallelism: 3
- The job will run 2 pods in parallel for 3 times to achieve 6 completions.
- Here is the manifest file with those parameters.

Multiple Job Pods and Parallelism


job-definition.yaml

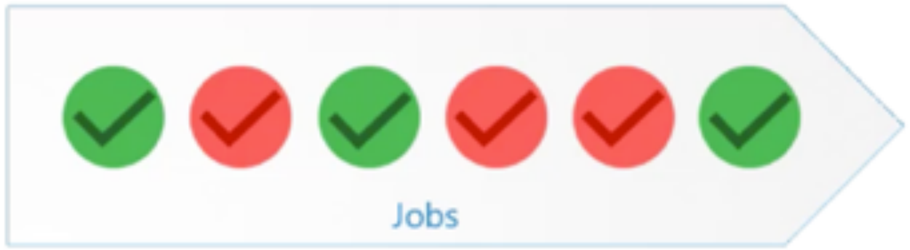
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: random-error-job
spec:
  completions: 3
  parallelism: 3
  template:
    spec:
      containers:
        - name: random-error
          image:          random-error
      restartPolicy: Never
```



```
▶ kubectl create -f job-definition.yaml
```

```
▶ kubectl get jobs
NAME               DESIRED   SUCCESSFUL   AGE
random-error-job     3          2         38s
```

```
▶ kubectl get pods
NAME                      READY    STATUS       RESTARTS
random-exit-job-ktmtt     0/1      Completed    0
random-exit-job-sdsrf     0/1      Error        0
random-exit-job-wwqbn     0/1      Completed    0
random-exit-job-fkhfn     0/1      Error        0
random-exit-job-fvf5t     0/1      Error        0
random-exit-job-nmghp     0/1      Completed    0
```



kubernetes

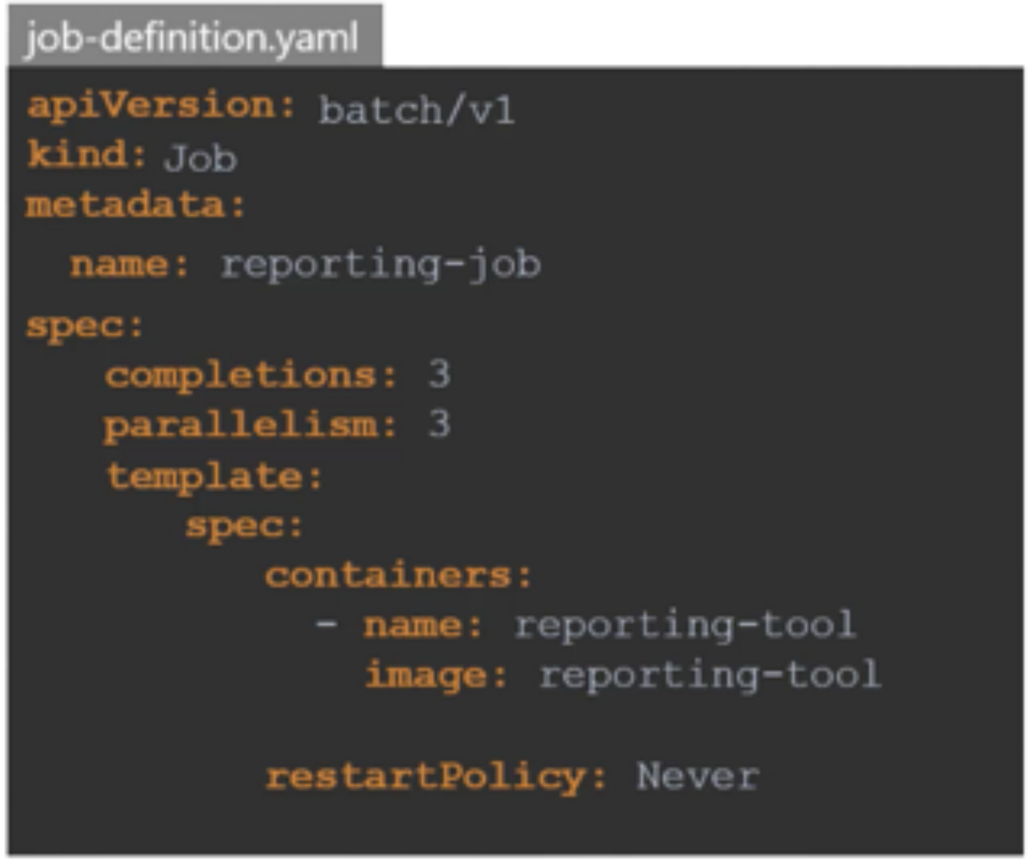
Cron Jobs

Application Lifecycle Management

Crobjob

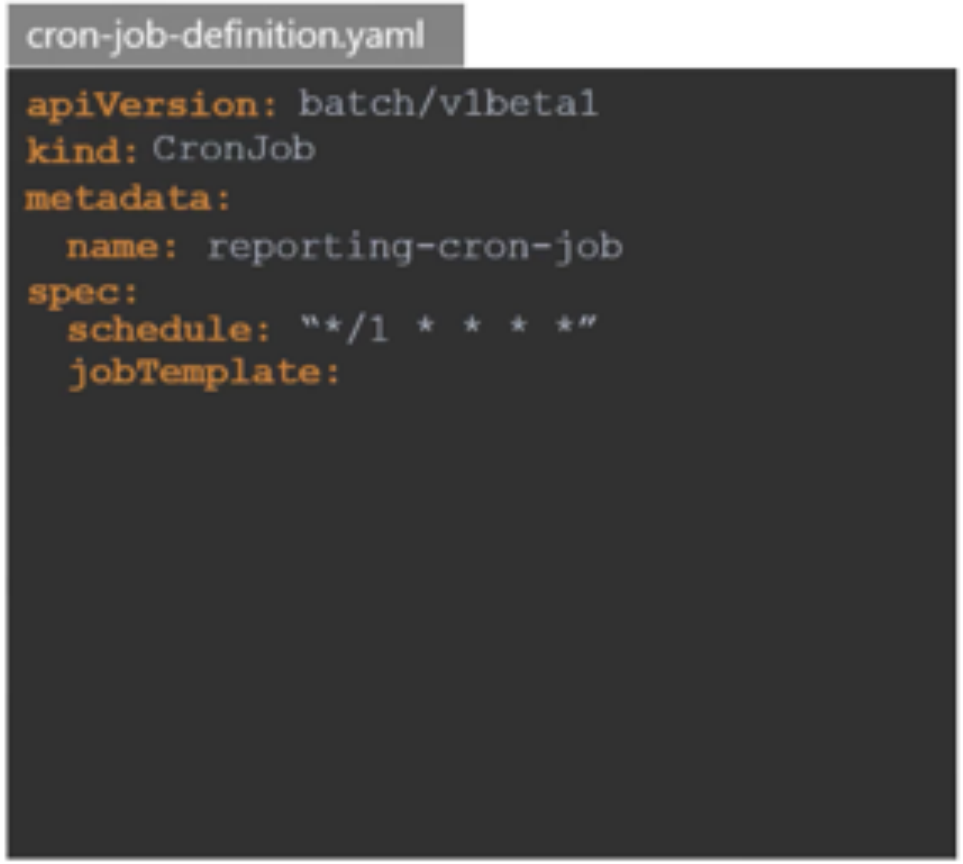
- What if you want to run a batch job on specific schedules, for example, every 2 hours. You can create a Kubernetes cronjob with a cron expression.
  - The job will automatically kick in as per the schedule you mention in the job.
- A CronJob is basically a Kubernetes Job with time-based scheduling and some specific parameters to handle failure.

CronJob



job-definition.yaml

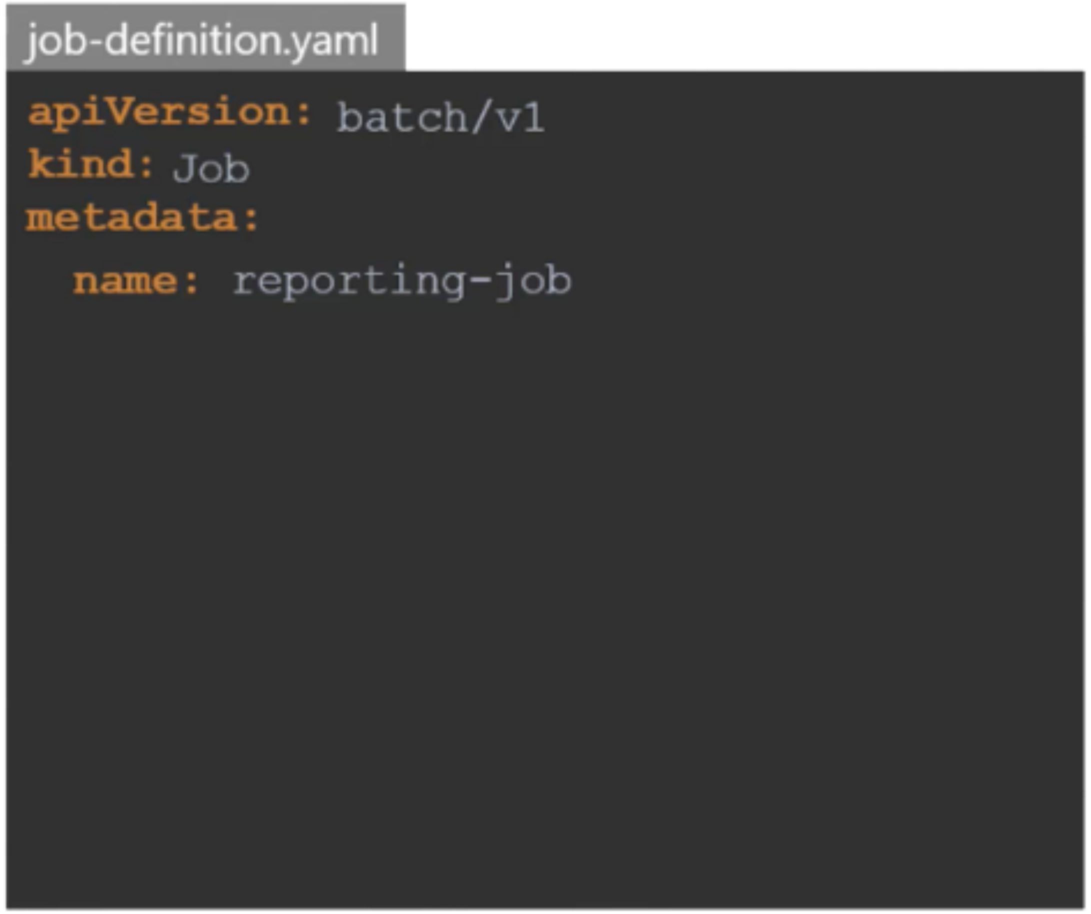
```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: reporting-job
spec:
    completions: 3
    parallelism: 3
    template:
        spec:
            containers:
              - name: reporting-tool
                image: reporting-tool

            restartPolicy: Never
```



cron-job-definition.yaml

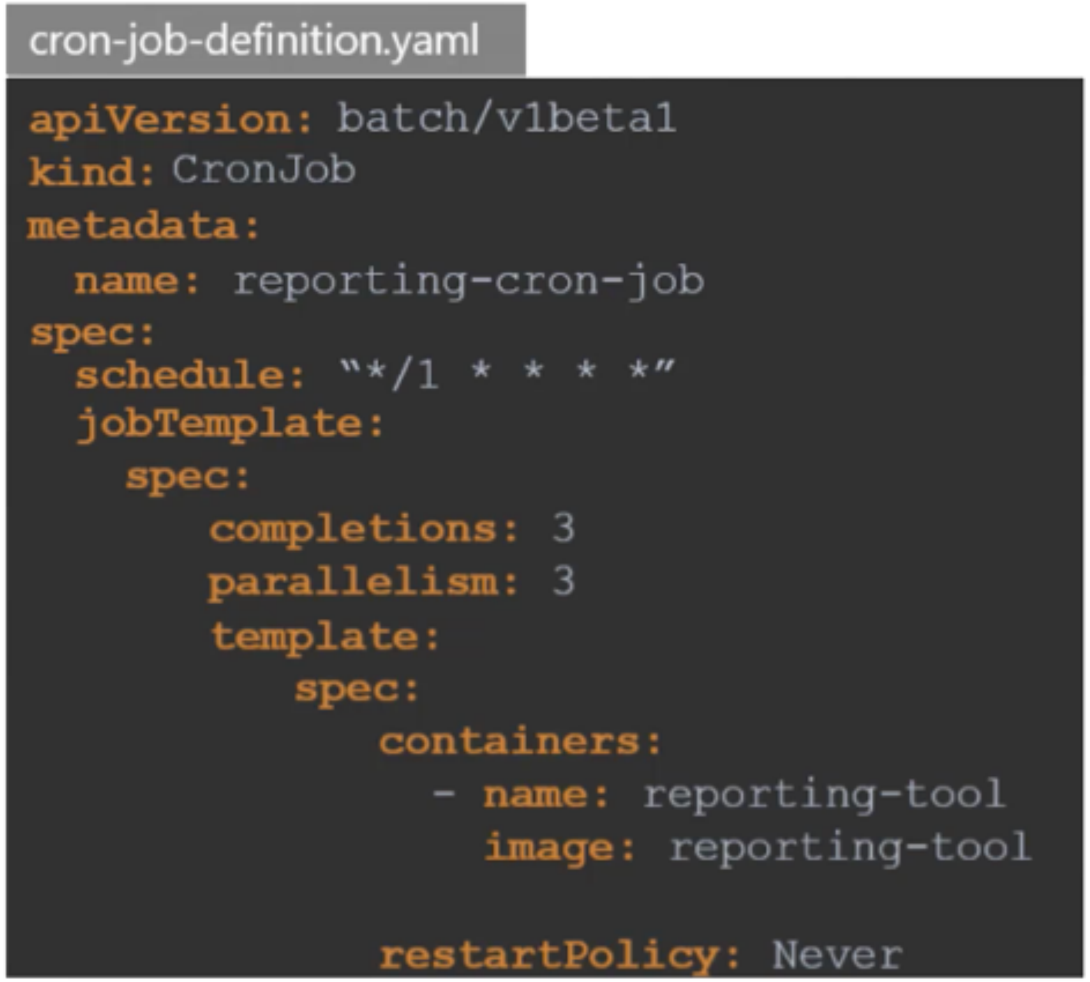
```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: reporting-cron-job
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
```

CronJob



job-definition.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: reporting-job
```



cron-job-definition.yaml

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: reporting-cron-job
spec:
  schedule: “*/1 * * * *”
  jobTemplate:
    spec:
        completions: 3
        parallelism: 3
        template:
            spec:
                containers:
                  - name: reporting-tool
                    image: reporting-tool

                restartPolicy: Never
```

kubernetes

Managing Application Logs

Monitoring & Logging

Application Logs

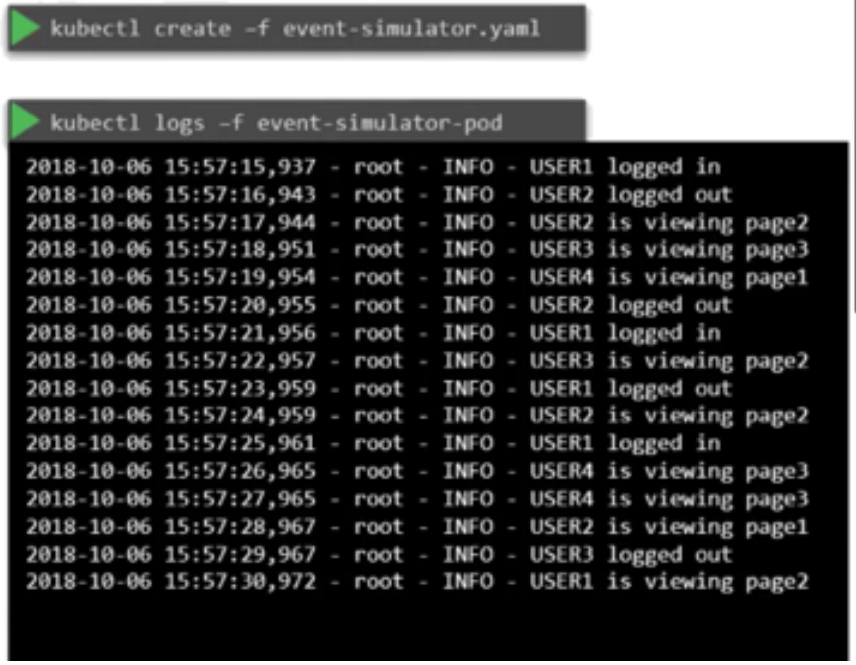
- Application logs are the logs from the applications that run on Kubernetes.
- The data stored in these logs consists of the information that your applications output as they run.
- Typically, this data is written to stdout inside the container where the application runs.

Viewing Application Logs

- There are two main ways to interact with application log data.
- The first is to run a command like

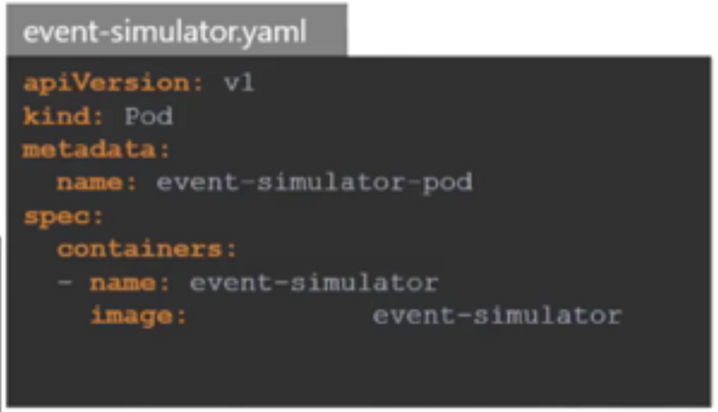
kubectl logs pod-name

where "pod-name" is the name of the pod that hosts the application whose logs you want to access.



```
kubectl create -f event-simulator.yaml
```

```
kubectl logs -f event-simulator-pod
2018-10-06 15:57:15,937 - root - INFO - USER1 logged in
2018-10-06 15:57:16,943 - root - INFO - USER2 logged out
2018-10-06 15:57:17,944 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:18,951 - root - INFO - USER3 is viewing page3
2018-10-06 15:57:19,954 - root - INFO - USER4 is viewing page1
2018-10-06 15:57:20,955 - root - INFO - USER2 logged out
2018-10-06 15:57:21,956 - root - INFO - USER1 logged in
2018-10-06 15:57:22,957 - root - INFO - USER3 is viewing page2
2018-10-06 15:57:23,959 - root - INFO - USER1 logged out
2018-10-06 15:57:24,959 - root - INFO - USER2 is viewing page2
2018-10-06 15:57:25,961 - root - INFO - USER1 logged in
2018-10-06 15:57:26,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:27,965 - root - INFO - USER4 is viewing page3
2018-10-06 15:57:28,967 - root - INFO - USER2 is viewing page1
2018-10-06 15:57:29,967 - root - INFO - USER3 logged out
2018-10-06 15:57:30,972 - root - INFO - USER1 is viewing page2
```



event-simulator.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: event-simulator-pod
spec:
  containers:
  - name: event-simulator
    image:          event-simulator
```

Storing & Analyzing Application Logs

- Suppose you want to store logs persistently and analyze them systematically.
- In that case, you’re better served by using an external logging tool like IBM Log Analysis with LogDNA to collect and interpret the logs.
- The easiest way to go about this is to run a so-called sidecar container, which runs alongside the application, collects its logs, and makes them available to an external logging tool.

kubernetes

Daemon Sets & Stateful Sets

Monitoring & Logging

DaemonSet - When Your Pods Must Be Everywhere

- A DaemonSet is a controller that ensures that the pod runs on all the nodes of the cluster.
  - If a node is added/removed from a cluster, DaemonSet automatically adds/deletes the pod.
- Some typical use cases of a DaemonSet is to run cluster level applications like:
  - Monitoring Services
  - Logs Collection Daemon
  - Cluster services like kube-proxy
  - Security solutions like antivirus, intrusion detection, and image scanning

StatefulSet - When Your Pods Are Unique

- A StatefulSet is a controller that helps you deploy and scale groups of Kubernetes pods.
- When using Kubernetes, most of the time you don’t care how your pods are scheduled, but sometimes you care that pods are deployed in order, that they have a persistent storage volume, or that they have a unique, stable network identifier across restarts and reschedules.
- In those cases, StatefulSets can help you accomplish your objective.
- To scale *stateful systems*, you need a **StatefulSet**.

StatefulSet - When Your Pods Are Unique

- Each pod created by the StatefulSet has an ordinal value (0 through # replicas - 1) and a stable network ID (which is statefulsetname-ordinal) assigned to it.
- You can also create a **VolumeClaimTemplate** in the manifest file that will create a persistent volume for each pod.
- When pods are deployed by a StatefulSet, they will go in order from 0 to the final pod and require that each pod is Running and Ready before creating the next pod.

StatefulSet - When to use?

- Some examples of reasons you'd use a StatefulSet include:
  - A Redis pod that has access to a volume, but you want it to maintain access to the same volume even if it is redeployed or restarted
  - A Cassandra cluster and have each node maintain access to its data
  - A webapp that needs to communicate with its replicas using known predefined network identifier