Kubernetes Workloads API: ReplicaSet, Deployment StatefulSet, DaemonSet, Job, CronJob

Check GitHub for helpful DevOps tools:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Ask Personal AI Document assistant to learn interactively (FASTER)!

1 Download PDF

2 Go to website

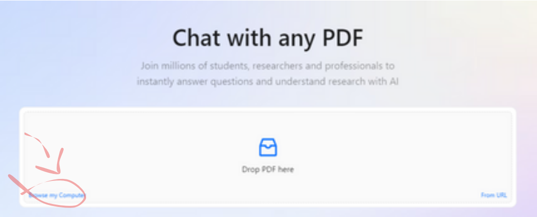
3 Browse file

4 Chat with Document

1 https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesWorkloads.pdf

Click there to go to ChatPdf website 2

3



Chat with any PDF

Join millions of students, researchers and professionals to instantly answer questions and understand research with AI

Drop PDF here

Browse my Computer

From URL

Ask questions about document! 4

Complety new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

https://www.hackthebox.com/

What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Workloads API: Intro

1) What are Kubernetes Workloads

The Kubernetes Workload API is a key component of Kubernetes, focusing on managing and orchestrating workloads (applications and services) that run on a Kubernetes cluster.

2) k8s Workloads

Pod - The smallest deployable unit in Kubernetes, representing one or more containers that share the same network namespace and storage volumes.

ReplicaSet - Ensures a specified number of identical Pod replicas are running at any given time.

Deployment - Manages stateless applications and provides declarative updates for Pods and ReplicaSets.

StatefulSet - Manages stateful applications where each instance (Pod) requires a unique identity or stable storage.

DaemonSet - Ensures that a copy of a Pod runs on all (or some) nodes in the cluster.

Job - Manages batch or one-off tasks that need to complete successfully.

CronJob - Extends Job by running tasks on a schedule.

Kubernetes Workload API: ReplicaSet

Download project repo

```
git clone https://github.com/vfarcic/kubernetes-demo
```

As covered in intro, ReplicaSet is responsible to assure that specified number of pods runs all the time.

apply

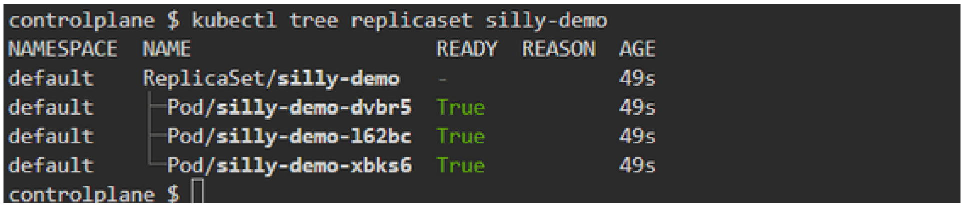
```
cd kubernetes-demo
kubectl apply --filename replicaset/base.yaml
```

Install krew and tree:

```
curl -O https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/Tools/InstallTree
bash InstallTree
source ~/.bashrc
```

Check ownership of resource. It’s easy to find out, that replicaset is responsible for creating pods (number of those pods is specified in yaml file)

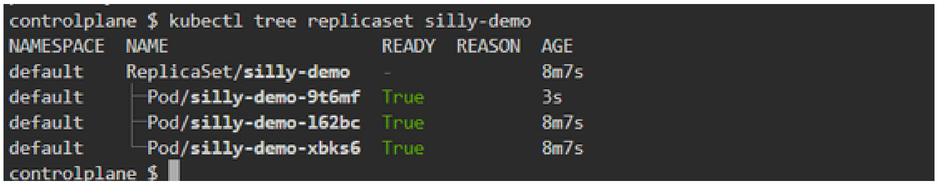
```
kubectl tree replicaset silly-demo
```



```
controlplane $ kubectl tree replicaset silly-demo
NAMESPACE  NAME                        READY  REASON  AGE
default    ReplicaSet/silly-demo       -              49s
default    ├─Pod/silly-demo-dvbr5      True           49s
default    ├─Pod/silly-demo-162bc      True           49s
default    └─Pod/silly-demo-xbks6      True           49s
controlplane $ 
```

Delete one of the pods and check what will happen:

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



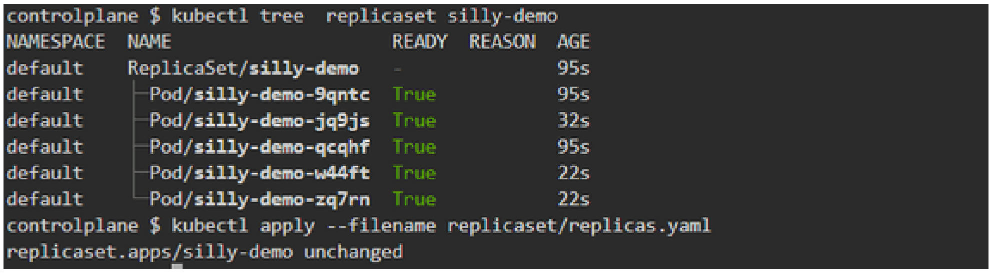
```
controlplane $ kubectl tree replicaset silly-demo
NAMESPACE  NAME                        READY  REASON  AGE
default    ReplicaSet/silly-demo       -              8m7s
default     ├─Pod/silly-demo-9t6mf     True           3s
default     ├─Pod/silly-demo-162bc     True           8m7s
default     └─Pod/silly-demo-xbks6     True           8m7s
controlplane $
```

A new Pod was created, and the total count remains stable. Now, update the number of Pods by applying a ReplicaSet YAML with the same name but a different replica count:

```
kubectl apply --filename replicaset/replicas.yaml
```

Check any changes

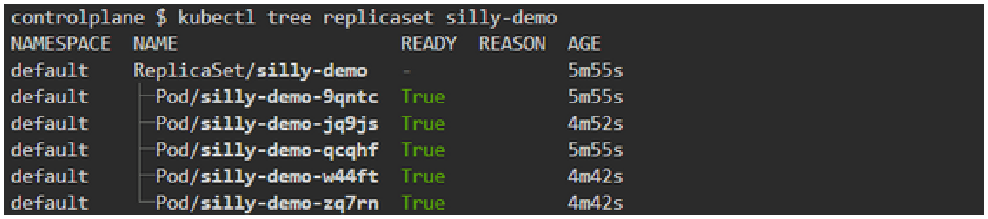
```
kubectl tree replicaset silly-demo
```



```
controlplane $ kubectl tree  replicaset silly-demo
NAMESPACE  NAME                        READY  REASON  AGE
default    ReplicaSet/silly-demo       -              95s
default     ├─Pod/silly-demo-9qntc     True           95s
default     ├─Pod/silly-demo-jq9js     True           32s
default     ├─Pod/silly-demo-qcqhf     True           95s
default     ├─Pod/silly-demo-w44ft     True           22s
default     └─Pod/silly-demo-zq7rn     True           22s
controlplane $ kubectl apply --filename replicaset/replicas.yaml
replicaset.apps/silly-demo unchanged
```

New pods were created, reflecting the replica count change. Next, update the image type:

```
kubectl apply --filename replicaset/image.yaml
kubectl tree replicaset silly-demo
```



```
controlplane $ kubectl tree replicaset silly-demo
NAMESPACE  NAME                        READY  REASON  AGE
default    ReplicaSet/silly-demo       -              5m55s
default     ├─Pod/silly-demo-9qntc     True           5m55s
default     ├─Pod/silly-demo-jq9js     True           4m52s
default     ├─Pod/silly-demo-qcqhf     True           5m55s
default     ├─Pod/silly-demo-w44ft     True           4m42s
default     └─Pod/silly-demo-zq7rn     True           4m42s
```

Pod images remain unchanged as ReplicaSet only manages pod count in real-time. To use a different image, delete all existing pods.

Kubernetes Workload API: Deployment

delete previous ReplicaSet

```
kubectl delete --filename replicaset/image.yaml
```

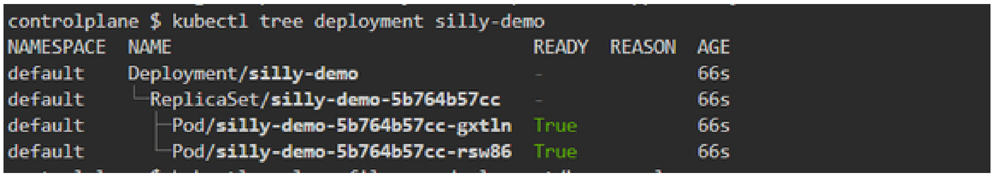
Deploymet manages rolling updates, scaling, and rollback for stateless applications

apply

```
kubectl apply --filename deployment/base.yaml
```

As you can see, Deployment make use of ReplicaSet to manage pods

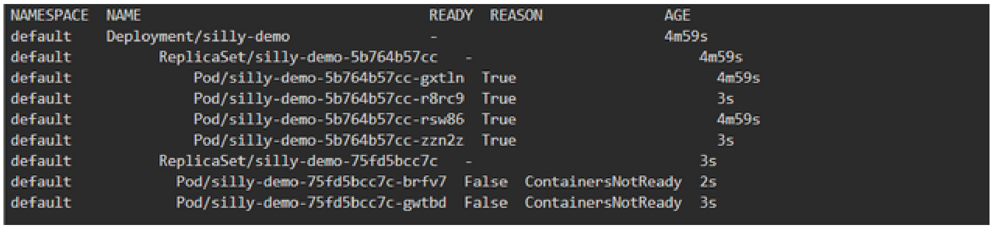
```
kubectl tree replicaset silly-demo
```



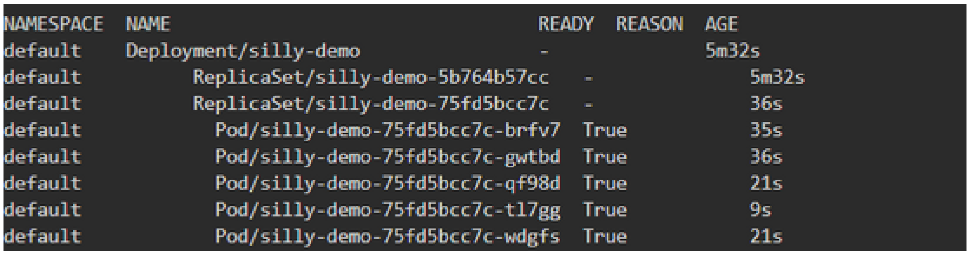
```
controlplane $ kubectl tree deployment silly-demo
NAMESPACE  NAME                                 READY  REASON  AGE
default    Deployment/silly-demo                -              66s
default    └ReplicaSet/silly-demo-5b764b57cc    -              66s
default      ├Pod/silly-demo-5b764b57cc-gxtln   True           66s
default      └Pod/silly-demo-5b764b57cc-rsw86   True           66s
```

Difference in behaviour can be seen if we apply same deployment with different image. Pods will be updated in rolling update manner, one by one.

```
kubectl apply --filename deployment/image.yaml \
&& watch kubectl tree deployment silly-demo
```



```
NAMESPACE  NAME                                    READY  REASON              AGE
default    Deployment/silly-demo                   -                          4m59s
default        ReplicaSet/silly-demo-5b764b57cc    -                              4m59s
default            Pod/silly-demo-5b764b57cc-gxtln  True                            4m59s
default            Pod/silly-demo-5b764b57cc-r8rc9  True                            3s
default            Pod/silly-demo-5b764b57cc-rsw86  True                            4m59s
default            Pod/silly-demo-5b764b57cc-zzn2z  True                            3s
default        ReplicaSet/silly-demo-75fd5bcc7c    -                              3s
default          Pod/silly-demo-75fd5bcc7c-brfv7  False  ContainersNotReady  2s
default          Pod/silly-demo-75fd5bcc7c-gwtbd  False  ContainersNotReady  3s
```

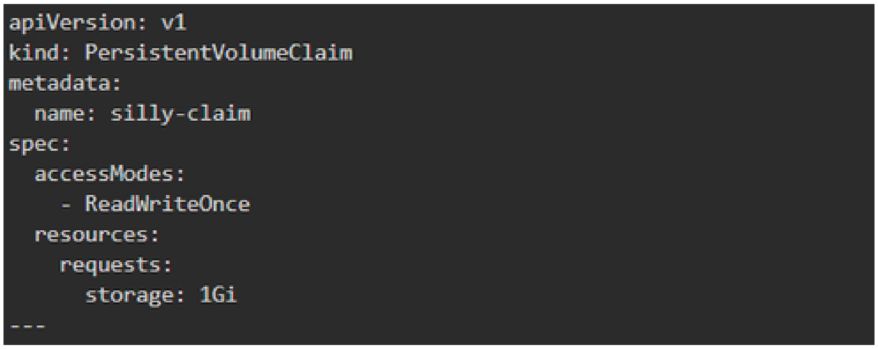


```
NAMESPACE  NAME                                   READY  REASON  AGE
default    Deployment/silly-demo                  -              5m32s
default          ReplicaSet/silly-demo-5b764b57cc   -                5m32s
default          ReplicaSet/silly-demo-75fd5bcc7c   -                36s
default            Pod/silly-demo-75fd5bcc7c-brfv7  True             35s
default            Pod/silly-demo-75fd5bcc7c-gwtbd  True             36s
default            Pod/silly-demo-75fd5bcc7c-qf98d  True             21s
default            Pod/silly-demo-75fd5bcc7c-tl7gg  True             9s
default            Pod/silly-demo-75fd5bcc7c-wdgfs  True             21s
```

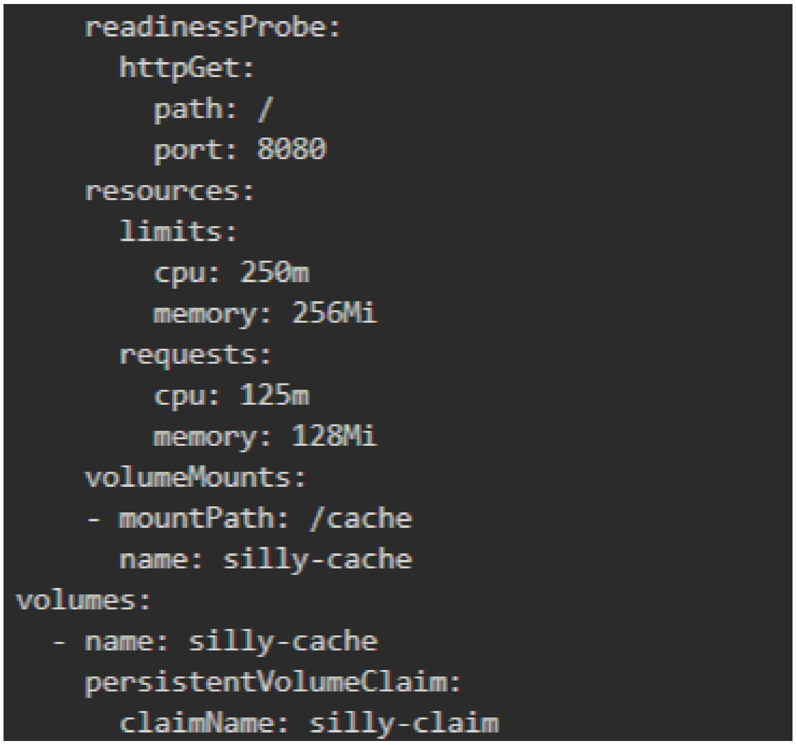
After a bit of a time, all pods from new replicaset got rolled out.

Deployment can manage PV, but you need to create PVC at first, then connect it to deployment and attach mount point on each container:

```
cat deployment/volume.yaml
```



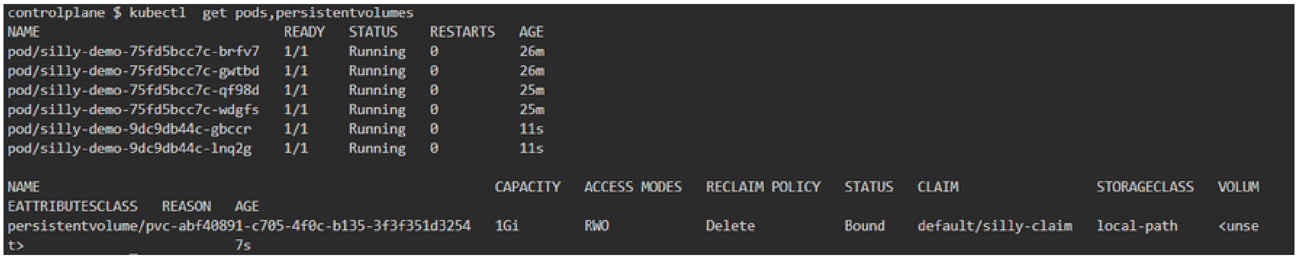
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: silly-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
```



```
      readinessProbe:
        httpGet:
          path: /
          port: 8080
      resources:
        limits:
          cpu: 250m
          memory: 256Mi
        requests:
          cpu: 125m
          memory: 128Mi
      volumeMounts:
      - mountPath: /cache
        name: silly-cache
volumes:
    - name: silly-cache
      persistentVolumeClaim:
        claimName: silly-claim
```

apply

```
kubectl apply --filename deployment/volume.yaml
kubectl  get pods,persistentvolumes
```



```
controlplane $ kubectl  get pods,persistentvolumes
NAME                              READY   STATUS    RESTARTS   AGE
pod/silly-demo-75fd5bcc7c-brfv7   1/1     Running   0          26m
pod/silly-demo-75fd5bcc7c-gwtbd   1/1     Running   0          26m
pod/silly-demo-75fd5bcc7c-qf98d   1/1     Running   0          25m
pod/silly-demo-75fd5bcc7c-wdgfs   1/1     Running   0          25m
pod/silly-demo-9dc9db44c-gbccr    1/1     Running   0          11s
pod/silly-demo-9dc9db44c-lnq2g    1/1     Running   0          11s

NAME                                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS   VOLUM
EATTRIBUTESCLASS   REASON   AGE
persistentvolume/pvc-abf40891-c705-4f0c-b135-3f3f351d3254   1Gi        RWO            Delete           Bound    default/silly-claim    local-path     <unse
t>                          7s
```

As you can see, 1 PV is created for all Pods. A Deployment can manage Pods with PVCs, but it doesn't provide guarantees about Pod identity or stable network identifiers. Delete deployment.

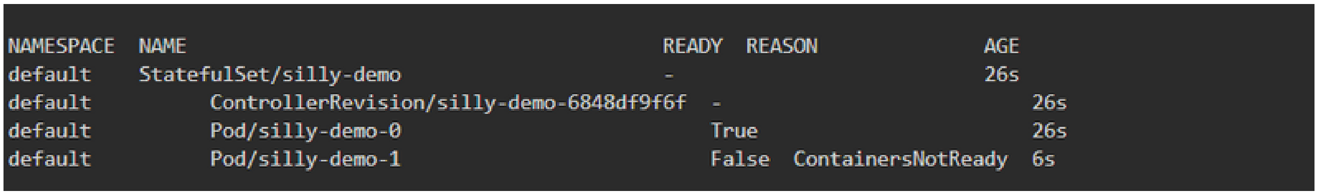
```
kubectl delete --filename deployment/volume.yaml
```

Kubernetes Workload API: StatefullSet

StatefullSet Use Case: Running databases or applications needing persistent storage or unique configurations. Unlike Deployments, StatefulSets automatically manage PersistentVolumeClaims for each Pod.

apply and check

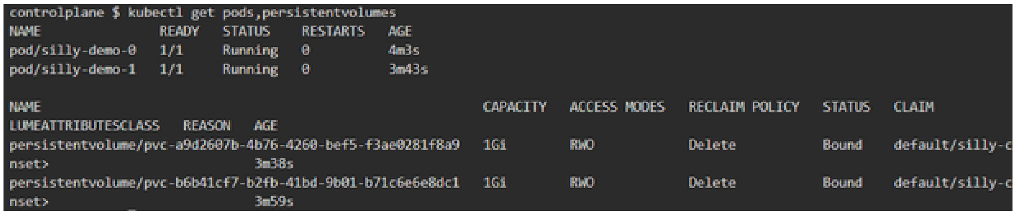
```
kubectl apply --filename statefulset/base.yaml
watch kubectl tree statefulset silly-demo
```



```
NAMESPACE  NAME                                         READY  REASON             AGE
default    StatefulSet/silly-demo                       -                         26s
default          ControllerRevision/silly-demo-6848df9f6f  -                           26s
default          Pod/silly-demo-0                          True                        26s
default          Pod/silly-demo-1                          False  ContainersNotReady  6s
```

Each created pod is numbered from 0 up. Check other specs:

```
kubectl get pods,persistentvolumes
```



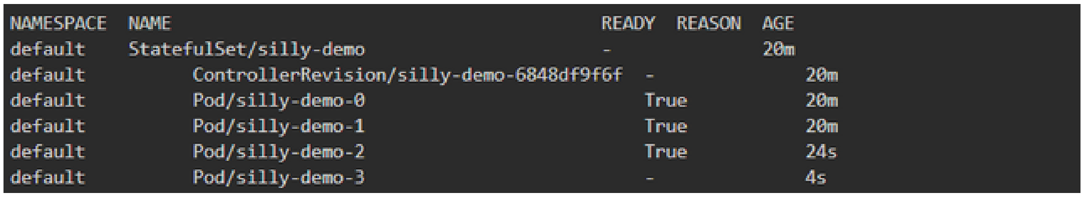
```
controlplane $ kubectl get pods,persistentvolumes
NAME               READY   STATUS    RESTARTS   AGE
pod/silly-demo-0   1/1     Running   0          4m3s
pod/silly-demo-1   1/1     Running   0          3m43s

NAME                                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
LUMEATTRIBUTESCLASS   REASON   AGE
persistentvolume/pvc-a9d2607b-4b76-4260-bef5-f3ae0281f8a9   1Gi        RWO            Delete           Bound    default/silly-c
nset>                          3m38s
persistentvolume/pvc-b6b41cf7-b2fb-41bd-9b01-b71c6e6e8dc1   1Gi        RWO            Delete           Bound    default/silly-c
nset>                          3m59s
```

As expected, each pod have its own volume. Notice that StatefullSet doesnt make use of ReplicaSet. Each pod is created separately.

Change number of pods and lets see what will happen:

```
kubectl apply --filename statefulset/replicas.yaml \
&& watch kubectl tree statefulset silly-demo
```



```
NAMESPACE  NAME                                    READY  REASON  AGE
default    StatefulSet/silly-demo                  -              20m
default          ControllerRevision/silly-demo-6848df9f6f  -              20m
default          Pod/silly-demo-0                          True           20m
default          Pod/silly-demo-1                          True           20m
default          Pod/silly-demo-2                          True           24s
default          Pod/silly-demo-3                          -              4s
```

New pods are created, one after another with according numeration. After deletion of statefullset, each pod and related PV will be deleted. From pods with biggest number, to the lowest.

```
kubectl delete --filename statefulset/replicas.yaml
```

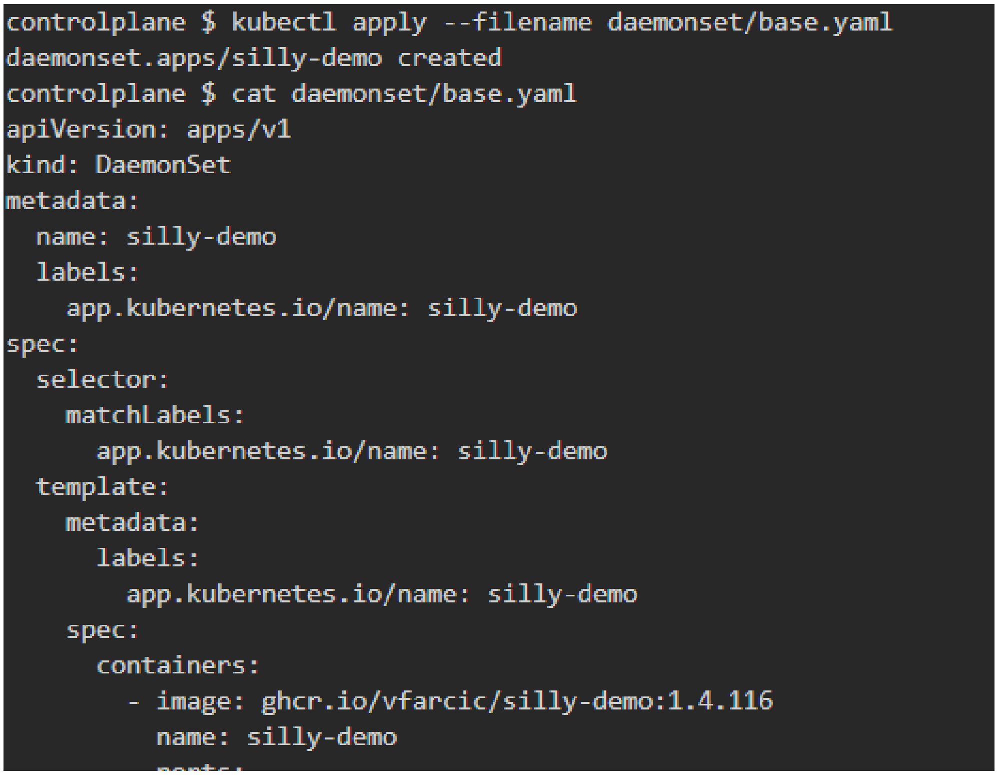
Kubernetes Workload API: DaemonSet

DaemonSet use case: Running node-specific services like logging, monitoring agents, or network daemons.

apply and check manifest:

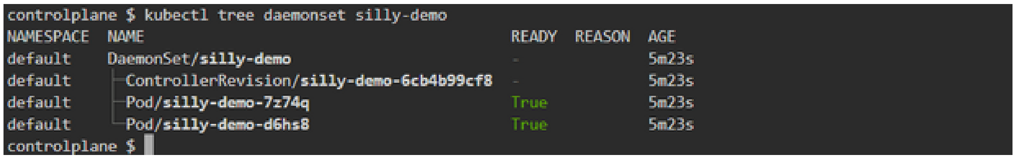
```
kubectl apply --filename daemonset/base.yaml
cat daemonset/base.yaml
```

An important point about DaemonSet is that its manifest does not define the number of replicas.



```
controlplane $ kubectl apply --filename daemonset/base.yaml
daemonset.apps/silly-demo created
controlplane $ cat daemonset/base.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: silly-demo
  labels:
    app.kubernetes.io/name: silly-demo
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: silly-demo
  template:
    metadata:
      labels:
        app.kubernetes.io/name: silly-demo
    spec:
      containers:
        - image: ghcr.io/vfarcic/silly-demo:1.4.116
          name: silly-demo
          ports:
```

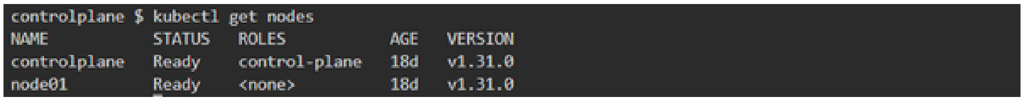
```
kubectl tree daemonset silly-demo
```



```
controlplane $ kubectl tree daemonset silly-demo
NAMESPACE  NAME                                     READY  REASON  AGE
default    DaemonSet/silly-demo                     -              5m23s
default    ├─ControllerRevision/silly-demo-6cb4b99cf8  -              5m23s
default    ├─Pod/silly-demo-7z74q                   True           5m23s
default    └─Pod/silly-demo-d6hs8                   True           5m23s
controlplane $
```

Even without defining replica count, DaemonSet creates Pods. What determines their number? The number of cluster nodes.

```
kubectl get nodes
```



```
controlplane $ kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
controlplane   Ready    control-plane   18d   v1.31.0
node01         Ready    <none>          18d   v1.31.0
```

The number of nodes in the cluster is directly correlated with the number of pods created by a DaemonSet. For each node in the cluster, the DaemonSet ensures that exactly one pod is scheduled and running. As nodes are added or removed, the number of pods managed by the DaemonSet automatically adjusts to match the updated node count.

Delete workload.

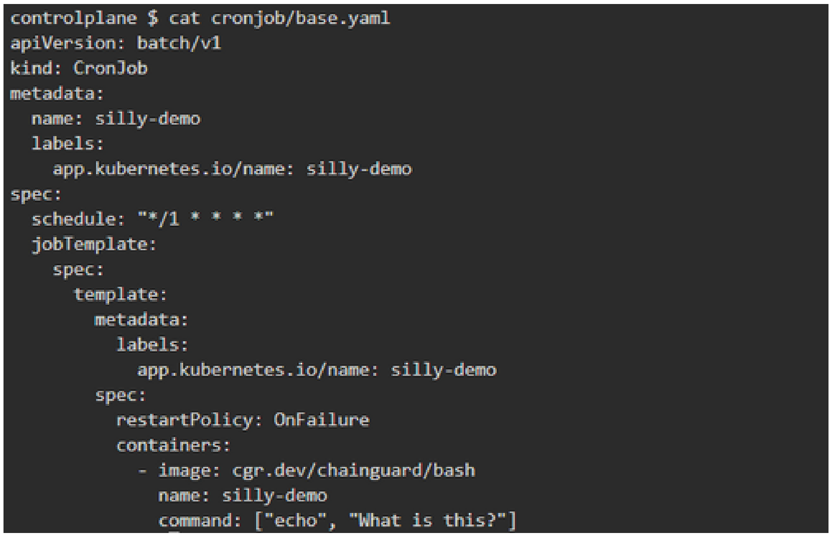
```
kubectl delete --filename daemonset/base.yaml
```

Kubernetes Workload API: CronJob

Job use case: Periodic tasks like backups, reports, or maintenance scripts.

Check manifest. CronJob manifest is same as Job, with little adjustment. There is an CronJob defined. Job will be ran every 1 minute:

```
cat cronjob/base.yaml
```

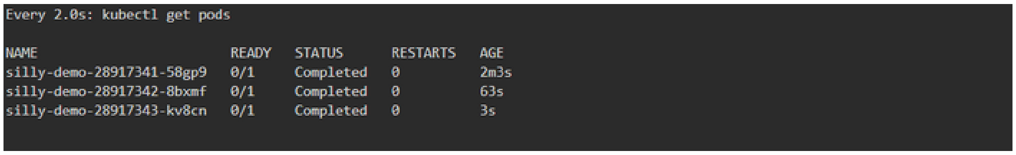


```
controlplane $ cat cronjob/base.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: silly-demo
  labels:
    app.kubernetes.io/name: silly-demo
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            app.kubernetes.io/name: silly-demo
        spec:
          restartPolicy: OnFailure
          containers:
            - image: cgr.dev/chainguard/bash
              name: silly-demo
              command: ["echo", "What is this?"]
```

apply

```
kubectl apply --filename cronjob/base.yaml
watch kubectl get pods
```

Pods are created every minute:



```
Every 2.0s: kubectl get pods

NAME                         READY   STATUS      RESTARTS   AGE
silly-demo-28917341-58gp9    0/1     Completed   0          2m3s
silly-demo-28917342-8bxmf    0/1     Completed   0          63s
silly-demo-28917343-kv8cn    0/1     Completed   0          3s
```

common troubleshooting

1) ReplicaSet Not Scaling Pods

Cause: Misconfigured ReplicaSet spec or insufficient resources.
Solution: Check kubectl get replicaset for replicas and resource availability. Review events with kubectl describe replicaset.

2) Deployment Not Updating

Cause: Spec issues, image errors, or rollout failures.
Solution: Inspect Deployment with kubectl describe. Check image errors, rollout strategy, or undo with kubectl rollout undo.

3) StatefulSet Pods Stuck in Pending

Cause: Unbound volumes or insufficient resources.
Solution: Check PVCs with kubectl get pvc, and node resources with kubectl describe nodes. Verify storage and events.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27

Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

https://kubernetes.io/docs/setup/

Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

https://github.com/MichaelRobotics

PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!