Deploy Replica Set and Replication Controller, and deployment. Also learn the advantages and disadvantages of each.

Deployments

A deployment in Kubernetes is a method of managing and scaling applications.

It allows you to define the desired state of your application, including the number of replicas of your application that should be running, and the specific configuration of your containers.

Kubernetes then automatically manages the deployment, ensuring that the desired state is maintained even in the face of failures or changes in demand.

For example, let's say you have a web application that you want to run on a Kubernetes cluster. You could create a deployment that specifies that you want three replicas of your application to be running at all times and that each replica should be based on a specific Docker image.

The deployment would then ensure that three instances of your application are running and available to handle incoming requests, and would automatically replace any instances that fail.

Additionally, if you wanted to update your application to a new version, you could simply update the Docker image in your deployment and Kubernetes would automatically roll out the update to all of your replicas(we will talk about the rollout, rollback, and update), ensuring that your application stays up and running.

Creating a Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web-app-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: my-web-app
  template:
    metadata:
      labels:
        app: my-web-app
    spec:
      containers:
      - name: my-web-app
        image: my-web-app:v1.0
        ports:
        - containerPort: 80
```

After creating the YAML and saving the configuration for the deployment you can run the 'kubectl create' command

```
kubectl create my-web-app-deployment.yaml
```

you can now view the deployments by running the command

```
kubectl get deployments
```

Benefits of Deployments

There are several benefits to using deployments in Kubernetes:

1. **Scalability**: Deployments make it easy to scale up or down the number of replicas of your application based on changing demand, ensuring that your application remains available and responsive to users.

2. **Resilience**: Deployments ensure that your application is resilient to failures by automatically replacing any containers that fail, keeping your application up and running even in the face of hardware or network failures.

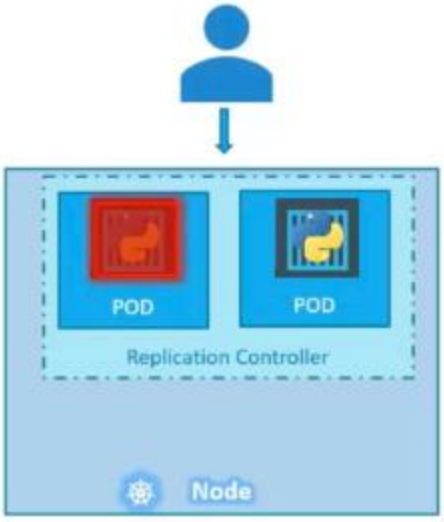
3. **Easy updates:** Deployments allow you to easily roll out updates to your application by simply updating the Docker image in your deployment, without having to manually update each individual container.

4. **Rollback**: In case of an issue with a new update, you can easily revert to a previous version by rolling back your deployment to a previous revision, preserving the stability and reliability of your application.

5. **Centralized management**: Deployments provide a centralized and consistent way of managing the desired state of your applications across your entire cluster, making it easier to maintain and update your applications over time.

Replication controller

Replication Controllers are a component of the Kubernetes orchestration platform, and This is the older technology being replaced by ReplicaSet. they serve several important purposes



1. **High Availability:** Replication Controllers ensure that the desired number of replicas of a particular application or service is always running and available, even in the face of failures or node crashes.

2. **Auto-Scaling:** Replication Controllers can automatically scale the number of replicas up or down based on load, providing an easy way to manage the resources required by an application.

3. **Load Balancing**: Replication Controllers distribute the workload across multiple replicas, providing load balancing and increasing the overall reliability and performance of the application.

4. **Rollouts and Rollbacks**: Replication Controllers can be used to manage updates to applications, allowing for phased rollouts, rollbacks, and automatic rollbacks in the case of failures.

5. **Self-Healing:** Replication Controllers monitor the health of replicas and automatically replace failed replicas, ensuring that the desired state of the application is always maintained.

In summary, Replication Controllers help ensure high availability, scalability, and resiliency for applications running in a Kubernetes cluster.

Creating a ReplicationController

let's create a ReplicationController using a YAML file

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: onai57
spec:
  replicas: 3
  selector:
    app: onai57
  template:
    metadata:
      name: onai57
      labels:
        app: onai57
    spec:
      containers:
        - name: onai57
          image: onai/54
          ports:
            - containerPort: 80
```

After creating the replication controller in the YAML file lets then create the replication controller

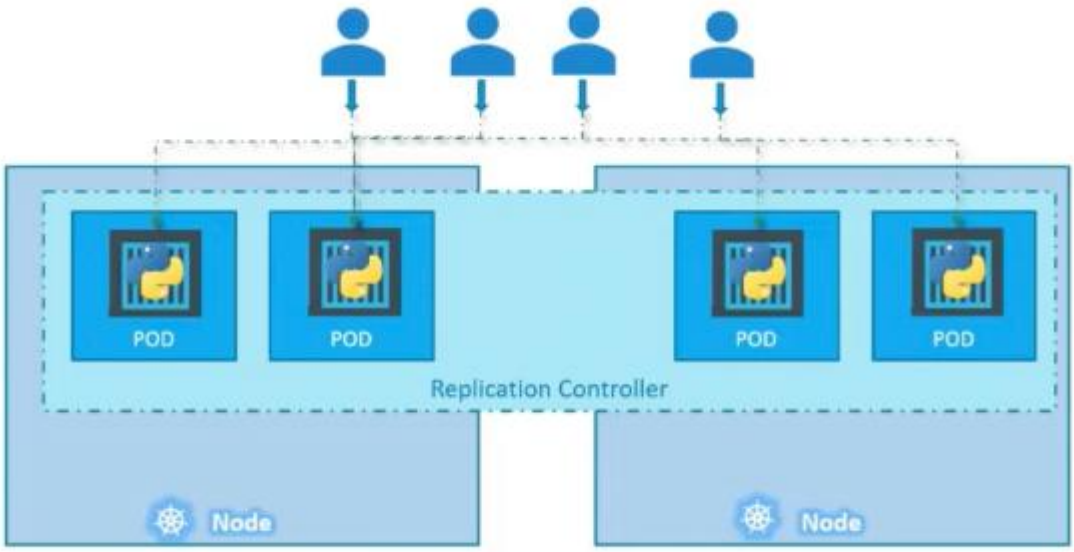
```
kubectl create replicationcontroller.yaml
```

When the ReplicationController is being created it is first creating all the pods defined in the replicas section which is 3. You can view the created ReplicationController by running the command

```
kubectl get rc
```

```
> kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY     AGE
myapp-rc   3         3         3         19s
```

Remember in the new updates of Kubernetes the ReplicationController is being deprecated and instead use the ReplicaSet.



ReplicaSet

ReplicaSet is a component in the Kubernetes orchestration platform used to manage the desired state of replicated applications. It ensures that a specified number of replicas of a particular application or service are running and available at all times, even in the face of failures or node crashes. ReplicaSets use more advanced set-based selectors compared to Replication Controllers,

allowing for more fine-grained control over which pods are managed. ReplicaSets also offer more sophisticated update strategies and fine-grained control over rollouts, making it easier to manage updates to applications. Additionally, ReplicaSets are more performant and scalable than Replication Controllers, providing a more capable solution for managing replicated applications in Kubernetes.

The difference in creating a ReplicaSet is the API version, and the selector match labels: here it can be used to manage also pods that were already created, and the labels and selector define which app to monitor.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sample
  template:
    metadata:
      labels:
        app: sample
    spec:
      containers:
      - name: sample
        image: nginx:latest
        ports:
        - containerPort: 80
```

you can then use the ‘kubectl apply’ to create the ReplicaSet

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

Replication Controller and ReplicaSet are both components in Kubernetes used to manage the desired state of replicated applications. The main differences between the two are:

1. **Selector support:** Replication Controllers use simple label selectors to identify the pods they manage, while ReplicaSets support more advanced set-based selectors.
2. **Update Behavior:** Replication Controllers are more basic and do not support sophisticated update strategies, while ReplicaSets offer more fine-grained control over updates and rollouts.
3. **Scalability:** ReplicaSets are more scalable and can handle larger numbers of replicas compared to Replication Controllers.
4. **Performance**: ReplicaSets are more performant than Replication Controllers, as they use more efficient algorithms for selecting and managing replicas.

Overall, ReplicaSets are a more advanced and capable replacement for Replication Controllers, offering a more feature-rich and scalable solution for managing replicated applications in Kubernetes.

How to scale ReplicaSet

The first way in which we can scale a ReplicaSet is we change the YAML definition that we first used, previously in our YAML file, we had 2 replicas next we will add the replicas to 6.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-replicaset
spec:
  replicas: 6
```

```
  selector:
    matchLabels:
      app: sample
  template:
    metadata:
      labels:
        app: sample
    spec:
      containers:
      - name: sample
        image: nginx:latest
        ports:
        - containerPort: 80
```

After editing the YAML file we can be able to scale the application by running the following commands

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

This will go and overwrite the first ReplicaSet created and bring in the new scaled version we can check for the pods by running

```
kubectl get pods
```

The second way we can scale the ReplicaSet is by running the ‘kubectl scale’ command.

```
kubectl scale replicasets<replicaset-name> --replicas=<number-of-replicas>
```

In order to scale to 6 replicas we can run

```
kubectl scale replicasets sample-replicaset --replicas=6
```

you can then view the number of pods to see if the pods were scaled

```
kubectl get pods
```

Namespaces

Using namespaces allows you to split complex systems with numerous components into smaller distinct groups.

They can also be used for separating resources in a multi-tenant environment, splitting up resources into production, development, and QA environments, or in any other way you may need.

Resource names only need to be unique within a namespace.

Two different namespaces can contain resources of the same name. But, while most types of resources are namespaced, a few aren't. One of them is the Node resource, which is global and not tied to a single namespace. You'll learn about other cluster-level resources in later chapters.

Namespaces in Kubernetes are a way of partitioning a single cluster into multiple virtual clusters, each with its own resources, network policies, and access controls.

Namespaces provide a way to organize and isolate resources within a single cluster, making it easier to manage large-scale, complex applications.

For example, if you have a multi-tenant cluster that needs to host multiple applications for different teams, you can use namespaces to isolate the resources for each team and ensure that each team only has access to the resources they need.

This way, different teams can work independently without interfering with each other's resources, and administrators can manage each team's resources separately.

☑ Advantages & Disadvantages

◇ ReplicationController

Advantages:

- Ensures high availability by maintaining pod replicas.
- Simpler than newer controllers.

Disadvantages:

- No rolling updates.
- No rollback support.
- Deprecated in favor of ReplicaSet/Deployment.

◇ ReplicaSet

Advantages:

- Supports set-based selectors (more flexible).
- Maintains pod count reliably.

Disadvantages:

- No rolling updates or rollback by itself.
- Typically used **internally by Deployments**.

◇ Deployment

Advantages:

- Supports updates, rollback, pause/resume.
- Declarative and production-ready.
- Creates and manages ReplicaSets for versioning.

Disadvantages:

- Slightly more complex YAML.
- May feel heavy for very simple cases.