Update Application Image and Settings

Objectives

- Update applications with minimal downtime by using deployment strategies.

Application Code, Configuration, and Data

Modern applications loosely couple code, configuration, and data. Configuration files and data are not hardcoded as part of the software. Instead, the software loads the configuration and data from an external source. Externalizing configuration and data enables deploying an application to different environments without requiring a change to the application source code.

OpenShift provides configuration map, secret, and volume resources to store the application configuration and data. The application code is available through container images.

Because OpenShift deploys applications from container images, developers must build a new version of the image when they update the code of their application. Organizations usually use a continuous integration and continuous delivery (CI/CD) pipeline to automatically build the image from the application source code, and then push the resulting image to a container registry.

You use OpenShift resources, such as configuration maps and secrets, to update the application configuration. To control the deployment process of a new image version, use a `Deployment` object.

Deployment Strategies

Deploying functional application changes or new versions to users is a significant phase of the CI/CD pipelines, where you add value to the development process.

Introducing application changes carries risks, such as downtime during the deployment, bugs, or reduced application performance. You can reduce or mitigate some risks with testing and validation stages in the pipelines.

Application or service downtime can result in lost business, disruption to other services that depend on yours, and violations of the service-level agreements, among others. To reduce downtime and minimize risks in deployments, use a deployment strategy. A deployment strategy changes or upgrades an application to minimize the impact of those changes.

In OpenShift, you use `Deployment` objects to define deployments and deployment strategies. The `RollingUpdate` and `Recreate` strategies are the main OpenShift deployment strategies.

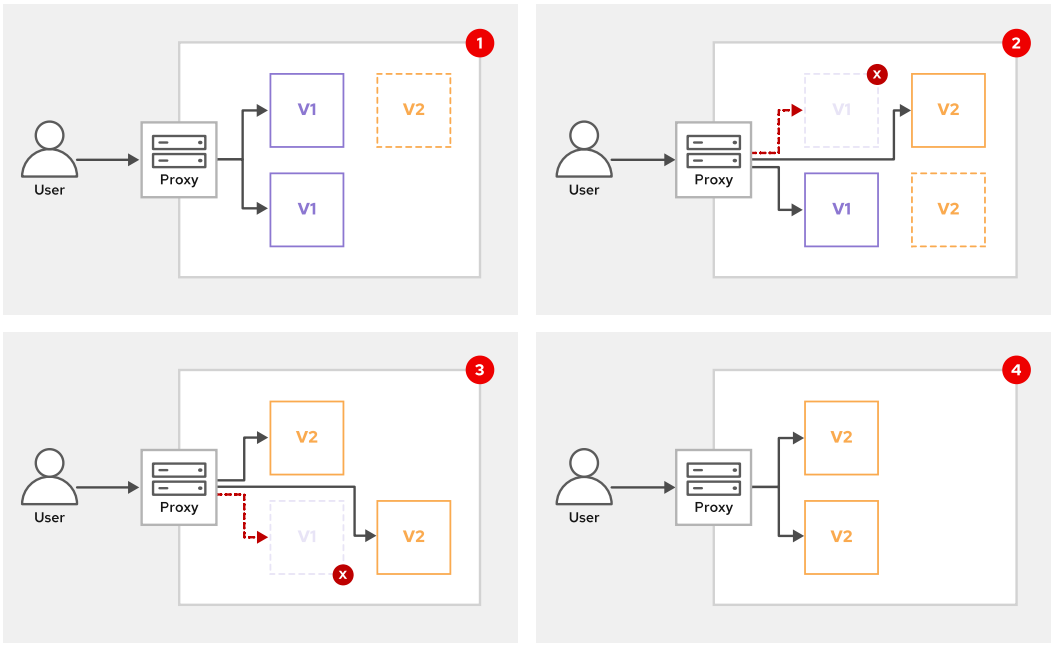
To select the `RollingUpdate` or `Recreate` strategies, you set the `.spec.strategy.type` property of the `Deployment` object. The following snippet shows a `Deployment` object that uses the `Recreate` strategy:

```
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: myapp2
  strategy:
    type: Recreate
  template:
...output omitted...
```

Rolling Update Strategy

The `RollingUpdate` strategy replaces one instance after another until all instances are replaced. In this strategy, both versions of the application run simultaneously, and the `RollingUpdate` strategy scales down instances of the previous version only when the new version is ready. The main drawback is that this strategy requires compatibility between the versions in the deployment.

The following graphic shows the deployment of a new version of an application by using the `RollingUpdate` strategy:



1. Some application instances run a code version that needs updating (v1). OpenShift scales up a new instance with the updated application version (v2). Because the new instance with version v2 is not ready, only the version v1 instances fulfill customer requests.

2. The instance with v2 is ready and accepts customer requests. OpenShift scales down an instance with version v1, and scales up a new instance with version v2. Both versions of the application fulfill customer requests.

3. The new instance with v2 is ready and accepts customer requests. OpenShift scales down the remaining instance with version v1.

4. No instances remain to replace. The application update was successful and without downtime.

The `RollingUpdate` strategy supports continuous deployment, and eliminates application downtime during deployments. You can use this strategy if the different versions of your application can run at the same time.

> **NOTE**
>
> The `RollingUpdate` strategy is the default strategy if you do not specify a strategy on the `Deployment` objects.

The following snippet shows a `Deployment` object that uses the `RollingUpdate` strategy:

```
apiVersion: apps/v1
kind: Deployment
metadata:
...output omitted...
spec:
  progressDeadlineSeconds: 600
  replicas: 10
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: myapp2
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 50%
    type: RollingUpdate
  template:
...output omitted...
```

Out of the many parameters to configure the `RollingUpdate` strategy, the preceding snippet shows the `maxSurge` and `maxUnavailable` parameters.

During a rolling update, the number of pods for the application varies, because OpenShift starts new pods for the new revision and removes pods from the previous revision.
The `maxSurge` parameter indicates how many pods OpenShift can create above the normal number of replicas. The `maxUnavailable` parameter indicates how many pods OpenShift can remove below the normal number of replicas. You can express these parameters as percentages or as the number of pods.

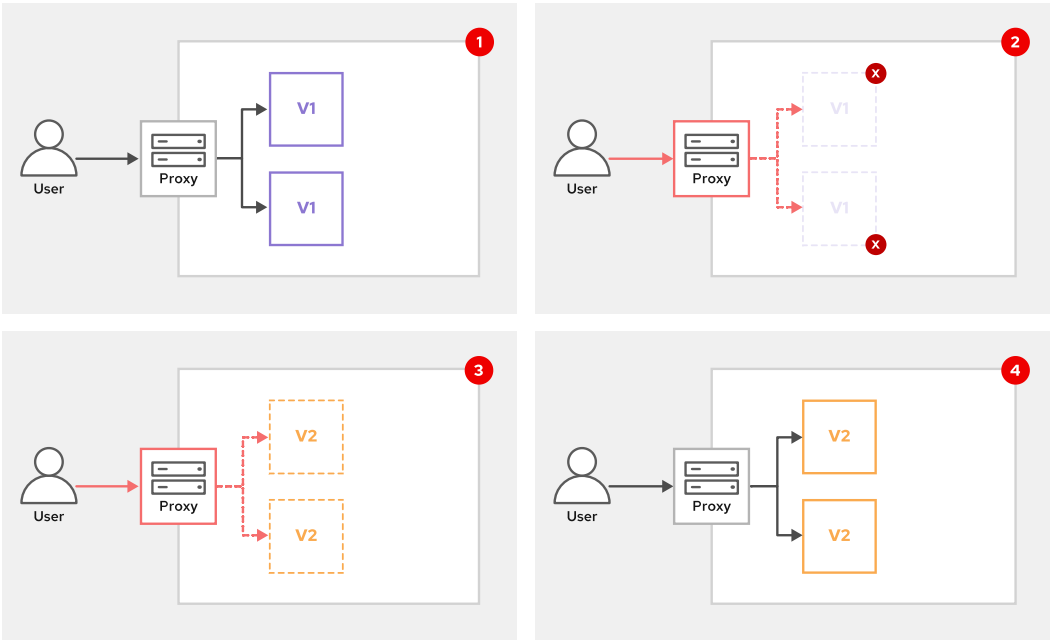
If you do not configure a readiness probe for your deployment, then during a rolling update, OpenShift starts sending client traffic to new pods as soon as they are running. However, the application inside a container might not be immediately ready to accept client requests, which might need to load files to cache, to establish a network connection to a database, or to perform initial tasks that might take time to complete. Consequently, OpenShift redirects client requests to a container that is not yet ready, and these requests fail.

Adding a readiness probe to the deployment prevents OpenShift from sending traffic to new pods that are not ready.

Recreate Strategy

In this strategy, all the instances of an application are stopped first, and are then replaced with new ones. The major drawback of this strategy is that it causes downtime in your services. For a period, no application instances are available to fulfill requests.

The following graphic shows the deployment of a new version of an application that uses the `Recreate` strategy:



1. The application has some instances that run a code version to update (v1).
2. OpenShift scales down the running instances to zero. Scaling down to zero causes application downtime, because no instances are available to fulfill requests.
3. OpenShift scales up new instances with a new version of the application (v2). When the new instances are booting, the downtime continues.
4. Starting the new instances resolves the application outage and completes the `Recreate` strategy.

The `Recreate` strategy is not recommended for applications that need high availability, for example, medical systems. You can use this strategy when your application cannot have different simultaneously running code versions. You might also use this strategy to execute data migrations or data transformations before the new code starts.

Rolling out Applications

When you update a `Deployment` object, OpenShift automatically rolls out the application. If you apply several modifications in a row, such as modifying the image version, updating the environment variables, and configuring the readiness probe, then OpenShift rolls out the application for each modification.

To prevent these multiple deployments, pause the rollout, apply all your modifications to the `Deployment` object, and then resume the rollout. OpenShift then performs a single rollout to apply all your modifications to the `Deployment` object.

- Use the `oc rollout pause` command for the `myapp` deployment:

```
[user@host ~]$ oc rollout pause deployment/myapp
```

- The following commands modify the image, an environment variable, and the readiness probe:

```
[user@host ~]$ oc set image deployment/myapp \
nginx-120=registry.access.redhat.com/ubi9/nginx-120:1-86
[user@host ~]$ oc set env deployment/myapp NGINX_LOG_TO_VOLUME=1
[user@host ~]$ oc set probe deployment/myapp --readiness --get-url http://:8080
```

- Resume the rollout of the `myapp` deployment:

```
[user@host ~]$ oc rollout resume deployment/myapp
```

  OpenShift rolls out the application to apply all your modifications to the `Deployment` object.

You can follow a similar process when you create and configure a new deployment:

- Create the deployment, and set the number of replicas to zero. By setting the replicas to zero, OpenShift does not roll out your application, and no pods are running:

```
[user@host ~]$ oc create deployment myapp2 \
--image registry.access.redhat.com/ubi9/nginx-120:1-86 --replicas 0
```

- Confirm that the deployment has no running pods:

```
[user@host ~]$ oc get deployment/myapp2
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp2   0/0     0            0           9s
```

- Apply the configuration to the `Deployment` object. The following command adds a readiness probe:

```
[user@host ~]$ oc set probe deployment/myapp2 --readiness --get-url http://:8080
```

- Scale up the deployment and OpenShift rolls out the application:

```
[user@host ~]$ oc scale deployment/myapp2 --replicas 10
```

```
[user@host ~]$ oc get deployment/myapp2
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
myapp2   10/10   10           10          18s
```

Monitoring Replica Sets

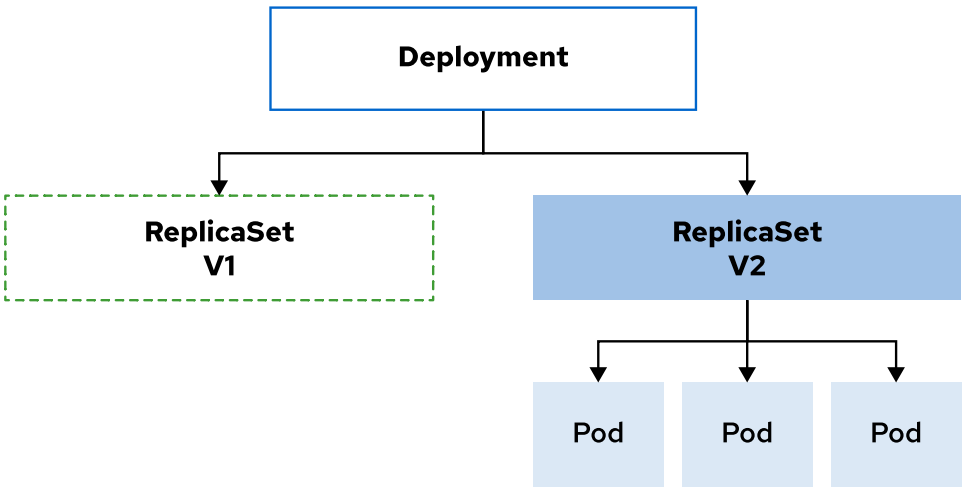
When OpenShift rolls out an application from the `Deployment` object, it creates a `ReplicaSet` object. Replica sets are responsible for creating and monitoring the pods. If a pod fails, then the `ReplicaSet` object deploys a new one.

To deploy pods, replica sets use the pod template definition from the `Deployment` object. OpenShift copies the template definition from the `Deployment` object when it creates the `ReplicaSet` object.

When you update the `Deployment` object, OpenShift does not update the existing `ReplicaSet` object. Instead, it creates another `ReplicaSet` object with the new pod template definition. Then, OpenShift rolls out the application according to the update strategy.

Therefore, several `ReplicaSet` objects for a deployment can exist at the same time on your system. During a rolling update, the earlier and the new `ReplicaSet` objects coexist and coordinate the rollout of the new application version. After the rollout completes, OpenShift keeps the earlier `ReplicaSet` object so that you can roll back if the new application version does not operate correctly.

The following graphic shows a `Deployment` object and two `ReplicaSet` objects. The earlier `ReplicaSet` object for version 1 of the application does not run any pods. The current `ReplicaSet` object for version 2 of the application manages three replicated pods.



Do not directly change or delete `ReplicaSet` objects, because OpenShift manages them through the associated `Deployment` objects. The `.spec.revisionHistoryLimit` attribute in `Deployment` objects specifies how many `ReplicaSet` objects OpenShift keeps. OpenShift automatically deletes the extra `ReplicaSet` objects. Also, when you delete a `Deployment` object, OpenShift deletes all the associated `ReplicaSet` objects.

Run the `oc get replicaset` command to list the `ReplicaSet` objects. OpenShift uses the `Deployment` object name as a prefix for the `ReplicaSet` objects.

```
[user@host ~]$ oc get replicaset
NAME                DESIRED   CURRENT   READY   AGE
myapp2-574968dd59   0         0         0       3m27s
myapp2-76679885b9   10        10        10      22s
myapp2-786cbf9bc8   0         0         0       114s
```

The preceding output shows three `ReplicaSet` objects for the `myapp2` deployment. Whenever you modified the `myapp2` deployment, OpenShift created a `ReplicaSet` object. The second object in the list is active and monitors 10 pods. The other `ReplicaSet` objects do not manage any pods.

During a rolling update, two `ReplicaSet` objects are active. The earlier `ReplicaSet` object is scaling down, and at the same time the new `ReplicaSet` object is scaling up:

```
[user@host ~]$ oc get replicaset
NAME                DESIRED   CURRENT   READY   AGE
myapp2-574968dd59   0         0         0       13m
myapp2-5fb5766df5   4         4         2       21s   (1)
myapp2-76679885b9   8         8         8       10m   (2)
myapp2-786cbf9bc8   0         0         0       11m
```

1. The new `ReplicaSet` object already started four pods, but the `READY` column shows that the readiness probe succeeded for only two pods so far. These two pods are likely to receive client traffic.

2. The `ReplicaSet` object already scaled down from 10 to 8 pods.

Managing Rollout

Because OpenShift preserves `ReplicaSet` objects from earlier deployment versions, you can roll back if you notice that the new version of the application does not work.

Use the `oc rollout undo` command to roll back to the preceding deployment version. The command uses the existing `ReplicaSet` object for that version to roll back the pods. The command also reverts the `Deployment` object to the preceding version.

```
[user@host ~]$ oc rollout undo deployment/myapp2
```

Use the `oc rollout status` command to control the rollout process:

```
[user@host ~]$ oc rollout status deployment/myapp2
deployment "myapp2" successfully rolled out
```

If the rollout operation fails, because you specify a wrong container image name or the readiness probe fails, then OpenShift does not automatically roll back your deployment. In this case, run the `oc rollout undo` command to revert to the preceding working configuration.

By default, the `oc rollout undo` command rolls back to the preceding deployment version. If you need to roll back to an earlier revision, then list the available revisions and add the `--to-revision` option to the `oc rollout undo` command.

- Use the `oc rollout history` command to list the available revisions:

```
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1         <none>
3         <none>
4         <none>
5         <none>
```

> **NOTE**
>
> The `CHANGE-CAUSE` column provides a user-defined message that describes the revision. You can store the message in the `kubernetes.io/change-cause` deployment annotation after every rollout:

```
[user@host ~]$ oc annotate deployment/myapp2 \
kubernetes.io/change-cause="Image updated to 1-86"
deployment.apps/myapp2 annotated
[user@host ~]$ oc rollout history deployment/myapp2
deployment.apps/myapp2
REVISION  CHANGE-CAUSE
1         <none>
3         <none>
4         <none>
5         Image updated to 1-86
```

- Add the `--revision` option to the `oc rollout history` command for more details about a specific revision:

```
[user@host ~]$ oc rollout history deployment/myapp2 --revision 1
deployment.apps/myapp2 with revision #1
Pod Template:
  Labels: app=myapp2
          pod-template-hash=574968dd59
  Containers:
   nginx-120:
    Image:       registry.access.redhat.com/ubi9/nginx-120:1-86
    Port:        <none>
    Host Port:   <none>
    Environment: <none>
    Mounts:      <none>
  Volumes: <none>
```

  The `pod-template-hash` attribute is the suffix of the associated `ReplicaSet` object. For example, you can inspect the `ReplicaSet` object for more details by using the `oc describe replicaset myapp2-574968dd59` command.

- Roll back to a specific revision by adding the `--to-revision` option to the `oc rollout undo` command:

```
[user@host ~]$ oc rollout undo deployment/myapp2 --to-revision 1
```

If you use floating tags to refer to container image versions in deployments, then the resulting image when you roll back a deployment might have changed in the container registry. As a result, the image that you run after the rollback might not be the original one that you used.

To prevent this issue, use the OpenShift image streams for referencing images instead of floating tags. A later section in this course discusses the OpenShift image streams in more detail.

REFERENCES

For more information about deployment strategies, refer to the *Using Deployment Strategies* section in the *Deployments* chapter in the Red Hat OpenShift Container Platform 4.18 *Building Applications* documentation

at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html/building_applications/deployments#deployment-strategies

For more information about readiness probes, refer to the *Monitoring Application Health by Using Health Checks* chapter in the Red Hat OpenShift Container Platform 4.18 *Building Applications* documentation at https://docs.redhat.com/en/documentation/openshift_container_platform/4.18/html-single/building_applications/index#application-health