- bgrant0607
- janetkuo title: Deployments

{:toc}

What is a Deployment?

A *Deployment* provides declarative updates for Pods and Replica Sets (the next-generation Replication Controller). You only need to describe the desired state in a Deployment object, and the Deployment controller will change the actual state to the desired state at a controlled rate for you. You can define Deployments to create new resources, or replace existing ones by new ones.

A typical use case is:

- Create a Deployment to bring up a Replica Set and Pods.
- Check the status of a Deployment to see if it succeeds or not.
- Later, update that Deployment to recreate the Pods (for example, to use a new image).
- Rollback to an earlier Deployment revision if the current Deployment isn't stable.
- Pause and resume a Deployment.

Creating a Deployment

Here is an example Deployment. It creates a Replica Set to bring up 3 nginx Pods.

{% include code.html language="yaml" file="nginx-deployment.yaml" ghlink="/docs/concepts/workloads/controllers/nginx-deployment.yaml" %}

Run the example by downloading the example file and then running this command:

```
$ kubectl create -f docs/user-guide/nginx-deployment.yaml --record
deployment "nginx-deployment" created
```

Setting the kubectl flag `--record` to `true` allows you to record current command in the annotations of the resources being created or updated. It will be useful for future introspection; for example, to see the commands executed in each Deployment revision.

Then running `get` immediately will give:

```
$ kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         0         0            0           1s
```

This indicates that the Deployment's number of desired replicas is 3 (according to deployment's `.spec.replicas`), the number of current replicas (`.status.replicas`) is 0, the number of up-to-date replicas (`.status.updatedReplicas`) is 0, and the number of available replicas (`.status.availableReplicas`) is also 0.

Running the `get` again a few seconds later, should give:

```
$ kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           18s
```

This indicates that the Deployment has created all three replicas, and all replicas are up-to-date (contains the latest pod template) and available (pod status is ready for at least Deployment's `.spec.minReadySeconds`). Running `kubectl get rs` and `kubectl get pods` will show the Replica Set (RS) and Pods created.

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-2035384211   3         3         0       18s
```

You may notice that the name of the Replica Set is always `<the name of the Deployment>-<hash value of the pod template>`.

```
$ kubectl get pods --show-labels
NAME                                READY     STATUS    RESTARTS   AGE       LABELS
nginx-deployment-2035384211-7ci7o   1/1       Running   0          18s       app=nginx,pod-template-hash=20
nginx-deployment-2035384211-kzszj   1/1       Running   0          18s       app=nginx,pod-template-hash=20
nginx-deployment-2035384211-qqcnn   1/1       Running   0          18s       app=nginx,pod-template-hash=20
```

The created Replica Set will ensure that there are three nginx Pods at all times.

Note: You must specify appropriate selector and pod template labels of a Deployment (in this case, `app = nginx`), i.e. don't overlap with other controllers (including Deployments, Replica Sets, Replication Controllers, etc.) Kubernetes won't stop you from doing that, and if you end up with multiple controllers that have overlapping selectors, those controllers will fight with each other's and won't behave correctly.

Updating a Deployment

Note: a Deployment's rollout is triggered if and only if the Deployment's pod template (i.e. `.spec.template`) is changed, e.g. updating labels or container images of the template. Other updates, such as scaling the Deployment, will not trigger a rollout.

Suppose that we now want to update the nginx Pods to start using the `nginx:1.9.1` image instead of the `nginx:1.7.9` image.

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
deployment "nginx-deployment" image updated
```

Alternatively, we can `edit` the Deployment and change `.spec.template.spec.containers[0].image` from `nginx:1.7.9` to `nginx:1.9.1`:

```
$ kubectl edit deployment/nginx-deployment
deployment "nginx-deployment" edited
```

To see its rollout status, simply run:

```
$ kubectl rollout status deployment/nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
deployment "nginx-deployment" successfully rolled out
```

After the rollout succeeds, you may want to `get` the Deployment:

```
$ kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           36s
```

The number of up-to-date replicas indicates that the Deployment has updated the replicas to the latest configuration. The current replicas indicates the total replicas this Deployment manages, and the available replicas indicates the number of current replicas that are available.

We can run `kubectl get rs` to see that the Deployment updated the Pods by creating a new Replica Set and scaling it up to 3 replicas, as well as scaling down the old Replica Set to 0 replicas.

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-1564180365   3         3         0       6s
nginx-deployment-2035384211   0         0         0       36s
```

Running `get pods` should now show only the new Pods:

```
$ kubectl get pods
NAME                               READY     STATUS    RESTARTS   AGE
nginx-deployment-1564180365-khku8   1/1       Running   0          4s
nginx-deployment-1564180365-nacti   1/1       Running   0          4s
nginx-deployment-1564180365-z9gth   1/1       Running   0          4s
```

Next time we want to update these Pods, we only need to update the Deployment's pod template again.

Deployment can ensure that only a certain number of Pods may be down while they are being updated. By default, it ensures that at least 25% less than the desired number of Pods are up (25% max unavailable).

Deployment can also ensure that only a certain number of Pods may be created above the desired number of Pods. By default, it ensures that at most 25% more than the desired number of Pods are up (25% max surge).

For example, if you look at the above Deployment closely, you will see that it first created a new Pod, then deleted some old Pods and created new ones. It does not kill old Pods until a sufficient number of new Pods have come up, and does not create new Pods until a sufficient number of old Pods have been killed. It makes sure that number of available Pods is at least 2 and the number of total Pods is at most 4.

```
$ kubectl describe deployments
Name:           nginx-deployment
Namespace:      default
CreationTimestamp:  Tue, 15 Mar 2016 12:01:06 -0700
Labels:         app=nginx
Selector:       app=nginx
Replicas:       3 updated | 3 total | 3 available | 0 unavailable
StrategyType:       RollingUpdate
MinReadySeconds:    0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
OldReplicaSets:     <none>
NewReplicaSet:      nginx-deployment-1564180365 (3/3 replicas created)
Events:
  FirstSeen LastSeen    Count   From                    SubobjectPath   Type        Reason              Me
  --------- --------    -----   ----                    -------------   --------    ------              --
  36s       36s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
  23s       23s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sc
```

Here we see that when we first created the Deployment, it created a Replica Set (nginx-deployment-2035384211) and scaled it up to 3 replicas directly. When we updated the Deployment, it created a new Replica Set (nginx-deployment-1564180365) and scaled it up to 1 and then scaled down the old Replica Set to 2, so that at least 2 Pods were available and at most 4 Pods were created at all times. It then continued scaling up and down the new and the old Replica Set, with the same rolling update strategy. Finally, we'll have 3 available replicas in the new Replica Set, and the old Replica Set is scaled down to 0.

Multiple Updates

Each time a new deployment object is observed by the deployment controller, a Replica Set is created to bring up the desired Pods if there is no existing Replica Set doing so. Existing Replica Set controlling Pods whose labels match `.spec.selector` but whose template does not match `.spec.template` are scaled down. Eventually, the new Replica Set will be scaled to `.spec.replicas` and all old Replica Sets will be scaled to 0.

If you update a Deployment while an existing deployment is in progress, the Deployment will create a new Replica Set as per the update and start scaling that up, and will roll the Replica Set that it was scaling up previously -- it will add it to its list of old Replica Sets and will start scaling it down.

For example, suppose you create a Deployment to create 5 replicas of `nginx:1.7.9`, but then updates the Deployment to create 5 replicas of `nginx:1.9.1`, when only 3 replicas of `nginx:1.7.9` had been created. In that case, Deployment will immediately start killing the 3 `nginx:1.7.9` Pods that it had created, and will start creating `nginx:1.9.1` Pods. It will not wait for 5 replicas of `nginx:1.7.9` to be created before changing course.

Rolling Back a Deployment

Sometimes you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, two previous Deployment's rollout history are kept in the system so that you can rollback anytime you want (you can change that by modifying revision history limit).

Note: a Deployment's revision is created when a Deployment's rollout is triggered. This means that the new revision is created if and only if the Deployment's pod template (i.e. `.spec.template`) is changed, e.g. updating labels or container images of the template. Other updates, such as scaling the Deployment, will not create a Deployment revision -- so that we can facilitate simultaneous manual- or auto-scaling. This implies that when you rollback to an earlier revision, only the Deployment's pod template part will be rolled back.

Suppose that we made a typo while updating the Deployment, by putting the image name as `nginx:1.91` instead of `nginx:1.9.1`:

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.91
deployment "nginx-deployment" image updated
```

The rollout will be stuck.

```
$ kubectl rollout status deployments nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

Press Ctrl-C to stop the above rollout status watch. For more information on stuck rollouts, read more here.

You will also see that both the number of old replicas (nginx-deployment-1564180365 and nginx-deployment-2035384211) and new replicas (nginx-deployment-3066724191) are 2.

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-1564180365   2         2         0       25s
nginx-deployment-2035384211   0         0         0       36s
nginx-deployment-3066724191   2         2         2       6s
```

Looking at the Pods created, you will see that the 2 Pods created by new Replica Set are stuck in an image pull loop.

```
$ kubectl get pods
NAME                                READY     STATUS             RESTARTS   AGE
nginx-deployment-1564180365-0iae    1/1       Running            0          5s
nginx-deployment-1564180365-jbqqo    1/1       Running            0          5s
nginx-deployment-3066724191-8mng    0/1       ImagePullBackOff   0          s
nginx-deployment-3066724191-eocby    0/1       ImagePullBackOff   0          s
```

Note that the Deployment controller will stop the bad rollout automatically, and will stop scaling up the new Replica Set.

```
$ kubectl describe deployment
Name:           nginx-deployment
Namespace:      default
CreationTimestamp:  Tue, 15 Mar 2016 14:48:04 -0700
Labels:         app=nginx
Selector:       app=nginx
Replicas:       2 updated | 3 total | 2 available | 2 unavailable
StrategyType:       RollingUpdate
MinReadySeconds:    0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
OldReplicaSets:     nginx-deployment-1564180365 (2/2 replicas created)
NewReplicaSet:      nginx-deployment-3066724191 (2/2 replicas created)
Events:
  FirstSeen LastSeen    Count   From                    SubobjectPath   Type        Reason              Mes
  --------- --------    -----   ----                    -------------   --------    ------              ---
  1m        1m          1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  22s       22s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  22s       22s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  22s       22s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  21s       21s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  13s       13s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  13s       13s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  13s       13s         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
```

To fix this, we need to rollback to a previous revision of Deployment that is stable.

Checking Rollout History of a Deployment

First, check the revisions of this deployment:

```
$ kubectl rollout history deployment/nginx-deployment
deployments "nginx-deployment":
REVISION    CHANGE-CAUSE
1           kubectl create -f docs/user-guide/nginx-deployment.yaml --record
2           kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
3           kubectl set image deployment/nginx-deployment nginx=nginx:1.91
```

Because we recorded the command while creating this Deployment using `--record`, we can easily see the changes we made in each revision.

To further see the details of each revision, run:

```
$ kubectl rollout history deployment/nginx-deployment --revision=2
deployments "nginx-deployment" revision 2
  Labels:       app=nginx
          pod-template-hash=1159050644
  Annotations:  kubernetes.io/change-cause=kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1
  Containers:
   nginx:
    Image:      nginx:1.9.1
    Port:       80/TCP
     QoS Tier:
        cpu:      BestEffort
        memory:   BestEffort
    Environment Variables:      <none>
  No volumes.
```

Rolling Back to a Previous Revision

Now we've decided to undo the current rollout and rollback to the previous revision:

```
$ kubectl rollout undo deployment/nginx-deployment
deployment "nginx-deployment" rolled back
```

Alternatively, you can rollback to a specific revision by specify that in `--to-revision`:

```
$ kubectl rollout undo deployment/nginx-deployment --to-revision=2
deployment "nginx-deployment" rolled back
```

For more details about rollout related commands, read `kubectl rollout`.

The Deployment is now rolled back to a previous stable revision. As you can see, a `DeploymentRollback` event for rolling back to revision 2 is generated from Deployment controller.

```
$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3         3         3            3           30m

$ kubectl describe deployment
Name:           nginx-deployment
Namespace:      default
CreationTimestamp:  Tue, 15 Mar 2016 14:48:04 -0700
Labels:         app=nginx
Selector:       app=nginx
Replicas:       3 updated | 3 total | 3 available | 0 unavailable
StrategyType:       RollingUpdate
MinReadySeconds:    0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
OldReplicaSets:     <none>
NewReplicaSet:      nginx-deployment-1564180365 (3/3 replicas created)
Events:
  FirstSeen LastSeen    Count   From                    SubobjectPath   Type        Reason              Mes
  --------- --------    -----   ----                    -------------   --------    ------              ---
  30m       30m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  29m       29m         1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  2m        2m          1       {deployment-controller }                Normal      ScalingReplicaSet   Sca
  2m        2m          1       {deployment-controller }                Normal      DeploymentRollback  Rol
  29m       2m          2       {deployment-controller }                Normal      ScalingReplicaSet   Sca
```

Clean up Policy

You can set `.spec.revisionHistoryLimit` field to specify how much revision history of this deployment you want to keep. By default, all revision history will be kept; explicitly setting this field to `0` disallows a deployment being rolled back.

Scaling a Deployment

You can scale a Deployment by using the following command:

```
$ kubectl scale deployment nginx-deployment --replicas 10
deployment "nginx-deployment" scaled
```

Assuming horizontal pod autoscaling is enabled in your cluster, you can setup an autoscaler for your Deployment and choose the minimum and maximum number of Pods you want to run based on the CPU utilization of your existing Pods.

```
$ kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80
deployment "nginx-deployment" autoscaled
```

RollingUpdate Deployments support running multiple versions of an application at the same time. When you or an autoscaler scales a RollingUpdate Deployment that is in the middle of a rollout (either in progress or paused), then the Deployment controller will balance the additional replicas in the existing active ReplicaSets (ReplicaSets with Pods) in order to mitigate risk. This is called *proportional scaling*.

For example, you are running a Deployment with 10 replicas, maxSurge=3, and maxUnavailable=2.

```
$ kubectl get deploy
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    10        10        10           10          50s
```

You update to a new image which happens to be unresolvable from inside the cluster.

```
$ kubectl set image deploy/nginx-deployment nginx=nginx:sometag
deployment "nginx-deployment" image updated
```

The image update starts a new rollout with ReplicaSet nginx-deployment-1989198191 but it's blocked due to the maxUnavailable requirement that we mentioned above.

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY     AGE
nginx-deployment-1989198191   5         5         0         9s
nginx-deployment-618515232    8         8         8         1m
```

Then a new scaling request for the Deployment comes along. The autoscaler increments the Deployment replicas to 15. The Deployment controller needs to decide where to add these new 5 replicas. If we weren't using proportional scaling, all 5 of them would be added in the new ReplicaSet. With proportional scaling, we spread the additional replicas across all ReplicaSets. Bigger proportions go to the ReplicaSets with the most replicas and lower proportions go to ReplicaSets with less replicas. Any leftovers are added to the ReplicaSet with the most replicas. ReplicaSets with zero replicas are not scaled up.

In our example above, 3 replicas will be added to the old ReplicaSet and 2 replicas will be added to the new ReplicaSet. The rollout process should eventually move all replicas to the new ReplicaSet, assuming the new replicas become healthy.

```
$ kubectl get deploy
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    15        18        7            8           7m
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY     AGE
nginx-deployment-1989198191   7         7         0         7m
nginx-deployment-618515232    11        11        11        7m
```

Pausing and Resuming a Deployment

You can also pause a Deployment mid-way and then resume it. A use case is to support canary deployment.

Update the Deployment again and then pause the Deployment with `kubectl rollout pause`:

```
$ kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1; kubectl rollout pause deployment/nginx-d
deployment "nginx-deployment" image updated
deployment "nginx-deployment" paused
```

Note that any current state of the Deployment will continue its function, but new updates to the Deployment will not have an effect as long as the Deployment is paused.

The Deployment was still in progress when we paused it, so the actions of scaling up and down Replica Sets are paused too.

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-1564180365   2         2         2       1h
nginx-deployment-2035384211   2         2         0       1h
nginx-deployment-3066724191   0         0         0       1h
```

In a separate terminal, watch for rollout status changes and you'll see the rollout won't continue:

```
$ kubectl rollout status deployment/nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
```

To resume the Deployment, simply do `kubectl rollout resume`:

```
$ kubectl rollout resume deployment/nginx-deployment
deployment "nginx-deployment" resumed
```

Then the Deployment will continue and finish the rollout:

```
$ kubectl rollout status deployment/nginx-deployment
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment spec update to be observed...
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
deployment nginx-deployment successfully rolled out
```

```
$ kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-1564180365   3         3         3       1h
nginx-deployment-2035384211   0         0         0       1h
nginx-deployment-3066724191   0         0         0       1h
```

Note: You cannot rollback a paused Deployment until you resume it.

Deployment status

A Deployment enters various states during its lifecycle. It can be progressing while rolling out a new ReplicaSet, it can be complete, or it can fail to progress.

Progressing Deployment

Kubernetes marks a Deployment as *progressing* when one of the following tasks is performed:

- The Deployment is in the process of creating a new ReplicaSet.
- The Deployment is scaling up an existing ReplicaSet.
- The Deployment is scaling down an existing ReplicaSet.
- New pods become available.

You can monitor the progress for a Deployment by using `kubectl rollout status`.

Complete Deployment

Kubernetes marks a Deployment as *complete* when it has the following characteristics:

- The Deployment has minimum availability. Minimum availability means that the Deployment's number of available replicas equals or exceeds the number required by the Deployment strategy.
- All of the replicas associated with the Deployment have been updated to the latest version you've specified, meaning any updates you've requested have been completed.
- No old pods for the Deployment are running.

You can check if a Deployment has completed by using `kubectl rollout status`. If the rollout completed successfully, `kubectl rollout status` returns a zero exit code.

```
$ kubectl rollout status deploy/nginx
Waiting for rollout to finish: 2 of 3 updated replicas are available...
deployment "nginx" successfully rolled out
$ echo $?
0
```

Failed Deployment

Your Deployment may get stuck trying to deploy its newest ReplicaSet without ever completing. This can occur due to some of the following factors:

- Insufficient quota
- Readiness probe failures
- Image pull errors
- Insufficient permissions
- Limit ranges
- Application runtime misconfiguration

One way you can detect this condition is to specify a deadline parameter in your Deployment spec: (`spec.progressDeadlineSeconds`). `spec.progressDeadlineSeconds` denotes the number of seconds the Deployment controller waits before indicating (via the Deployment status) that the Deployment progress has stalled.

The following `kubectl` command sets the spec with `progressDeadlineSeconds` to make the controller report lack of progress for a Deployment after 10 minutes:

```
$ kubectl patch deployment/nginx-deployment -p '{"spec":{"progressDeadlineSeconds":600}}'
"nginx-deployment" patched
```

Once the deadline has been exceeded, the Deployment controller adds a DeploymentCondition with the following attributes to the Deployment's `status.conditions`:

- Type=Progressing
- Status=False
- Reason=ProgressDeadlineExceeded

See the Kubernetes API conventions for more information on status conditions.

Note that in version 1.5, Kubernetes will take no action on a stalled Deployment other than to report a status condition with `Reason=ProgressDeadlineExceeded`.

Note: If you pause a Deployment, Kubernetes does not check progress against your specified deadline. You can safely pause a Deployment in the middle of a rollout and resume without triggering the condition for exceeding the deadline.

You may experience transient errors with your Deployments, either due to a low timeout that you have set or due to any other kind of error that can be treated as transient. For example, let's suppose you have insufficient quota. If you describe the Deployment you will notice the following section:

```
$ kubectl describe deployment nginx-deployment
<...>
Conditions:
  Type            Status  Reason
  ----            ------  ------
  Available       True    MinimumReplicasAvailable
  Progressing     True    ReplicaSetUpdated
  ReplicaFailure  True    FailedCreate
<...>
```

If you run `kubectl get deployment nginx-deployment -o yaml`, the Deployement status might look like this:

```
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: 2016-10-04T12:25:39Z
    lastUpdateTime: 2016-10-04T12:25:39Z
    message: Replica set "nginx-deployment-4262182780" is progressing.
    reason: ReplicaSetUpdated
    status: "True"
    type: Progressing
  - lastTransitionTime: 2016-10-04T12:25:42Z
    lastUpdateTime: 2016-10-04T12:25:42Z
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: 2016-10-04T12:25:39Z
    lastUpdateTime: 2016-10-04T12:25:39Z
    message: 'Error creating: pods "nginx-deployment-4262182780-" is forbidden: exceeded quota:
      object-counts, requested: pods=1, used: pods=3, limited: pods=2'
    reason: FailedCreate
    status: "True"
    type: ReplicaFailure
  observedGeneration: 3
  replicas: 2
  unavailableReplicas: 2
```

Eventually, once the Deployment progress deadline is exceeded, Kubernetes updates the status and the reason for the Progressing condition:

```
Conditions:
  Type            Status  Reason
  ----            ------  ------
  Available       True    MinimumReplicasAvailable
  Progressing     False   ProgressDeadlineExceeded
  ReplicaFailure  True    FailedCreate
```

You can address an issue of insufficient quota by scaling down your Deployment, by scaling down other controllers you may be running, or by increasing quota in your namespace. If you satisfy the quota conditions and the Deployment controller then completes the Deployment rollout, you'll see the Deployment's status update with a successful condition (`Status=True` and `Reason=NewReplicaSetAvailable`).

```
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
```

`Type=Available` with `Status=True` means that your Deployment has minimum availability. Minimum availability is dictated by the parameters specified in the deployment strategy. `Type=Progressing` with `Status=True` means that your Deployment is either in the middle of a rollout and it is progressing or that it has successfully completed its progress and the minimum required new replicas are available (see the Reason of the condition for the particulars - in our case `Reason=NewReplicaSetAvailable` means that the Deployment is complete).

You can check if a Deployment has failed to progress by using `kubectl rollout status`. `kubectl rollout status` returns a non-zero exit code if the Deployment has exceeded the progression deadline.

```
$ kubectl rollout status deploy/nginx
Waiting for rollout to finish: 2 out of 3 new replicas have been updated...
error: deployment "nginx" exceeded its progress deadline
$ echo $?
1
```

Operating on a failed deployment

All actions that apply to a complete Deployment also apply to a failed Deployment. You can scale it up/down, roll back to a previous revision, or even pause it if you need to apply multiple tweaks in the Deployment pod template.

Use Cases

Canary Deployment

If you want to roll out releases to a subset of users or servers using the Deployment, you can create multiple Deployments, one for each release, following the canary pattern described in managing resources.

Writing a Deployment Spec

As with all other Kubernetes configs, a Deployment needs `apiVersion`, `kind`, and `metadata` fields. For general information about working with config files, see deploying applications, configuring containers, and using kubectl to manage resources documents.

A Deployment also needs a `.spec` section.

Pod Template

The `.spec.template` is the only required field of the `.spec`.

The `.spec.template` is a pod template. It has exactly the same schema as a Pod, except it is nested and does not have an `apiVersion` or `kind`.

In addition to required fields for a Pod, a pod template in a Deployment must specify appropriate labels (i.e. don't overlap with other controllers, see selector) and an appropriate restart policy.

Only a `.spec.template.spec.restartPolicy` equal to `Always` is allowed, which is the default if not specified.

Replicas

`.spec.replicas` is an optional field that specifies the number of desired Pods. It defaults to 1.

Selector

`.spec.selector` is an optional field that specifies a label selector for the Pods targeted by this deployment.

If specified, `.spec.selector` must match `.spec.template.metadata.labels`, or it will be rejected by the API. If `.spec.selector` is unspecified, `.spec.selector.matchLabels` will be defaulted to `.spec.template.metadata.labels`.

Deployment may kill Pods whose labels match the selector, in the case that their template is different than `.spec.template` or if the total number of such Pods exceeds `.spec.replicas`. It will bring up new Pods with `.spec.template` if number of Pods are less than the desired number.

Note that you should not create other pods whose labels match this selector, either directly, via another Deployment or via another controller such as Replica Sets or Replication Controllers. Otherwise, the Deployment will think that those pods were created by it. Kubernetes will not stop you from doing this.

If you have multiple controllers that have overlapping selectors, the controllers will fight with each other's and won't behave correctly.

Strategy

`.spec.strategy` specifies the strategy used to replace old Pods by new ones. `.spec.strategy.type` can be "Recreate" or "RollingUpdate". "RollingUpdate" is the default value.

Recreate Deployment

All existing Pods are killed before new ones are created when `.spec.strategy.type==Recreate`.

Rolling Update Deployment

The Deployment updates Pods in a rolling update fashion when `.spec.strategy.type==RollingUpdate`. You can specify `maxUnavailable` and `maxSurge` to control the rolling update process.

Max Unavailable

`.spec.strategy.rollingUpdate.maxUnavailable` is an optional field that specifies the maximum number of Pods that can be unavailable during the update process. The value can be an absolute number (e.g. 5) or a percentage of desired Pods (e.g. 10%). The absolute number is calculated from percentage by rounding up. This can not be 0 if `.spec.strategy.rollingUpdate.maxSurge` is 0. By default, a fixed value of 1 is used.

For example, when this value is set to 30%, the old Replica Set can be scaled down to 70% of desired Pods immediately when the rolling update starts. Once new Pods are ready, old Replica Set can be scaled down further, followed by scaling up the new Replica Set, ensuring that the total number of Pods available at all times during the update is at least 70% of the desired Pods.

Max Surge

`.spec.strategy.rollingUpdate.maxSurge` is an optional field that specifies the maximum number of Pods that can be created above the desired number of Pods. Value can be an absolute number (e.g. 5) or a percentage of desired Pods (e.g. 10%). This can not be 0 if `MaxUnavailable` is 0. The absolute number is calculated from percentage by rounding up. By default, a value of 1 is used.

For example, when this value is set to 30%, the new Replica Set can be scaled up immediately when the rolling update starts, such that the total number of old and new Pods do not exceed 130% of desired Pods. Once old Pods have been killed, the new Replica Set can be scaled up further, ensuring that the total number of Pods running at any time during the update is at most 130% of desired Pods.

Progress Deadline Seconds

`.spec.progressDeadlineSeconds` is an optional field that specifies the number of seconds you want to wait for your Deployment to progress before the system reports back that the Deployment has failed progressing - surfaced as a condition with `Type=Progressing`, `Status=False`. and `Reason=ProgressDeadlineExceeded` in the status of the resource. The deployment controller will keep retrying the Deployment. In the future, once automatic rollback will be implemented, the deployment controller will roll back a Deployment as soon as it observes such a condition.

If specified, this field needs to be greater than `.spec.minReadySeconds`.

Min Ready Seconds

`.spec.minReadySeconds` is an optional field (with default value of 600s) that specifies the minimum number of seconds for which a newly created Pod should be ready without any of its containers crashing, for it to be considered available. This defaults to 0 (the Pod will be considered available as soon as it is ready). To learn more about when a Pod is considered ready, see Container Probes.

Rollback To

`.spec.rollbackTo` is an optional field with the configuration the Deployment is rolling back to. Setting this field will trigger a rollback, and this field will be cleared every time a rollback is done.

Revision

`.spec.rollbackTo.revision` is an optional field specifying the revision to rollback to. This defaults to 0, meaning rollback to the last revision in history.

Revision History Limit

A deployment's revision history is stored in the replica sets it controls.

`.spec.revisionHistoryLimit` is an optional field (with default value of two) that specifies the number of old Replica Sets to retain to allow rollback. Its ideal value depends on the frequency and stability of new deployments. All old Replica Sets will be kept by default, consuming resources in `etcd` and crowding the output of `kubectl get rs`, if this field is not set. The configuration of each Deployment revision is stored in its Replica Sets; therefore, once an old Replica Set is deleted, you lose the ability to rollback to that revision of Deployment.

More specifically, setting this field to zero means that all old replica sets with 0 replica will be cleaned up. In this case, a new deployment rollout cannot be undone, since its revision history is cleaned up.

Paused

`.spec.paused` is an optional boolean field for pausing and resuming a Deployment. It defaults to false (a Deployment is not paused).

Alternative to Deployments

kubectl rolling update

Kubectl rolling update updates Pods and Replication Controllers in a similar fashion. But Deployments are recommended, since they are declarative, server side, and have additional features, such as rolling back to any previous revision even after the rolling update is done.