Kubectl

Last updated 6/28/2023

Comments (1)

kubectl is the command line tool that is used to interact with Kubernetes clusters, and therefore with your Managed Containers namespace.

After a local installation, the command line tool can be used from your workstation to perform mutating commands on non-production clusters.

Installation

You can download the binaries with curl from the external website as documented. The remote repository is available internally as the proxy repository on Artifactory.

For example, the external URL

```
https://storage.googleapis.com/kubernetes-release/re
```

is available on Artifactory in the googleapis-storage repository at

> Newer versions are available from the internal repository, even if they are not visible. Just update the version and request from the repository on the command-line.

```
curl -LO https://generic.ci.artifacts.walmart.com/artifa
```

A successful installation allows you to verify the version of kubectl:

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"11", ... }
Server Version: version.Info{Major:"1", Minor:"9", ...  }
```

The server version output only works after you have configured kubectl.

Additionally, we recommend configuration of command completion for kubectl for bash or zsh users. This and other useful tips are documented on the kubectl cheatsheet.

Installing Optional WCNP Plugins

To use these plugins, you should have already installed kubectl as shown above.

> *These plugins require kubectl 1.12.0 or higher, and a BASH execution environment.*

Installing the plugins on a linux system is done using a single command:

```
curl "http://resources.k8s.walmart.com/get-kubectl-plu
```

Once installed, you may issue additional kubectl commands offered through the plugins, as described in the following sections.

Configuration

You can use sledge to configure kubectl access and connect it to your existing namespace.

```
sledge connect scus-lab-a1
```

Project-specific configuration can be stored in a `k8s` folder. It is then potentially available for command line usage.

CLI Examples

Extensive documentation for command line usage of kubectl is available on the kubectl website, including a handy cheat sheet.

As an application developer, there are a number of common issues for you to troubleshoot in your Kubernetes environment. Here are a number of samples that provide you with the ability to do a majority of the troubleshooting necessary on the Kubernetes side for most applications.

Using namespaces in kubectl commands

Your deployment should most likely be using a non-default namespace.

To interact with your namespaced deployment, kubectl commands require you to provide the namespace as part of the command.

For example, if your kitt.yml contains `namespace: strati-training`, then you would run kubectl commands with a `-n strati-training` command line argument.

Check the status of your deployment

Use `kubectl get deploy` to view the status of all deployments, or include the name to check the status of a single deployment.

```
$ kubectl get deploy
NAME                DESIRED   CURRENT   UP-TO-DATE
frontend-deployment   3         3         3           0          5r
```

Here there is a problem. Our deployment has the number of replicas that we want, but none of them are available for some reason. Let's take a closer look to see what might be occuring.

Check the detailed status of a deployment

Use kubectl describe deploy {deployment name} for a detailed view of a deployment and the overall status.

```
$ kubectl describe deploy frontend-deployment
Name:                   frontend-deployment
Namespace:              default
CreationTimestamp:      Tue, 09 Apr 2019 09:31:20 -0
Labels:                 app=frontend
Annotations:            deployment.kubernetes.io/revision
                        kubectl.kubernetes.io/last-applied-config
                          {"apiVersion":"apps/v1","kind":"Deploym
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 0 avail
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% ma
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.7.9
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      False   MinimumReplicasUnavailable
  Progressing    False   ProgressDeadlineExceeded
OldReplicaSets:  <none>
NewReplicaSet:   frontend-deployment-5c689d88bb
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  18m   deployment-contro
```

This shows the status, the pod template used for the ReplicaSet, and the latest events. Pay attention to the OldReplicaSet and NewReplicaSet items. These contain the hash or the ReplicaSet currently being managed by the deployment. Using that, we can now view the individual pods to see whether we can find what might be wrong. Also, take note of the labels in the pod template. These will make finding the pods for this deployment a bit more easily in a large environment.

Check the status of a single pod in a deployment

First, we need to find the pods that are a part of this deployment. We can do that with kubectl get pods and the label mentioned in the deployment description above using the -l flag.

```
$ kubectl get pods -l app=nginx
NAME                               READY   STATUS          RES
frontend-deployment-5c689d88bb-6t4g6   0/1     Im
frontend-deployment-5c689d88bb-fn9k7   0/1     Ima
frontend-deployment-5c689d88bb-vb84v   0/1     Im
```

This shows us the problem. Now let's go ahead and take a closer look at one of these pods to be certain we understand the issue.

We can view details of the pod, in the same way we did in the deployment by using the `describe` command and one of the pod names.

> The pod name is using that ReplicaSet hash mentioned in the deployment description above. This is useful if you have large deployments that might be stuck in a rolling out status. Always make sure the hash matches the deployment Old or New ReplicaSet of interest.

```
$ kubectl describe pod frontend-deployment-5c689d
Name:          frontend-deployment-5c689d88bb-6t
Namespace:       default
Priority:        0
PriorityClassName:  <none>
Node:           worke1547000001/10.74.248.40
Start Time:      Tue, 09 Apr 2019 09:31:21 -0500
Labels:          app=nginx
              pod-template-hash=5c689d88bb
Annotations:       <none>
Status:          Pending
IP:             10.24.21.231
Controlled By:     ReplicaSet/frontend-deployment-5c
Containers:
 nginx:
  Container ID:
  Image:         nginx:1.7.9
  Image ID:
  Port:          80/TCP
  Host Port:      0/TCP
  State:         Waiting
   Reason:       ImagePullBackOff
  Ready:         False
  Restart Count:  0
  Environment:   <none>
  Mounts:
   /var/run/secrets/kubernetes.io/serviceaccount fro
Conditions:
 Type           Status
 Initialized      True
 Ready           False
 ContainersReady  False
```

```
  Type:    Secret (a volume populated by a Secret)
  SecretName:  default-token-b6qsz
  Optional:   false
QoS Class:     BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecu
             node.kubernetes.io/unreachable:NoExecute
Events:
 Type    Reason    Age              From                Me
 ----    ------    ----             ----                -------
 Normal  Scheduled  30m              default-schedule
 Normal  Pulling    28m (x4 over 30m)    kubelet, wor
 Warning  Failed    28m (x4 over 30m)    kubelet, wor
 Warning  Failed    28m (x4 over 30m)    kubelet, wor
 Warning  Failed    5m39s (x99 over 30m)  kubelet, w
 Normal  BackOff    41s (x120 over 30m)  kubelet, w
```

This gives us much more information. Specifically the last few events tell us what problem is causing the image pull failure. We can't connect to the docker registry directly. Because these systems live inside of the Walmart network, we have to use the local artifactory repos for accessing outside resources. For more details, go to the KITT Docker documentation.

We can make the changes by editing the deployment and checking the status.

```
$ kubectl get deploy
NAME              DESIRED   CURRENT   UP-TO-DATE
frontend-deployment  3        3        3         3         38
```

Viewing the application logs inside of a pod

Now that our application is running in a deployment, we will need to view the actual application logs in the container. While these logs are most likely available in your Splunk or ELK instance, it can be valuable to look at the actual container logs, especially if you are having an issue with log collection.

To view the logs of a pod with only one container you can use kubectl logs {podname}:

```
$ kubectl logs frontend-deployment-5d7c89c76c-djg
80.91.33.133 - - [Day/Month/Year:08:05:50 +0000] '
173.203.139.108 - - [Day/Month/Year:08:05:03 +00
80.91.33.133 - - [Day/Month/Year:08:05:35 +0000] '
5.83.131.103 - - [Day/Month/Year:08:05:51 +0000] '
80.91.33.133 - - [Day/Month/Year:08:05:59 +0000] '
200.6.73.40 - - [Day/Month/Year:08:05:42 +0000] "G
80.91.33.133 - - [Day/Month/Year:08:05:48 +0000] '
93.180.71.3 - - [Day/Month/Year:08:05:58 +0000] "G
```

If you have a pod with multiple containers you will need to use the -c flag to specify which containers logs you want to view.

Getting a command line in your container

In some cases, it may be necessary to get a command line inside of your actual container to troubleshoot an issue you are having. Provided your base image is Alpine, or any other distribution with a full shell, you can connect to a container and execute a command.

Using kubectl exec -it {pod name} -- /bin/bash you can execute a bash shell on a particular pod. You can also execute commands directly when you know their location, or they are part of the containers internal path.

```
$ kubectl exec -it frontend-deployment-5d7c89c76c-
nginx version: nginx/1.11.4
built by gcc 4.9.2 (Debian 4.9.2-10)
built with OpenSSL 1.0.1t  3 May 2016
TLS SNI support enabled
configure arguments: --prefix=/etc/nginx --sbin-path=
```

Perform a manual scaling request

_This action results in a modification to cluster objects outside of the normal KITT release process.
Depending on the nature of other changes in the kitt file, the changes made here may persist or get overridden from the kitt file.

For example, a manual update of HorizontalPodAutoscaler minReplicas or maxReplicas using the kubectl will ignore the kitt.yml scaling.min or scaling.max values. To retain any scaling configuration, you must go back and update the corresponding kitt.yml in your GitHub repo._

Use of this requires installing WCNP kubectl plugins as described above.

While KITT-managed deployments are set up by default to use autoscaling, there may be times during larger-scale testing, or one-time predicted traffic spikes, where

manual scale up is desired.
This command performs a manual scale up or scale down of the provisioned HPA for the deployment.

Example Usage:

```
$ kubectl scale-hpa --context=scus-lab-a1 --hpa myap
scaling hpa/myapp...
horizontalpodautoscaler.autoscaling/myapp configure
hpa myapp successfully updated to min=2 and max=3.
$
```

More information about the command is available using the kubectl help option for the command.

Perform a rolling restart

> *Use of this requires installing WCNP kubectl plugins as [described above](#).*

Deleting a pod serves as a restart for a particular app instance. But there may be times when a restart of all running instances is desired. If the application has many instances, it can be cumbersome to issue individual pod delete commands for every instance.

This command performs a *rolling restart* of all active pods for a deployment using a single convenient command. Rolling restarts help avoid app outages by keeping one or more active pods up. The number of restarting pods is determined by the metadata settings of the deployment itself. With KITT this is defaulted to a maximum of `25%` of the available pods.

Example Usage:

```
$ kubectl rolling-restart --timeout 3m --context=scus-
starting rolling restart of deployment/myapp...
deployment.extensions/myapp env updated
Waiting for deployment "myapp" rollout to finish: 1 olc
Waiting for deployment "myapp" rollout to finish: 1 olc
deployment "myapp" successfully rolled out
Finished rolling restart of deployment myapp.
$
```

The `timeout` option, `-t` or `--timeout`, is used purely for polling purposes, after the restart command has been submitted to the cluster. If the polling stops before the cluster operation is complete, the restarts will continue on the cluster, up until `progressDeadlineSeconds` is exceeded. The value for `progressDeadlineSeconds` is set by KITT to a default of `600` on the kubernetes deployment at the time of app installation.

More information about the command is available using the kubectl help option for the command.

Was this helpful?

Comments(1)

YJ

Type your comment here

Post

LG **Lin Gong** May 18th 2022, 3:04 PM

Use homebrew to install is much easier:
brew install kubectl
(edited)

26