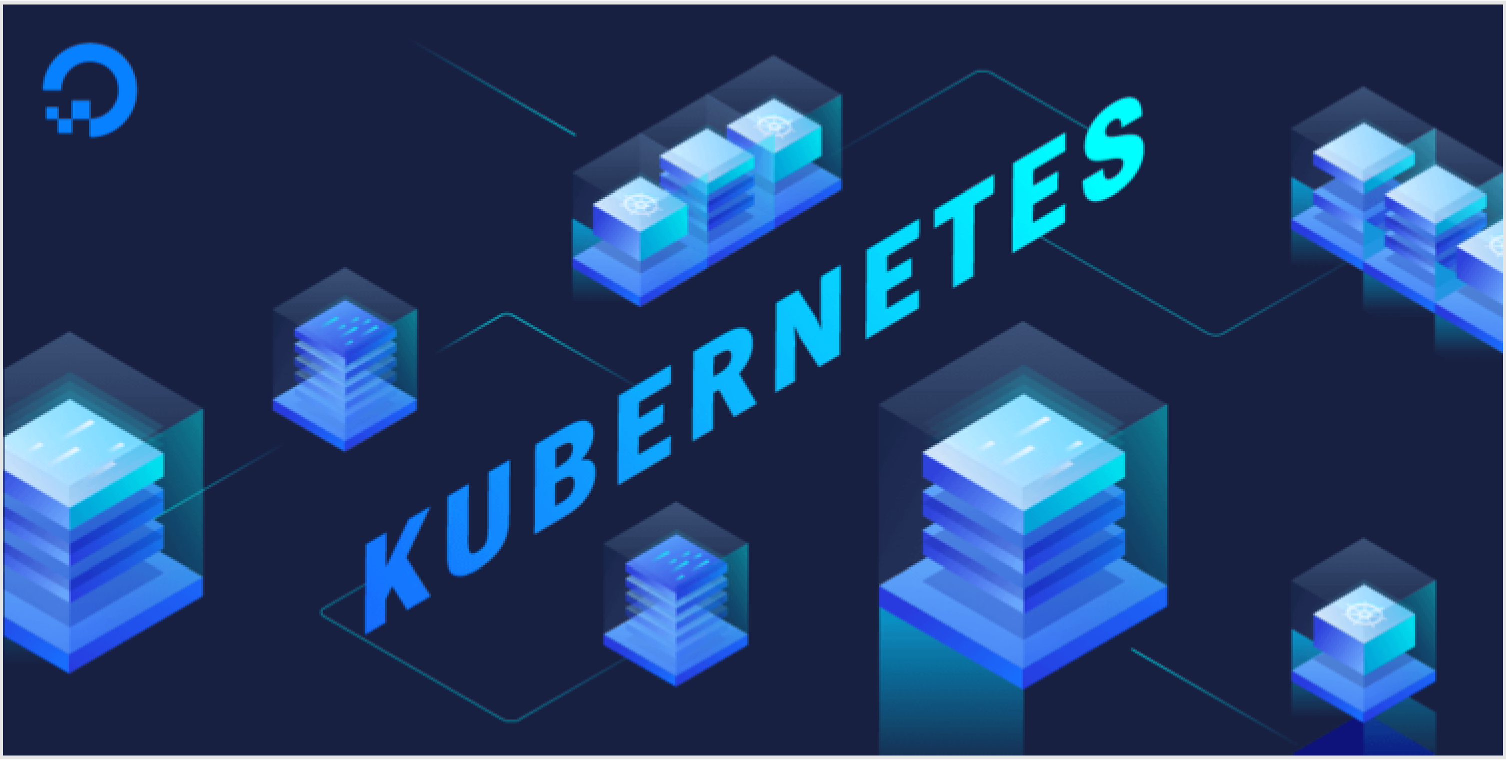

How To Install Software on Kubernetes Clusters with the Helm Package Manager

3

Posted August 15, 2018 19.3k KUBERNETES

By: Brian Boucheron

Introduction

Helm is a package manager for Kubernetes that allows developers and operators to more easily configure and deploy applications on Kubernetes clusters.

In this tutorial we will set up Helm and use it to install, reconfigure, rollback, then delete an instance of the Kubernetes Dashboard application. The dashboard is an official web-based Kubernetes GUI.

For a conceptual overview of Helm and its packaging ecosystem, please read our article *An Introduction to Helm*.

Prerequisites

For this tutorial you will need:

- A Kubernetes 1.8+ cluster with role-based access control (RBAC) enabled.
- The `kubectl` command-line tool installed on your local machine, configured to connect to your cluster. You can read more about installing `kubectl` in the official documentation.

  You can test your connectivity with the following command:

  ```
  $ kubectl cluster-info
  ```

  If you see no errors, you're connected to the cluster. If you access multiple clusters with `kubectl`, be sure to verify that you've selected the correct cluster context:

  ```
  $ kubectl config get-contexts
  ```

  ```
  Output
  CURRENT   NAME                   CLUSTER                      AUTHINFO                      NAI
  *         do-nyc1-k8s-example    do-nyc1-k8s-example          do-nyc1-k8s-example-admin
            docker-for-desktop     docker-for-desktop-cluster   docker-for-desktop
  ```

  In this example the asterisk (*) indicates that we are connected to the `do-nyc1-k8s-example` cluster. To switch clusters run:

  ```
  $ kubectl config use-context context-name
  ```

When you are connected to the correct cluster, continue to Step 1 to begin installing Helm.

Step 1 — Installing Helm

First we'll install the `helm` command-line utility on our local machine. Helm provides a script that handles the installation process on MacOS, Windows, or Linux.

Change to a writable directory and download the script from Helm's GitHub repository:

```
$ cd /tmp
$ curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get > install-helm.sh
```

Make the script executable with `chmod`:

```
$ chmod u+x install-helm.sh
```

At this point you can use your favorite text editor to open the script and inspect it to make sure it's safe. When you are satisfied, run it:

```
$ ./install-helm.sh
```

You may be prompted for your password. Provide it and press `ENTER`.

```
Output
helm installed into /usr/local/bin/helm
Run 'helm init' to configure helm.
```

Next we will finish the installation by installing some Helm components on our cluster.

Step 2 — Installing Tiller

Tiller is a companion to the `helm` command that runs on your cluster, receiving commands from `helm` and communicating directly with the Kubernetes API to do the actual work of creating and deleting resources. To give Tiller the permissions it needs to run on the cluster, we are going to make a Kubernetes `serviceaccount` resource.

Note: We will bind this `serviceaccount` to the **cluster-admin** cluster role. This will give the `tiller` service superuser access to the cluster and allow it to install all resource types in all namespaces. This is fine for exploring Helm, but you may want a more locked-down configuration for a production Kubernetes cluster.

Please refer to the official Helm RBAC documentation for more information on setting up different RBAC scenarios for Tiller.

Create the **tiller** `serviceaccount`:

```
$ kubectl -n kube-system create serviceaccount tiller
```

Next, bind the **tiller** `serviceaccount` to the **cluster-admin** role:

```
$ kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:
```

Now we can run `helm init`, which installs Tiller on our cluster, along with some local housekeeping tasks such as downloading the **stable** repo details:

```
$ helm init --service-account tiller
```

```
Output
. . .

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
For more information on securing your installation see: https://docs.helm.sh/using_helm/#securing-you
Happy Helming!
```

To verify that Tiller is running, list the pods in the**kube-system** namespace:

```
$ kubectl get pods --namespace kube-system
```

```
Output
NAME                                    READY     STATUS    RESTARTS   AGE
. . .
kube-dns-64f766c69c-rm9tz               3/3       Running   0          22m
kube-proxy-worker-5884                  1/1       Running   1          21m
kube-proxy-worker-5885                  1/1       Running   1          21m
kubernetes-dashboard-7dd4fc69c8-c4gwk   1/1       Running   0          22m
tiller-deploy-5c688d5f9b-lccsk          1/1       Running   0          40s
```

The Tiller pod name begins with the prefix `tiller-deploy-` .

Now that we've installed both Helm components, we're ready to use `helm` to install our first application.

Step 3 — Installing a Helm Chart

Helm software packages are called *charts*. Helm comes preconfigured with a curated chart repository called **stable**. You can browse the available charts in their GitHub repo. We are going to install the Kubernetes Dashboard as an example.

Use `helm` to install the `kubernetes-dashboard` package from the `stable` repo:

```
$ helm install stable/kubernetes-dashboard --name dashboard-demo
```

```
Output
NAME:   dashboard-demo
LAST DEPLOYED: Wed Aug  8 20:11:07 2018
NAMESPACE: default
STATUS: DEPLOYED
```

```
. . .
```

Notice the `NAME` line, highlighted in the above example output. In this case we specified the name `dashboard-demo`. This is the name of our *release*. A Helm *release* is a single deployment of one chart with a specific configuration. You can deploy multiple releases of the same chart with, each with its own configuration.

If you don't specify your own release name using `--name`, Helm will create a random name for you.

We can ask Helm for a list of releases on this cluster:

```
$ helm list
```

```
Output
NAME            REVISION    UPDATED                     STATUS      CHART                       NAMES
dashboard-demo    1           Wed Aug  8 20:11:11 2018    DEPLOYED    kubernetes-dashboard-0.7.1  def
```

We can now use `kubectl` to verify that a new service has been deployed on the cluster:

```
$ kubectl get services
```

```
Output
NAME                                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
dashboard-demo-kubernetes-dashboard   ClusterIP   10.32.104.73   <none>        443/TCP   51s
kubernetes                              ClusterIP   10.32.0.1      <none>        443/TCP   34m
```

Notice that by default the service name corresponding to our release is a combination of the Helm release name and the chart name.

Now that we've deployed the application, let's use Helm to change its configuration and update the deployment.

Step 4 — Updating a Release

The `helm upgrade` command can be used to upgrade a release with a new or updated chart, or update the it's configuration options.

We're going to make a simple change to our `dashboard-demo` release to demonstrate the update and rollback process: we'll update the name of the dashboard service to just `dashboard`, instead of `dashboard-demo-kubernetes-dashboard`.

The `kubernetes-dashboard` chart provides a `fullnameOverride` configuration option to control the service name. Let's run `helm upgrade` with this option set:

```
$ helm upgrade dashboard-demo stable/kubernetes-dashboard --set fullnameOverride="dashboard"
```

You'll see output similar to the initial `helm install` step.

Check if your Kubernetes services reflect the updated values:

```
$ kubectl get services
```

```
Output
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes              ClusterIP   10.32.0.1       <none>        443/TCP   36m
dashboard               ClusterIP   10.32.198.148   <none>        443/TCP   40s
```

Our service name has been updated to the new value.

Note: At this point you may want to actually load the Kubernetes Dashboard in your browser and check it out. To do so, first run the following command:

```
$ kubectl proxy
```

This creates a proxy that lets you access remote cluster resources from your local computer. Based on the previous instructions your dashboard service is named `kubernetes-dashboard` and it's running in the `default` namespace. You may now access the dashboard at the following url:

```
http://localhost:8001/api/v1/namespaces/default/services/https:dashboard:/proxy/
```

If necessary, substitute your own service name and namespace for the highlighted portions. Instructions for actually using the dashboard are out of scope for this tutorial, but you can read the official Kubernetes Dashboard docs for more information.

Next we'll look at Helm's ability to roll back releases.

Step 5 — Rolling Back a Release

When we updated our `dashboard-demo` release in the previous step, we created a second *revision* of the release. Helm retains all the details of previous releases in case you need to roll back to a prior configuration or chart.

Use `helm list` to inspect the release again:

```
$ helm list
```

```
Output
NAME            REVISION    UPDATED                     STATUS      CHART                       NAMES
dashboard-demo  2           Wed Aug  8 20:13:15 2018    DEPLOYED    kubernetes-dashboard-0.7.1  default
```

The `REVISION` column tells us that this is now the second revision.

Use `helm rollback` to roll back to the first revision:

```
$ helm rollback dashboard-demo 1
```

You should see the following output, indicating that the rollback succeeded:

```
Output
Rollback was a success! Happy Helming!
```

At this point, if you run `kubectl get services` again, you will notice that the service name has changed back to its previous value. Helm has re-deployed the application with revision 1's configuration.

Next we'll look into deleting releases with Helm.

Step 6 — Deleting a Release

Helm releases can be deleted with the `helm delete` command:

```
$ helm delete dashboard-demo
```

```
Output
release "dashboard-demo" deleted
```

Though the release has been deleted and the dashboard application is no longer running, Helm saves all the revision information in case you want to re-deploy the release. If you tried to `helm install` a new `dashboard-demo` release right now, you'd get an error:

```
Error: a release named dashboard-demo already exists.
```

If you use the `--deleted` flag to list your deleted releases, you'll see that the release is still around:

```
$ helm list --deleted
```

```
Output
NAME            REVISION    UPDATED                     STATUS  CHART                       NAMESPACE
dashboard-demo  3           Wed Aug  8 20:15:21 2018    DELETED kubernetes-dashboard-0.7.1  default
```

To *really* delete the release and purge all old revisions, use the `--purge` flag with the `helm delete` command:

```
$ helm delete dashboard-demo --purge
```

Now the release has been truly deleted, and you can reuse the release name.

Conclusion

In this tutorial we installed the `helm` command-line tool and its `tiller` companion service. We also explored installing, upgrading, rolling back, and deleting Helm charts and releases.

For more information about Helm and Helm charts, please see the official Helm documentation.

By: Brian Boucheron

♡ Upvote (3)   Subscribe   Share

Related Tutorials

An Introduction to Service Meshes

How To Deploy a PHP Application with Kubernetes on Ubuntu 16.04

How To Set Up Multi-Node Deployments With Rancher 2.1, Kubernetes, and Docker Machine on Ubu...

How to Set Up an Nginx Ingress with Cert-Manager on DigitalOcean Kubernetes

How To Set Up an Elasticsearch, Fluentd and Kibana (EFK) Logging Stack on Kubernetes

7 Comments

Leave a comment...

Log In to Comment

ciaran4d51781530ee70807025 *December 4, 2018*

1 Hi, thanks for the Tutorial. On a Digital Ocean managed Kubernetes Cluster I get the following installation error creating the dashboard pod.

panic: secrets is forbidden: User "system:serviceaccount:default:dashboard-kubernetes-dashboard" cannot create resource "secrets" in API group "" in the namespace "kube-system"

leianivey *December 12, 2018*

1 I have the exact same error

charlla *December 13, 2018*

1 I was able to get the dash up and running by specifying the namespace to be kube-system. You also then have to change all the references to default, to also be kube-system. Still have a couple of permission issues on the dash itself though, will see if I can sort them.

Command:

```
helm install stable/kubernetes-dashboard --name dashboard-demo --namespace=kube-system
```

Update:
More on this issue here:
https://github.com/helm/charts/issues/3104

ciaran4d51781530ee70807025 *December 13, 2018*

0 Thank you!

youwontforgetthis *December 21, 2018*

0 I then used a token from the tiller user to log-in to the dashboard. Not sure if that's the user you are supposed to use but the dashboard appears to work.

```
kubectl -n kube-system get secret

kubectl -n kube-system describe secret tiller-token-sjnbt
```

ciaran4d51781530ee70807025 *December 21, 2018*

0 thanks!

sean5d038c3161ae8150a304d6 *January 16, 2019*

0 Followed the steps and getting this when attempting to view dashboard through `kubectl proxy`

```
{
  kind: "Status",
  apiVersion: "v1",
  metadata: { },
  status: "Failure",
  message: "no endpoints available for service "https:dashboard:"",
  reason: "ServiceUnavailable",
  code: 503
}
```