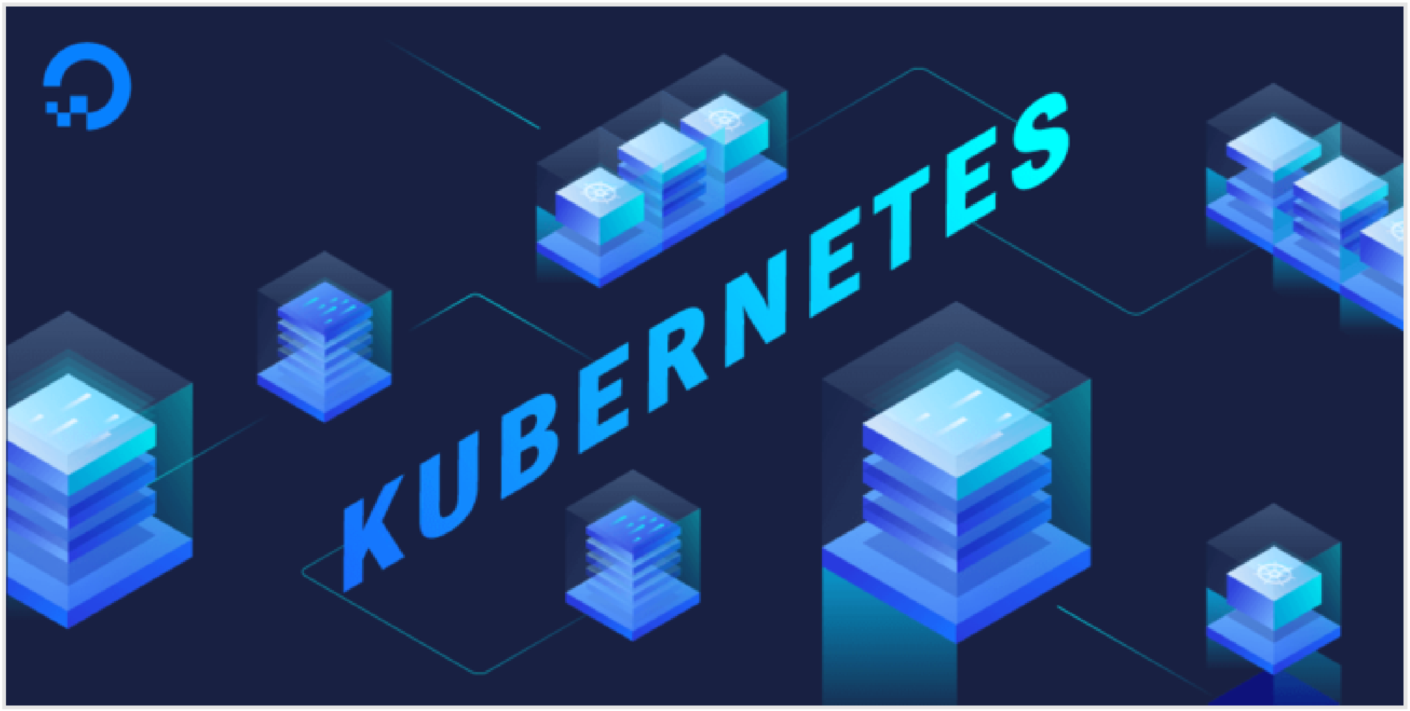


How To Install Software on Kubernetes Clusters with the Helm Package Manager

Posted August 15, 2018 71.7k KUBERNETES

By Brian Boucheron
Become an author

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 off

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
  *         do-nyc1-k8s-example    do-nyc1-k8s-example          do-nyc1-k8s-exampl

            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

SCROLL TO TOP

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-
```

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 resourc

SCROLL TO TOP

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 --serviceaccoun
```

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 Cluste

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

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

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

SCROLL TO TOP

```
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.

SCROLL TO TOP

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
dashboard-demo       1            Wed Aug  8 20:11:11 2018     DEPLOYED     kubernetes-dashb
```

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
dashboard-demo-kubernetes-dashboard   ClusterIP    10.32.104.73   <none>         443/TCP
kubernetes                             ClusterIP    10.32.0.1       <none>         443/TC
```

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`.

SCROLL TO TOP

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="dash
```

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.

SCROLL TO TOP

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
dashboard-demo  2         Wed Aug  8 20:13:15 2018    DEPLOYED    kubernetes-dashboard
```

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

SCROLL TO TOP

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
dashboard-demo  3           Wed Aug  8 20:15:21 2018    DELETED kubernetes-dashboard-0
```

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 H SCROLL TO TOP documentation.

By Brian Boucheron

Was this helpful? Yes No

17

Report an issue

Related

TUTORIAL

How To Autoscale Your Workloads on DigitalOcean Kubernetes

With Kubernetes applications that often have unexpected loads,...

CHEATSHEET

Getting Started with kubectl: A kubectl Cheat Sheet

kubectl is a command-line tool used to manage Kubernetes objects and...

TUTORIAL

How to Manage DigitalOcean and Kubernetes Infrastructure with Pulumi

Pulumi is a tool for

TUTORIAL

How To Deploy a PHP Application with Kubernetes on Ubuntu 18.04

Kubernetes is an open source container

SCROLL TO TOP

creating, deploying, and...

Still looking for an answer?

Ask a question

Search for more help

17 Comments

Leave a comment...

Log In to Comment

ciaran4d51781530ee70807025 *December 4, 2018*

2 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-k dashboard" cannot create resource "secrets" in API group "" in the namespace

SCROLL TO TOP

leianivey *December 12, 2018*

1 I have the exact same error

charlla *December 13, 2018*

2 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:

```
nstall stable/kubernetes-dashboard --name dashboard-demo --namespace=kube-syst
```

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!

SCROLL TO TOP

davedrager *June 26, 2019*

0 Was able to fix this by adding `--set rbac.clusterAdminRole=true` to the configuration of dashboard-demo. IE:

```
helm upgrade dashboard-demo stable/kubernetes-dashboard --set
fullnameOverride="dashboard" --set rbac.clusterAdminRole=true
```

sean5d038c3161ae8150a304d6 *January 16, 2019*

3 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
}
```

amartincolby *May 5, 2019*

1 At the risk of just making a "me too!" post, I am having the exact same problem. The insane thicket of often contradictory documentation and writing online makes finding insight into this problem borderline impossible.

cryto *September 9, 2019*

0 Did you manage to get it to work?

deepinthought *September 19, 2019*

0 How I fixed this error

- Step 1. - Copy token for login.

SCROLL TO TOP

```
$ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret

UVVd6n9BFSLTJ3HzD10O9Zm9pYJLWV5Oou48zNZXYMzsorL0B9jMNtcKWwBonVmRVzr8NO3ZW2jY
```

Or you can type this for the entire output.

```
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret |
```

- Step 2. - Next run the dashboard.

```
$ export POD_NAME=$(kubectl get pods -n kube-system -l "app=kubernetes-dashb
```

```
$ echo https://127.0.0.1:8443/
https://127.0.0.1:8443/
```

```
$ kubectl -n kube-system port-forward $POD_NAME 8443:8443
```

- Step 3.- A test curl should return an HTTP response of 200.

```
$ curl https://127.0.0.1:8443/ --insecure --head
HTTP/2 200
accept-ranges: bytes
cache-control: no-store
content-type: text/html; charset=utf-8
last-modified: Mon, 17 Dec 2018 09:04:43 GMT
content-length: 990
date: Thu, 19 Sep 2019 01:59:19 GMT
```

Open a browser and head over to https://127.0.0.1:8443/ and select Token.

Enter the token from Step 1.

SCROLL TO TOP

edited by MattIPv4

hasnaeen *February 1, 2019*

0 When I tried to run this,

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

Error: release dashboard-demo failed: namespaces "default" is forbidden: User "system:serviceaccount:kube-system:default" cannot get resource "namespaces" in API group "" in the namespace "default"

vantrixqa *April 17, 2019*

0 You probably ran `helm init` rather than `helm init --service-account tiller` (I know because I made the same mistake.).

If so, try running:

```
kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"tem
```

to specify the `serviceAccount` for your `tiller-deploy` deployment after having already initialised it. (You could also manually do it via `kubectl edit deployments -n kube-system tiller-deploy`).

EricMathison *February 21, 2019*

0 This isn't working for me with the latest update to 2.12.3. I cannot connect helm to tiller.

sbrady *May 21, 2019*

0 Yet another broken Kubernetes tutorial. Glad all these issues were fixed before going GA! /s

digitalocean457 *May 29, 2019*

SCROLL TO TOP

0 Hi, I had difficulty installing the dashboard and added a helm issue here:

https://github.com/helm/charts/issues/14289

along with a workaround.

arvtiwar *August 15, 2019*

0 Like it great work

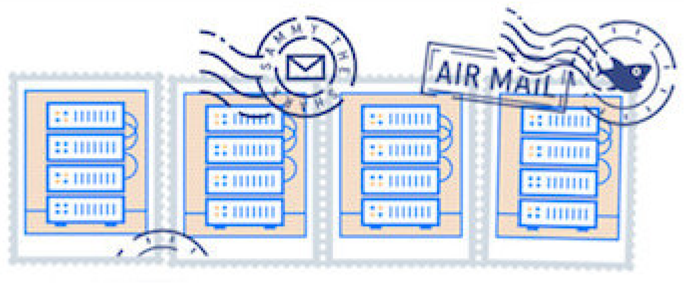
BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

SCROLL TO TOP

CONNECT WITH OTHER DEVELOPERS

Find a DigitalOcean Meetup near you.



GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

Featured on Community Intro to Kubernetes Learn Python 3 Machine Learning in Python Getting started with Go Migrate Node.js to Kubernetes

DigitalOcean Products Droplets Managed Databases Managed Kubernetes Spaces Object Storage Marketplace

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.

Learn More

SCROLL TO TOP

Company

About
Leadership
Blog
Careers
Partners
Referral Program
Press
Legal & Security

Products

Products Overview
Pricing
Droplets
Kubernetes
Managed Databases
Spaces
Marketplace
Load Balancers
Block Storage
Tools & Integrations
API
Documentation
Release Notes

Community

Tutorials
Q&A
Tools and Integrations
Tags
Product Ideas
Meetups
Write for DOnations
Droplets for Demos
Hatch Startup Program
Shop Swag
Research Program
Currents Research

Contact

Support
Sales
Report Abuse
System Status

SCROLL TO TOP

Open Source

SCROLL TO TOP