knóldus

knols . commitment . results

(https://www.knoldus.com/home)

June 26, 2020 (https://blog.knoldus.com/helm2-vs-helm3-simplified/) Yatharth Sharma (https://blog.knoldus.com/author/yatharthsharma4251/)

DevOps (https://blog.knoldus.com/category/tech-blogs/devops/), Tech Blogs (https://blog.knoldus.com/category/tech-blogs/)

Devops (https://blog.knoldus.com/tag/devops/), helm (https://blog.knoldus.com/tag/helm/), helm2 (https://blog.knoldus.com/tag/helm2/), helm3 (https://blog.knoldus.com/tag/helm3/), kubernetes (https://blog.knoldus.com/tag/kubernetes/)

Reading Time: 4 minutes

Hello readers, In this blog, I'll be covering the differences between the two versions of Helm, i.e. Helm2 and Helm3. The internal implementation of Helm3 has changed considerably from Helm2. If you want to know about what helm is, you can refer to my previous blog on Helm: Package Manager for Kubernetes. (Package Manager for Kubernetes.)
Going forward, let's start with the differences:

Tiller is Removed

The most apparent change is the removal of Tiller. The server-side component, Tiller, is now removed.

Why was it required?
Tiller was the server-side component which is used to maintain the state of helm release. It also stores all the release information in a config-map for every release in the same namespace where tiller is running. This config-map is used by helm whenever we try to upgrade a particular release, where helm compares the new manifest with the already present configs.

Why was tiller removed then?
Tiller is used by Helm to deploy the Kubernetes object. Previously, when Helm 2 was released, Kubernetes does not support RBAC policies, hence by default, Helm takes the maximum permission to make changes in Kubernetes. This can cause security issues in the cluster if Helm has not been properly deployed. But after Kubernetes 1.6, RBAC is enabled by default, so there is no need for Helm to keep track of who is allowed to installs what, as the same job can now be done natively by Kubernetes and that's why in Helm 3 tiller was removed completely. For knowing about possible security issues and how they can be solved, you can refer to this blog by bitnami (https://engineering.bitnami.com/articles/running-helm-in-production.html).

Improved Chart Upgrade Strategy: Three-way strategic merge patch

Helm 3 changed the upgrade strategy from two-way to three-way strategic merge patch. Let us see what they mean.

Helm 2 Upgrade Strategy
Helm 2 used a two-way strategic merge patch. Let's understand this is a simple way. When you deploy a release, it stores a config-map which contains the release configs. Now whenever you upgrade that

charge, it compares the newly supplied manifest with the existing manifest. This sounds good, but here exist one problem.

So what's the issue in two-way strategic merge?
The issue is that it does not consider the manual changes we have done using `kubectl edit` and when we upgrade the chart, it revert back those manual changes.

How is this resolved in Helm 3?
In Helm 3, we now use a three-way strategic merge patch. Helm considers the old manifest, its live state, and the new manifest when generating a patch. So when we try to upgrade any existing chart, it also considers our manual changes.

Scope for Release Names are limited to Namespaces now

Release name scope is now limited to only that namespace where it is deployed. Previously, the release name should we unique across the cluster, but now it is limited to a particular namespace.

An example to make understanding better
Previous, if we want to deploy any service, say `mysql`, we would be doing like this:
Deployment 1: `helm install stable/mysql --name mysql-develop --namespace develop`
Deployment 2: `helm install stable/mysql --name mysql-qa --namespace qa`
Now the same deployment can be achieved like this:
Deployment 1: `helm install stable/mysql --name mysql --namespace develop`
Deployment 2: `helm install stable/mysql --name mysql --namespace qa`

Secrets as default storage driver

Helm 2 used config-maps to store release information. In Helm 3, secrets are used instead as the default storage driver. Also, they are stored in the same namespace where the release is deployed. Previously, the release information was stored in the same namespace where tiller is deployed.

So what are the benefits of using secrets?
Helm 2 doesn't store the release information directly, rather it performs a couple of encryption operations to store it in config-map. Also, at the time of retrieving those configs, helm has to re-perform

those steps to decrypt that information. This whole process is simplied by having secrets in place. Helm 3 directly pull out the secrets, decrypt it and apply.

JSON Schema Chart Validation

A JSON Schema validation can now be forced upon chart values. It provides a schema which enforces a format over the values provided by the users. This helps in providing better error reporting if a user tries to use an incorrect set of values. This JSON Validation takes place when we run any of the following commands:

1. `helm install`
2. `helm upgrade`
3. `helm template`
4. `helm lint`

Release name is now required

Previously in Helm 2, the release name is provided by the flag `--name <release name>` . In case the flag is not supplied, Helm automatically provides a random name to the chart.
But in Helm 3, release name is compulsory. It follows a format like: `helm install <chart-name> <release-name>`. If no name is provided to a chart, it will throw an error. Eventually, you can also you `--generate-name` if you want to auto-generate a random release-name for your chart.

Namespaces not created automatically

In Helm2, if you deploy a chart in a non-existing namespace, it will create the namespace automatically. This is not the case with Helm3. In Helm3, you need to create the namespace first in order to deploy a release in that namespace.

Consolidation of `requirement.yaml` into `Chart.yaml`

The Chart dependency management system moved from `requirements.yaml` and `requirements.lock` to `Chart.yaml` and `Chart.lock`. The dependencies, which earlier were provided in `requirement.yaml`, are now provided in `chart.yaml`.

Removal of `helm serve`

↑

Helm `serve` command starts a local chart repository server that serves charts from a local directory. This has been removed in Helm3 version. If you still want to have a similar experience, you can have a look at ChartMuseum (https://chartmuseum.com/)

Chart.yaml `apiVersion` bump

As there are multiple enhancements in Helm3, the `apiVersion`, which we specify in the `Chart.yaml` file has bumped from `v1` to `v2`.

CLI Commands Renamed

Some CLI commands are also re-named in Helm3

`helm delete` is re-named to `helm uninstall`. But `helm delete` is still retained as an alias to `helm uninstall`. Also, in Helm2 , we use the `--purge` flag to remove the release configs also. In Helm3, this feature is enabled by default. To retain the previous behaviour, you can use the command `helm uninstall --keep-history`.

`helm inspect` command is renamed to `helm show`

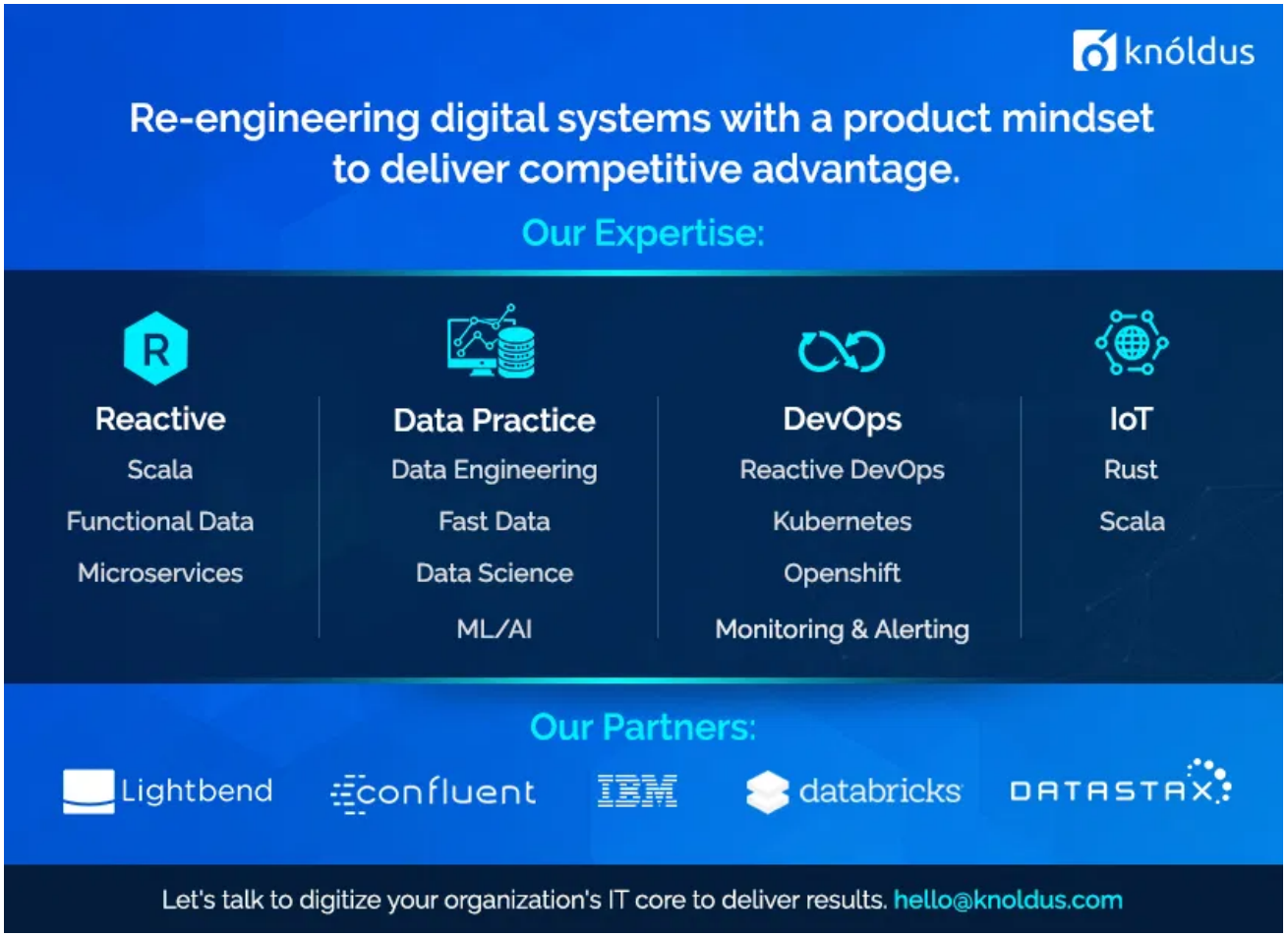
`helm fetch` command is renamed to `helm pull`

I've covered mostly all the features that were introduced in Helm3. Still, If you have any doubts or queries, feel free to contact me at yatharth.sharma@knoldus.in (https://blog.knoldus.com/maven-simplified/yatharth.sharma@knoldus.in).

Here are my references :
1. Helm Documentation (https://helm.sh/docs/faq/)
2. Helm2 vs Helm3 Dawid Ziolkowski (https://v2.helm.sh/docs/helm/#helm)

Also, I would like to thank you for sticking to the end. If you like this blog, please do show your appreciation by giving thumbs-ups and share this blog and give me suggestions on how I can improve my future posts to suit your needs. Follow me to get updates on different technologies.

↑



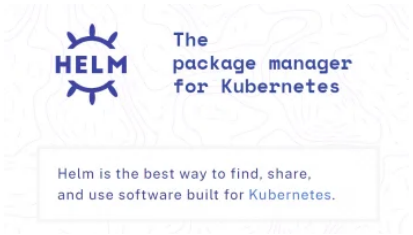
Related

(/manage-your-kubernetes-applications-with-helm/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=0)

Manage your Kubernetes applications with Helm (/manage-your-kubernetes-applications-with-helm/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=0&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=0)

May 1, 2019

In "DevOps"



(/helm-v2-package-manager-for-kubernetes/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=1)

Helm v2: Package Manager for Kubernetes (/helm-v2-package-manager-for-kubernetes/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=1&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=1)

June 4, 2020

In "DevOps"

(/upgrade-eks-to-1-16-using-terraform/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=2)

Upgrade EKS to 1.16 using Terraform (/upgrade-eks-to-1-16-using-terraform/?relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=2&relatedposts_hit=1&relatedposts_origin=74539&relatedposts_position=2)

June 22, 2020

In "AWS"

Written by Yatharth Sharma

(https://blog.knoldus.com/author/yatharthsharma4251/)

Yatharth Sharma is a Software Consultant at Knoldus Software LLP. He has done MCA from Bharati Vidyapeeth Institute of Computer Application and Management, Paschim Vihar. He has a decent knowledge of Java Language and currently working on DevOps technologies/tools like Ansible, Jenkins, Docker, Kubernetes. Apart from programming, he loves listening to rap music.

Knolway (https://www.knoldus.com/about/process)

Case Studies (https://www.knoldus.com/work/case-studies)

Testimonials (https://www.knoldus.com/about/testimonials)

Careers (https://www.knoldus.com/company/careers)

Tech Expertise (https://www.knoldus.com/work/technologies)

Conferences and Events (https://www.knoldus.com/about/events/conferences)

News Room (https://www.knoldus.com/news)

Our Partners (https://www.knoldus.com/partners)

CSR (https://www.knoldus.com/about/corporate-social-responsibility)

LEARN

Why Knoldus? (https://www.knoldus.com/connect/why-knoldus)

Blog (https://blog.knoldus.com/)

Resources (https://www.knoldus.com/learn/resources)

KnolX (http://knolx.knoldus.com/session)

Webinars and Videos (https://www.knoldus.com/learn/webinars)

Podcast (https://www.knoldus.com/learn/podcasts)

Rust Times (https://rusttimes.com/)

Innovation Labs (https://www.knoldus.com/about/innovation-labs)

Knoldus Tech Hub (https://techhub.knoldus.com/)

Open Source Contributions (https://www.knoldus.com/work/open-source)

CONNECT

Contact Us (https://www.knoldus.com/connect/contact-us)

Engagement Model (https://www.knoldus.com/connect/engagement-models)

Follow us Here:

(https://in.linkedin.com/company/knoldus) (https://www.youtube.com/channel/UCP4g5qGeUSY7OokXfim1QCQ) (https://twitter.com/Knolspeak?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor) (https://www.facebook.com/KnoldusSoftware/) (https://www.slideshare.net/knoldus) (https://www.instagram.com/knoldus_inc/?hl=en)

Subscribe to our newsletter

SUBSCRIBE (https://share.hsforms.com/1oRAE_GdlQsi3niRYzwv65g2u6gi)

Partners

(https://www.lightbend.com/partners/system-integrators)

(https://databricks.com/company/partners)

(https://www.confluent.io/partners/)

(https://www.docker.com/)

(https://www.hashicorp.com/ecosystem/find-a-partner/)

(https://www.ibm.com/partnerworld/public)