Published in The Startup

Wojciech Krzywiec Follow
Apr 16, 2020 · 10 min read · Listen

Save

How to declaratively run Helm charts using helmfile

Have you ever wonder how to achieve fully Infrastructure as a Code approach for deployment of your applications on Kubernetes cluster? No matter if you've already achieve it or just started to looking for the best tool, helmfile might be the answer for you. In this blog post I'll present how you can set up your first helmfile.

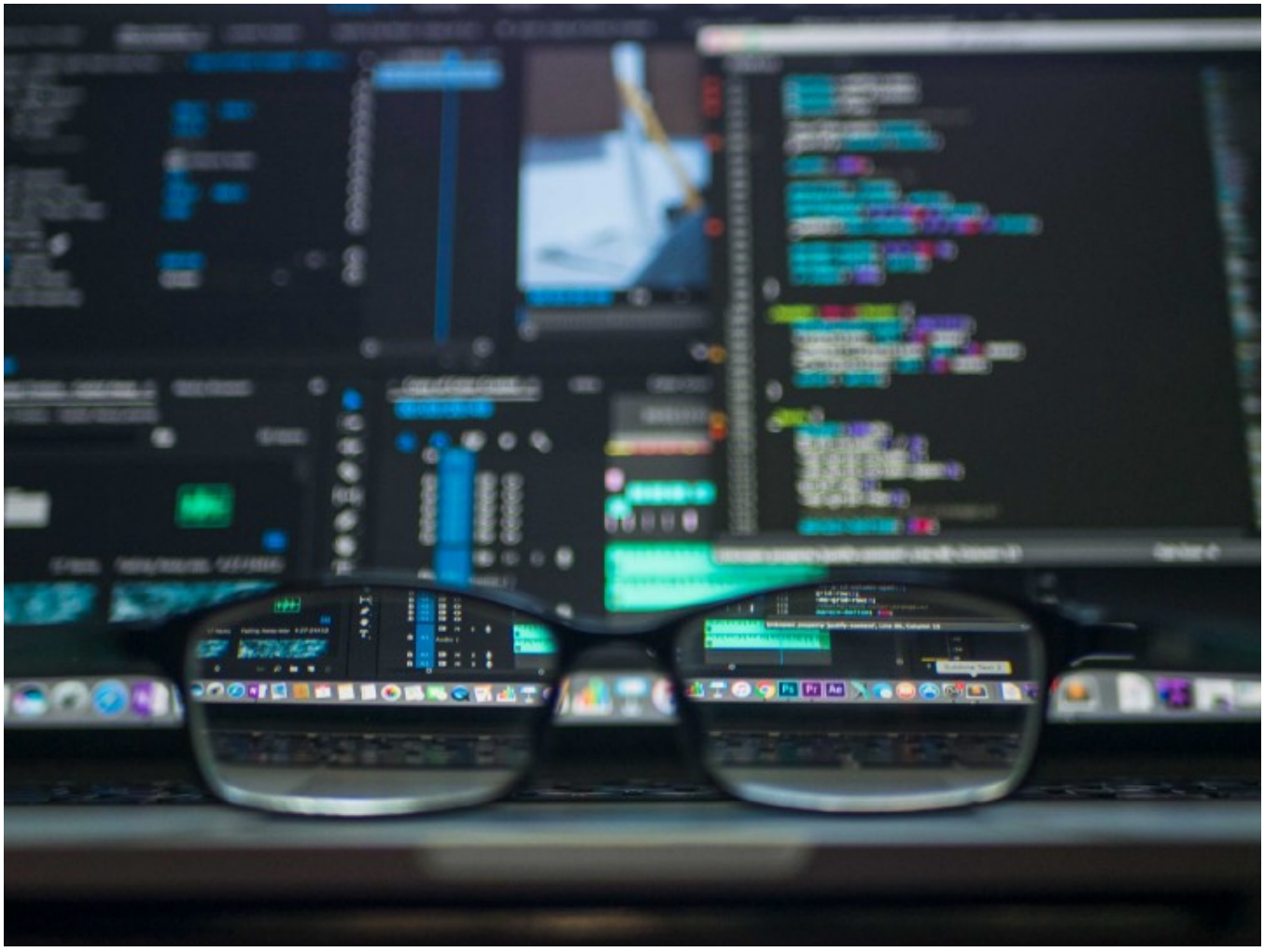
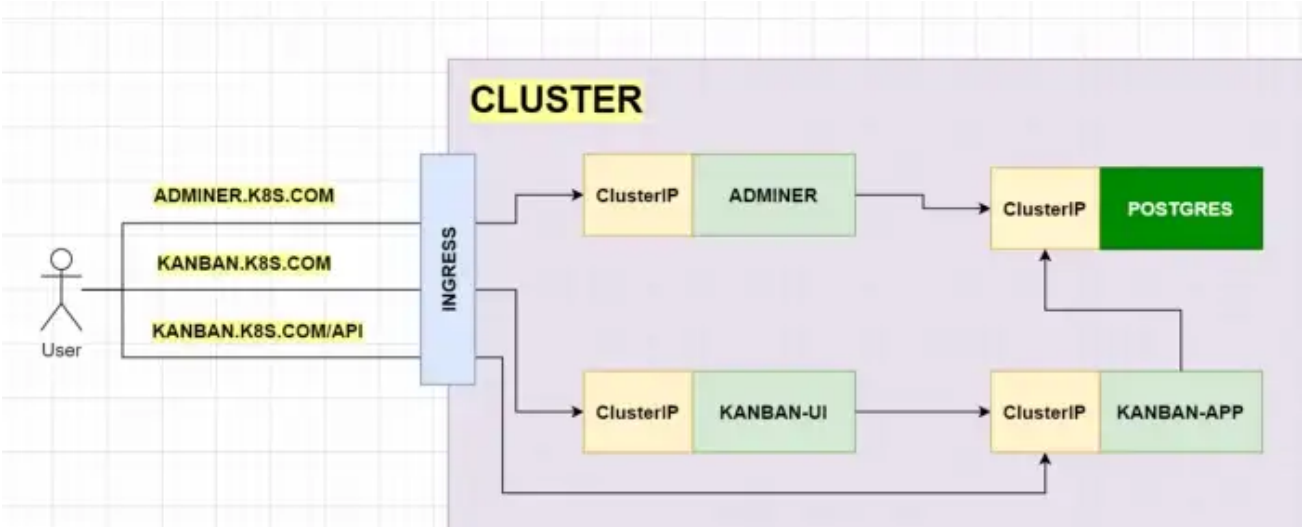


Photo by Kevin Ku on Unsplash

This post is a third, and the last, from my series on Kubernetes. In each one of them I present how to deploy applications on a Kubernetes cluster using one of the approaches:

- *using **kubectl** — Deployment of multiple apps on Kubernetes cluster — Walkthrough,*
- *using **Helm** — How to deploy application on Kubernetes with Helm,*
- *using **helmfile** — this one.*

If you haven't read first two I would recommend to do that before diving into this one. But it's not the must, I'll briefly introduce you to an environment which will be built.



Above picture presents the target solution. Inside the cluster there will be deployed frontend (*kanban-ui*) and backend (*kanban-app*). To persist data postgres database will be used. A source code for both microservices can be found in my GitHub repository — kanban-board.

Additionally I've added an *adminer* application, which is a GUI client for getting inside the database.

The above picture shows not only applications that will be deployed on *Kubernetes* cluster, but also objects that are required to run them, like *Deployments* or *Services*. With classic, *kubectl* approach, we were forced to create definition files for each one of them. Even if in most cases, for example in case of *ClusterIPs*, these objects haven't differ that much between applications.

Definitely this approach doesn't scale in microservice world, where usually there is a need to deploy dozens or even hundreds of applications. And if for each one of them we would be require to create couple YAML files it would really quick become a nightmare. But this problem might be solved with *Kubernetes* templating engine — *Helm*. We can define a

generic *Helm* chart (template) and reused it across multiple applications only by injecting specific values into the template.

> *Is it problem solved?*

To some extend yes, but here is a drawback of using *Helm*. In order to install or update any chart in a cluster you need to run a specific imperative command. In other words, in order to change state of a cluster you need to run a command that is specific for a given deployment.

In Java world we're using Maven or for JavaScript we use *npm*. Both of these tools allows to run the same command across multiple projects to perform the same action like running tests— `mvn test`. Helm currently doesn't support this feature, we're not able to install, update or rollback all applications from an entire cluster with a single command.

But there is a helmfile, which might be a remedy.

Helmfile allows to declare a definition of an entire *Kubernetes* cluster in a single YAML file, it bundles multiple *Helm* releases (installation of *Helm* charts) and allows to tune a specification of each release depending on a type of environment (develop, test, production) on which you might want to deploy your applications.

The best approach would be to show how it works on an example. Before that we need to set up a working environment. If you want to follow my steps, you would need to install and configure:

- Docker,
- locally installed Kubernetes cluster — minikube,
- Kubernetes command line tool — kubectl,
- Helm (v3),
- helmfile.

When everything is installed you can start up the *minikube* cluster and enable ingress addon:

```
$ minikube start
😄  minikube v1.8.1 on Ubuntu 18.04
✨  Automatically selected the docker driver
🔥  Creating Kubernetes in docker container with (CPUs=2) (8 available),
Memory=2200MB (7826MB available) ...
🐳  Preparing Kubernetes v1.17.3 on Docker 19.03.2 ...
```

```
    ▪ kubeadm.pod-network-cidr=10.244.0.0/16
❌  Unable to load cached images: loading cached images: stat
/home/wojtek/.minikube/cache/images/k8s.gcr.io/kube-proxy_v1.17.3: no such
file or directory
🚀  Launching Kubernetes ...
🌟  Enabling addons: default-storageclass, storage-provisioner
⌛  Waiting for cluster to come online ...
🏄  Done! kubectl is now configured to use "minikube"$ minikube addons
enable ingress
🌟  The 'ingress' addon is enabled
```

Then you'll need to edit you **hosts** file. Its location varies on OS:

- Ubuntu
- Windows
- MacOS

When you find it add following lines:

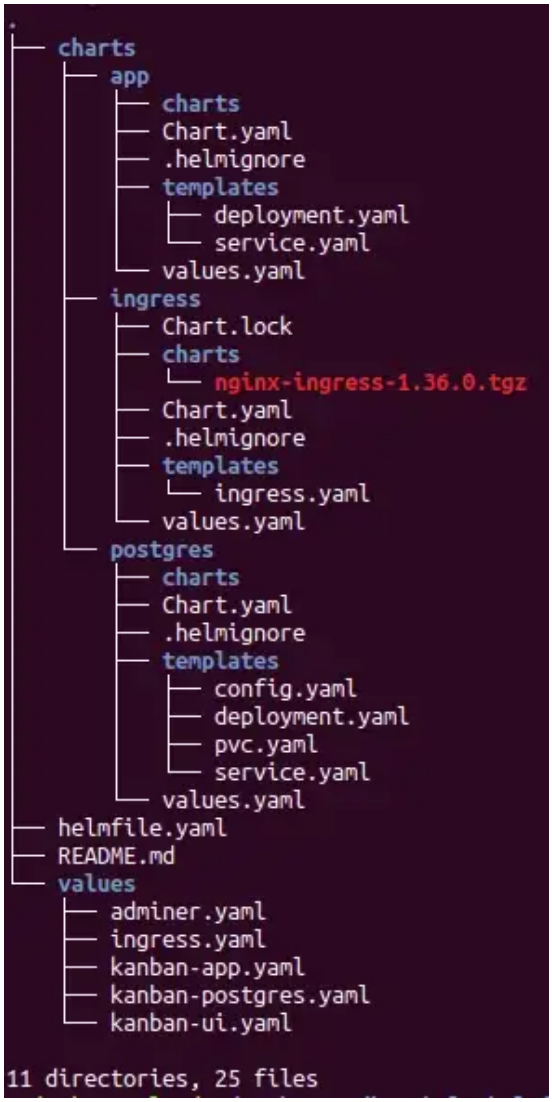
```
172.17.0.2  adminer.k8s.com
172.17.0.2  kanban.k8s.com
```

To make sure that this config is correct you'll need to check if an IP address of *minikube* cluster on your machine is the same as it's above. In order to do that run the command:

```
$ minikube ip
172.17.0.2
```

A preparations are done, let's do some *helmfile* magic. In previous blog post — *How to deploy application on Kubernetes with Helm* — I've described how to create Helm charts for Kanban Board project. Here we will reuse them. Therefore copy-paste all the files from here — k8s-helm-helmfile/helm.

All folders — `app` , `ingress` , `postgres` — put inside a new one called `charts` and all YAML files put inside a new folder called `values` . Then add a new `helmfile.yaml` file to a root directory, so the resulting structure will looks as follows:



Let me briefly explain what each of these folders means:

- `./charts` — here are all three Helms charts that are templates for each release — `app` (for *adminer*, *kanban-ui*, *kanban-app*), `postgres` and `ingress` ,
- `helmfile.yaml` — here will be a configuration for a *helmfile*, currently blank,
- `./values` — a folder with a values that are specific for each application which will be deployed.

Before moving to filling a `helmfile.yaml` I want to adjust one thing. In a previous, native *Helm,* solution there is an `ingress` chart, which has a dependency to nginx-ingress Helm chart available on Helm Hub. Before installing it on a cluster we needed to download it (will be saved in `.charts/ingress/charts` folder) using separate helm command.

If we use *helmfile,* the last step would not be needed, because it will figure out and download all required dependencies inside each chart. So we could keep `ingress` chart as it is, but I would like to have a different approach this time. I would like to treat *nginx-*

ingress chart as a separate *Helm* release, apart from the *Ingress Controller* configuration for routing.

Therefore remove following lines from the `./charts/ingress/Chart.yaml` file:

```
dependencies:
    - name: nginx-ingress
      version: 1.36.0
      repository: https://charts.helm.sh/stable
```

Also you can delete the `Chart.lock` file entirely.

After those changes `ingress` *Helm* chart is responsible only for defining *Ingress Controller.* The required Ingress backend service will be introduced with Helm release.

Now, finally we can move to arranging a `helmfile.yaml` .

First we need to indicate from which *Helm* repositories we would like to download charts. Because we will be downloading only one — *stable/nginx-ingress* — chart from a single repo, we put one entry:

```
        url: https://charts.helm.sh/stable
```

Then we can move on to next section — defining the *Helm* releases. It will contain a list of all applications that *helmfile* will need to deploy. Starting from the first one — `postgres` :

```
releases:
    - name: postgres
      chart: ./charts/postgres
      values:
          - ./values/kanban-postgres.yaml
```

There is not that much here, but for clarity:

- `name` — is a name of a release, we can name is whatever we like,
- `chart` — tells *helmfile* where a base *Helm* chart is located,

- `values` — because we're using a *Helm* chart template, we would like to inject some values into it, so here is a list of *YAML* files that holds them.

Next, let's define releases that are based on `app` chart:

```
releases:
    - name: adminer
      chart: ./charts/app
      values:
          - ./values/adminer.yaml

    - name: kanban-app
      chart: ./charts/app
      values:
          - ./values/kanban-app.yaml

    - name: kanban-ui
      chart: ./charts/app
      values:
          - ./values/kanban-ui.yaml
```

And next, we define a*Helm* release for an *Ingress* backend, which will be downloaded from a public repository:


```
releases:
    - name: ingress-backend
      chart: stable/nginx-ingress
      version: 1.36.0
```

In a `chart` section we provide information of which chart we would like to download and install. The `stable/` prefix is a name of a repository that was defined in a `repositories` few moments ago.

In `version` we specify which version of a chart to use.

And finally a definition for *Ingress Controller Helm* release:

```
releases:
    - name: ingress-controller
      chart: ./charts/ingress
      values:
          - ./values/ingress.yaml
```

And if we merge all of them, `helmfile.yaml` will look as follows:

```yaml
repositories:
  - name: stable
    url: https://kubernetes-charts.storage.googleapis.com

releases:
  - name: postgres
    chart: ./charts/postgres
    values:
      - ./values/kanban-postgres.yaml

  - name: adminer
    chart: ./charts/app
    values:
      - ./values/adminer.yaml

  - name: kanban-app
    chart: ./charts/app
    values:
      - ./values/kanban-app.yaml

  - name: kanban-ui
    chart: ./charts/app
    values:
      - ./values/kanban-ui.yaml

  - name: ingress-backend
    chart: stable/nginx-ingress
    version: 1.36.0

  - name: ingress-controller
    chart: ./charts/ingress
    values:
      - ./values/ingress.yaml
```

helmfile.yaml hosted with ❤ by **GitHub** **view raw**

And now it is a fun part, with just two commands we can deploy all applications.

First, we need to add the repository to our instance of Helm (you'll need to do that only once):

```
$ helmfile repos
Adding repo stable https://charts.helm.sh/stable
"stable" has been added to your repositories

Updating repo
Hang tight while we grab the latest from your chart repositories...
```

```
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈ Happy Helming!⎈
```

Then install all charts on a cluster:

```
$ helmfile sync
Adding repo stable https://charts.helm.sh/stable
"stable" has been added to your repositories

Updating repo
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "stable" chart repository
Update Complete. ⎈ Happy Helming!⎈

Building dependency release=postgres, chart=charts/postgres
Building dependency release=adminer, chart=charts/app
Building dependency release=kanban-app, chart=charts/app
Building dependency release=kanban-ui, chart=charts/app
Building dependency release=ingress-controller, chart=charts/ingress
Affected releases are:
  adminer (./charts/app) UPDATED
  ingress-backend (stable/nginx-ingress) UPDATED
  ingress-controller (./charts/ingress) UPDATED
  kanban-app (./charts/app) UPDATED
  kanban-ui (./charts/app) UPDATED
  postgres (./charts/postgres) UPDATED

Upgrading release=postgres, chart=charts/postgres
Upgrading release=kanban-app, chart=charts/app
Upgrading release=adminer, chart=charts/app
Upgrading release=kanban-ui, chart=charts/app
Upgrading release=ingress-controller, chart=charts/ingress
Upgrading release=ingress-backend, chart=stable/nginx-ingress
Release "adminer" does not exist. Installing it now.
NAME: adminer
LAST DEPLOYED: Wed Apr 15 16:16:31 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

Listing releases matching ^adminer$
Release "kanban-ui" does not exist. Installing it now.
NAME: kanban-ui
LAST DEPLOYED: Wed Apr 15 16:16:31 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

Listing releases matching ^kanban-ui$
adminer default   1        2020-04-15 16:16:31.990139954 +0200 CEST
deployed app-0.1.0 1.16.0

Release "kanban-app" does not exist. Installing it now.
NAME: kanban-app
LAST DEPLOYED: Wed Apr 15 16:16:31 2020
```

```
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

Listing releases matching ^kanban-app$
kanban-ui default   1        2020-04-15 16:16:31.968291217 +0200 CEST
deployed app-0.1.0 1.16.0

Release "postgres" does not exist. Installing it now.
NAME: postgres
LAST DEPLOYED: Wed Apr 15 16:16:31 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

Listing releases matching ^postgres$
kanban-app default   1        2020-04-15 16:16:31.958860307 +0200 CEST
deployed app-0.1.0 1.16.0

postgres default   1        2020-04-15 16:16:31.958951797 +0200 CEST
deployed postgres-0.1.0 1.16.0

Release "ingress-controller" does not exist. Installing it now.
NAME: ingress-controller
LAST DEPLOYED: Wed Apr 15 16:16:31 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None

Listing releases matching ^ingress-controller$
ingress-controller default   1        2020-04-15 16:16:31.958844983 +0200
CEST deployed ingress-0.1.0 1.16.0

Release "ingress-backend" does not exist. Installing it now.
NAME: ingress-backend
LAST DEPLOYED: Wed Apr 15 16:16:35 2020
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The nginx-ingress controller has been installed.
It may take a few minutes for the LoadBalancer IP to be available.
You can watch the status by running 'kubectl --namespace default get
services -o wide -w ingress-backend-nginx-ingress-controller'

An example Ingress that makes use of the controller:

apiVersion: extensions/v1beta1
  kind: Ingress
  metadata:
    annotations:
      kubernetes.io/ingress.class: nginx
    name: example
    namespace: foo
  spec:
    rules:
      - host: www.example.com
        http:
```

```
            paths:
              - backend:
                  serviceName: exampleService
                  servicePort: 80
                path: /
    # This section is only required if TLS is to be enabled for the
Ingress
    tls:
        - hosts:
            - www.example.com
          secretName: example-tls

If TLS is enabled for the Ingress, a Secret containing the certificate and
key must also be provided:

apiVersion: v1
  kind: Secret
  metadata:
    name: example-tls
    namespace: foo
  data:
    tls.crt: <base64 encoded cert>
    tls.key: <base64 encoded key>
  type: kubernetes.io/tls

Listing releases matching ^ingress-backend$
ingress-backend default   1        2020-04-15 16:16:35.070513181 +0200
CEST deployed nginx-ingress-1.36.0 0.30.0

UPDATED RELEASES:
NAME                CHART                   VERSION
adminer             ./charts/app              0.1.0
kanban-ui           ./charts/app              0.1.0
kanban-app          ./charts/app              0.1.0
postgres            ./charts/postgres         0.1.0
ingress-controller  ./charts/ingress          0.1.0
ingress-backend     stable/nginx-ingress     1.36.0
```

After couple minutes you will be able to enter http://kanban.k8s.com.

Conclusion

And that's it! Very simple and everything — external repository set up and a list of each *Helm* releases is in one single file, which can be stored under version control system (*git* for example).

And what is more to mention theses are not the only features of *helmfile*. More of them you can find in the official documentation.

To sum up the entire series, I hope you've learn something and you find out what are the difference for deployments in native *Kubernetes*, *Helm* and *helmfile*. I hope it's more clear for you how each one of them look like and what are they pros and cons. You may

comment it here or in other blog post what do you like and what you think is worth to correct from my side. I would be grateful.

As usual, here is a link to a source code — to this project:

wkrzywiec/k8s-helm-helmfile

With this project I want to compare 3 approaches of deploying same applictions to Kubernetes cluster: k8s - the entire...

github.com

And to kanban-board project:

wkrzywiec/kanban-board

This is a simple implementation of a Kanban Board, a tool that helps visualize and manage work. Originally it was first...

github.com

References

roboll/helmfile

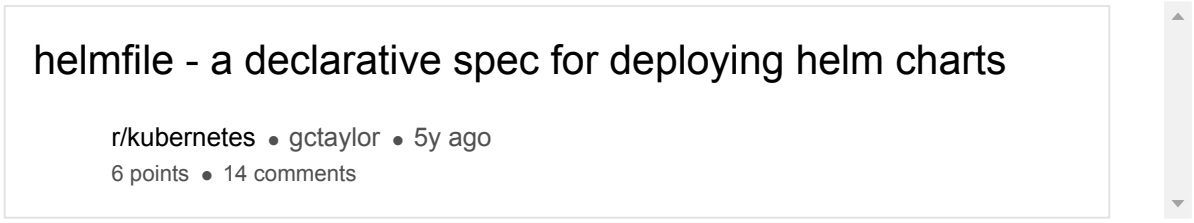
Deploy Kubernetes Helm Charts Even though Helmfile is used in production environments across multiple organizations, it...

github.com

cloudposse/helmfiles

Helmfiles is a comprehensive distribution of declarative chart invocations. It makes it really easy to get up and...

github.com



Dev Ops Software Engineering Helm Kubernetes Cloud Computing
