

Helm 简介和使用

AUTHOR: 彭玲 TIME: 2021/10/28

Helm 简介和使用

- Helm 简介
 - Helm 解决了什么痛点?
 - Helm 架构
 - Helm 重要概念
 - Helm 安装
 - Chart 文件结构
 - Chart 基本结构
 - Subchart 依赖文件
 - Chart 模板
 - 创建一个 Nginx 的 Chart
 - 模板渲染
 - 渲染结果 (`result/`)
 - `deployment.yaml`
 - `service.yaml`
 - `serviceaccount.yaml`
 - 添加模板文件
 - ConfigMap 模板
 - 渲染结果
 - 模板中使用变量
 - 定义变量
 - 重新渲染
 - 不同环境 变量设置
 - 通过命令行设置变量
 - Helm vs. kubebuilder 总结
-

Helm 简介

Helm 是 Kubernetes 的包管理工具，用来简化 K8S 应用的部署和管理。

在 Kubernetes 中部署容器云应用（容器或微服务编排）是一项有挑战性的工作，Helm 就是为了简化在 Kubernetes 中安装部署容器云应用的一个客户端工具。通过 Helm 能够帮助开发者定义、安装和升级 Kubernetes 中的容器云应用，以及容器云应用的分享。

Helm 解决了什么痛点?

在 Kubernetes 中部署一个可以使用的应用，需要涉及到很多的 Kubernetes 资源的共同协作。

比如，安装一个 WordPress 博客，用到了一些 Kubernetes 的一些资源对象，包括 Deployment 用于部署应用、Service 提供服务发现、Secret 配置 WordPress 的用户名和密码，可能还需要 PV 和 PVC 来提供持久化服务。并且 WordPress 数据是存储在 mariadb 里面的，所以需要 mariadb 启动就绪后才能启动 WordPress。这些 k8s 资源过于分散，不方便进行管理，直接通过 kubectl 来管理一个应用，会发

现这十分繁琐。

通过 Helm 在 k8s 中部署一个应用，可以解决以下几个问题：

- 统一管理、配置和更新这些分散的 k8s 的应用资源文件。
- 分发和复用一套应用模板。
- 将应用的一系列资源当做一个软件包管理。

Helm 架构

使用 Helm 能够从 Chart repository (Helm 应用仓库) 快速查找、下载安装软件包，并通过与 K8s API Server 交互构建应用。

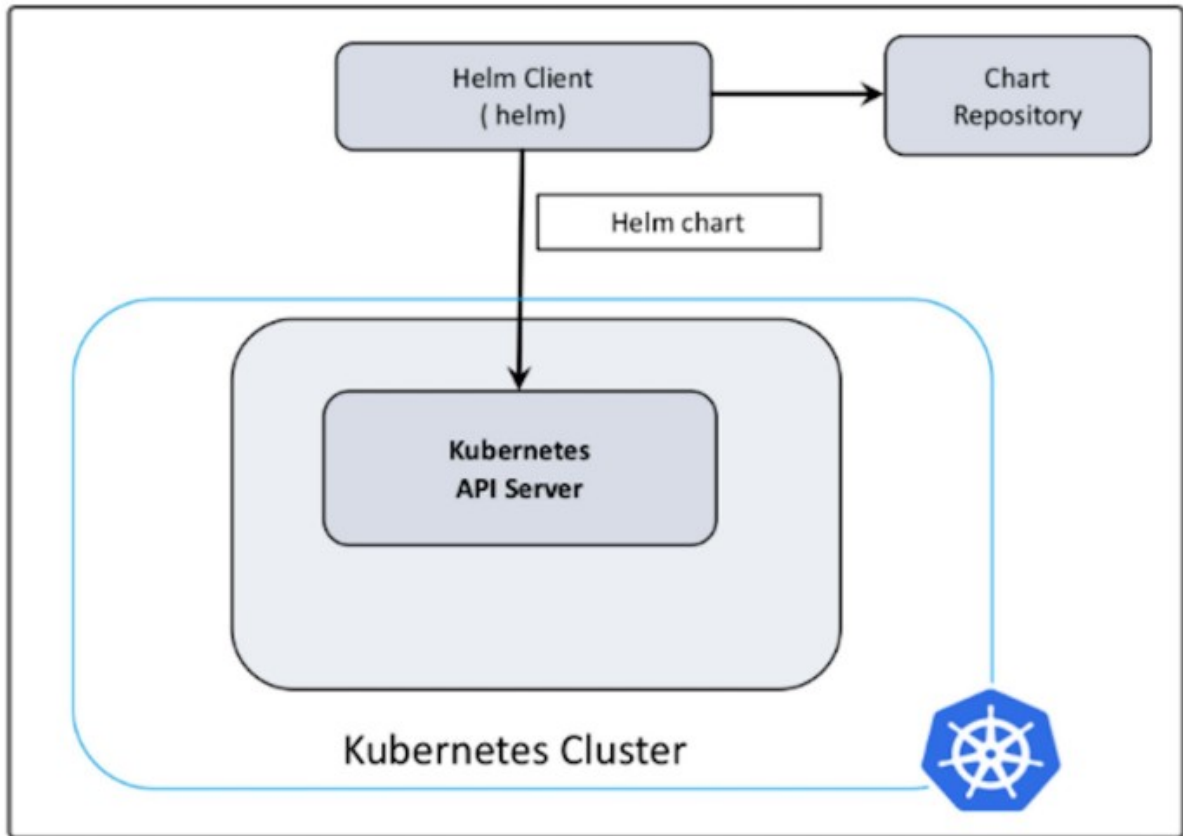


图1 Helm3 Workflow

Helm 重要概念

在 Helm 中，有三个需要了解的重要概念：Chart、Config、Release。

- Charts: Helm 使用的打包格式，一个 Chart 包含了一组 K8s 资源集合的描述文件。
- Config: 创建发布对象的 Chart 的配置信息。
- Release: Chart 的运行实例，包含特定的 Config。

Helm 安装

Helm 可以用 [源码](#) 或 [构建的二进制版本](#) 安装。这里使用官方提供的 [脚本](#) 一键安装：

```
1 # 从 https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get 获取
  脚本，并保存到 get_helm.sh
2 $ curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get >
  get_helm.sh
3
4 $ chmod 700 get_helm.sh
5
6 $ ./get_helm.sh
```

Chart 文件结构

Helm 的打包格式叫做 `chart`，所谓 `chart` 就是一系列文件，它描述了一组相关的 k8s 集群资源。

Chart 基本结构

使用 `helm create my-nginx` 创建一个 Chart。该 Chart 目录如下所示：

```
1 anxin@node38:~/pengling/k8s/helm-demo$ tree
2 .
3 └─ my-nginx
4     │   └─ charts # 存放依赖的 chart
5     │   └─ Chart.yaml # 该 Chart 的描述（包含 Chart 的基本信息，包括 chart 版本，名称
6     │   等）。
7     │   └─ templates # 放置模板文件（存放应用一系列 k8s 资源的 yaml 模板），想要渲染什么
8     │   文件出来，就在这个目录放置对应模板。
9     │       │   └─ deployment.yaml
10    │       │   └─ _helpers.tpl # 定义一些可重用的模板片断，此文件中的定义在任何资源定义模板
11    │       │   中可用。
12    │       │   └─ hpa.yaml
13    │       │   └─ ingress.yaml
14    │       │   └─ NOTES.txt # 介绍 chart 部署后的帮助信息，如何使用 chart 等。
15    │       │   └─ serviceaccount.yaml
16    │       │   └─ service.yaml
17    │       │   └─ tests
18    │       │       └─ test-connection.yaml
19    └─ values.yaml # 包含了必要的值定义（变量默认值），用于存储 templates 目录中模板
    文件中用到变量的值。
```

Subchart 依赖文件

使用 `helm fetch bitnami/wordpress --untar` 下载并解压后，查看 `wordpress chart` 包的 `Chart.yaml`。

其中 `dependencies` 属性下列出了依赖（Subchart），如 `mariadb` 等。

```
1 annotations:
2   category: CMS
3 apiVersion: v2
4 appVersion: 5.8.1
5 dependencies: # 依赖（子 Chart）
6   - condition: mariadb.enabled
```

```
7 name: mariadb
8 repository: https://charts.bitnami.com/bitnami
9 version: 9.x.x
10 - condition: memcached.enabled
11 name: memcached
12 repository: https://charts.bitnami.com/bitnami
13 version: 5.x.x
14 - name: common
15 repository: https://charts.bitnami.com/bitnami
16 tags:
17 - bitnami-common
18 version: 1.x.x
19 description: web publishing platform for building blogs and websites.
20 home: https://github.com/bitnami/charts/tree/master/bitnami/wordpress
21 icon: https://bitnami.com/assets/stacks/wordpress/img/wordpress-stack-
220x234.png
22 keywords:
23 - application
24 - blog
25 - cms
26 - http
27 - php
28 - web
29 - wordpress
30 maintainers:
31 - email: containers@bitnami.com
32 name: Bitnami
33 name: wordpress
34 sources:
35 - https://github.com/bitnami/bitnami-docker-wordpress
36 - https://wordpress.org/
37 version: 12.1.21
```

Chart 模板

模板渲染是 Helm 的核心内容，主要依赖其模板语言 (`template/`) 和内置对象 (`Chart`, `Template`, `Values` 等)。

创建一个 Nginx 的 Chart

使用 Helm 的 `Template` 功能，需要先创建一个 Chart，这是 Helm 的基本文件组成架构。

```
1 anxin@node38:~/pengling/k8s/helm-demo$ helm create my-nginx
2 Creating my-nginx
```

命令执行完成后，就会自动创建 Chart 的相关文件：

```
1 anxin@node38:~/pengling/k8s/helm-demo$ tree
2 .
3 └─ my-nginx
4     └─ charts # 存放依赖的 chart
5     └─ Chart.yaml # 该 Chart 的描述（包含 Chart 的基本信息，包括 chart 版本，名称等）。
```

```

6   └─ templates # 放置模板文件（存放应用一系列 k8s 资源的 yaml 模板），想要渲染什么
   文件出来，就在这个目录放置对应模板。
7   |   └─ deployment.yaml
8   |   └─ _helpers.tpl # 定义一些可重用的模板片断，此文件中的定义在任何资源定义模板
   中可用。
9   |   └─ hpa.yaml
10  |   └─ ingress.yaml
11  |   └─ NOTES.txt # 介绍 chart 部署后的帮助信息，如何使用 chart 等。
12  |   └─ serviceaccount.yaml
13  |   └─ service.yaml
14  |   └─ tests
15  |       └─ test-connection.yaml
16  └─ values.yaml # 包含了必要的值定义（变量默认值），用于存储 templates 目录中模板
   文件中用到变量的值。
17
18  4 directories, 10 files

```

模板渲染

不修改模板、不添加变量，使用 `helm template my-nginx/ --output-dir ./result` 直接渲染出结果文件。

```

1  anxin@node38:~/pengling/k8s/helm-demo$ helm template my-nginx/ --output-dir
   ./result
2  wrote ./result/my-nginx/templates/serviceaccount.yaml
3  wrote ./result/my-nginx/templates/service.yaml
4  wrote ./result/my-nginx/templates/deployment.yaml
5
6  ---
7  # Source: my-nginx/templates/tests/test-connection.yaml
8  apiVersion: v1
9  kind: Pod
10 metadata:
11   name: "RELEASE-NAME-my-nginx-test-connection"
12   labels:
13     helm.sh/chart: my-nginx-0.1.0
14     app.kubernetes.io/name: my-nginx
15     app.kubernetes.io/instance: RELEASE-NAME
16     app.kubernetes.io/version: "1.16.0"
17     app.kubernetes.io/managed-by: Helm
18   annotations:
19     "helm.sh/hook": test-success
20 spec:
21   containers:
22   - name: wget
23     image: busybox
24     command: ['wget']
25     args: ['RELEASE-NAME-my-nginx:80']
26   restartPolicy: Never

```

渲染结果 (result/)

根据一些变量和判断, helm 直接帮我们生成了三种资源 (ServiceAccount, Service, Deployment) 的文件。

```
1 anxin@node38:~/pengling/k8s/helm-demo$ tree result
2 result
3 └─ my-nginx
4     └─ templates
5         ├── deployment.yaml
6         ├── serviceaccount.yaml
7         └─ service.yaml
8
9 2 directories, 3 files
```

deployment.yaml

```
1 anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
  nginx/templates/deployment.yaml
2 ---
3 # Source: my-nginx/templates/deployment.yaml
4 apiVersion: apps/v1
5 kind: Deployment
6 metadata:
7   name: RELEASE-NAME-my-nginx
8   labels:
9     helm.sh/chart: my-nginx-0.1.0
10    app.kubernetes.io/name: my-nginx
11    app.kubernetes.io/instance: RELEASE-NAME
12    app.kubernetes.io/version: "1.16.0"
13    app.kubernetes.io/managed-by: Helm
14 spec:
15   replicas: 1
16   selector:
17     matchLabels:
18       app.kubernetes.io/name: my-nginx
19       app.kubernetes.io/instance: RELEASE-NAME
20   template:
21     metadata:
22       labels:
23         app.kubernetes.io/name: my-nginx
24         app.kubernetes.io/instance: RELEASE-NAME
25     spec:
26       serviceAccountName: RELEASE-NAME-my-nginx
27       securityContext:
28         {}
29       containers:
30         - name: my-nginx
31           securityContext:
32             {}
33           image: "nginx:1.16.0"
34           imagePullPolicy: IfNotPresent
35           ports:
36             - name: http
37               containerPort: 80
38               protocol: TCP
```

```
39     livenessProbe:
40         httpGet:
41             path: /
42             port: http
43     readinessProbe:
44         httpGet:
45             path: /
46             port: http
47     resources:
48         {}
```

service.yaml

查看其中一个文件 `service.yaml`，还是非常完整的，基本可以满足需要了，再根据自己的需求改改就好了。

```
1  anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
   nginx/templates/service.yaml
2  ---
3  # Source: my-nginx/templates/service.yaml
4  apiVersion: v1
5  kind: Service
6  metadata:
7     name: RELEASE-NAME-my-nginx
8     labels:
9         helm.sh/chart: my-nginx-0.1.0
10        app.kubernetes.io/name: my-nginx
11        app.kubernetes.io/instance: RELEASE-NAME
12        app.kubernetes.io/version: "1.16.0"
13        app.kubernetes.io/managed-by: Helm
14  spec:
15     type: ClusterIP
16     ports:
17     - port: 80
18       targetPort: http
19       protocol: TCP
20       name: http
21     selector:
22         app.kubernetes.io/name: my-nginx
23         app.kubernetes.io/instance: RELEASE-NAME
```

serviceaccount.yaml

```
1 anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
  nginx/templates/serviceaccount.yaml
2 ---
3 # Source: my-nginx/templates/serviceaccount.yaml
4 apiVersion: v1
5 kind: ServiceAccount
6 metadata:
7   name: RELEASE-NAME-my-nginx
8   labels:
9     helm.sh/chart: my-nginx-0.1.0
10    app.kubernetes.io/name: my-nginx
11    app.kubernetes.io/instance: RELEASE-NAME
12    app.kubernetes.io/version: "1.16.0"
13    app.kubernetes.io/managed-by: Helm
```

添加模板文件

ConfigMap 模板

试着添加一个模板文件 `configmap.yaml` 到 `templates` 目录 (`my-nginx/templates/`)，内容如下：

```
1 anxin@node38:~/pengling/k8s/helm-demo/my-nginx/templates$ vi configmap.yaml
2
3 apiVersion: v1
4 kind: ConfigMap
5 metadata:
6   name: julin-file
7   namespace: default
8 data:
9   application.yaml: |-
10     server:
11       port: 8080
12     julin:
13       name: Julin
14       age: 18
15       website: www.julin.com
```

渲染结果

执行 `helm template my-nginx/ -output-dir ./result` 命令后渲染的结果如下：

```
1 anxin@node38:~/pengling/k8s/helm-demo$ tree result
2 result
3 └─ my-nginx
4   └─ templates
5       ├── configmap.yaml
6       ├── deployment.yaml
7       ├── serviceaccount.yaml
8       └─ service.yaml
9
10 2 directories, 4 files
```

结果中 `configmap.yaml` 与模板并没有什么不同，因为我们没有在模板文件里使用变量和判断语句等。

```
1 # ConfigMap 模板渲染结果
```



```
2 anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
  nginx/templates/configmap.yaml
3 ---
4 # Source: my-nginx/templates/configmap.yaml
5 apiVersion: v1
6 kind: ConfigMap
7 metadata:
8   name: julin-file
9   namespace: default
10 data:
11   application.yaml: |-
12     server:
13       port: 8080
14     julin:
15       name: julin
16       age: 18
17       website: www.julin.com
```

模板中使用变量

修改 ConfigMap 模板如下:

```
1 anxin@node38:~/pengling/k8s/helm-demo$ vi my-nginx/templates/configmap.yaml
2
3 apiVersion: v1
4 kind: ConfigMap
5 metadata:
6   name: julin-config-{{ .Values.environment }}
7   namespace: default
8 data:
9   application.yaml: |-
10     server:
11       port: {{ .Values.server.port }}
12     julin:
13       name: {{ .Values.julin.name }}
14       age: {{ .Values.julin.age }}
15       {{- if .Values.julin.website }}
16       website: {{ .Values.julin.webSite }}
17       {{- end }}
```

可以看见在模板中使用了许多双大括号的变量 `{{ .Values.xxx }}`, 同时需要在 `values.yaml` 文件中定义这些变量。

定义变量

在 `values.yaml` 文件中定义变量:

```
1 anxin@node38:~/pengling/k8s/helm-demo$ vi my-nginx/values.yaml
2
3 ... # 原 values.yaml 内容
4
5 # 自定义变量
6 environment: dev
7 server:
8   port: 80
9 julin:
10   name: Julin Peng
11   age: 28
12   webSite: https://www.julin.com
```

重新渲染

重新执行命令 `helm template my-nginx/ -output-dir ./result`, 渲染的结果如下:

```
1 anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
  nginx/templates/configmap.yaml
2 ---
3 # Source: my-nginx/templates/configmap.yaml
4 apiVersion: v1
5 kind: ConfigMap
6 metadata:
7   name: julin-config-dev
8   namespace: default
9 data:
10  application.yaml: |-
11    server:
12      port: 80
13    julin:
14      name: Julin Peng
15      age: 28
16      website: https://www.julin.com
```

不同环境 变量设置

多环境管理在 Helm Template 这也是非常简单的, 我们创建一个 `values-dev.yaml` 的变量文件, 内容如下:

```
1 anxin@node38:~/pengling/k8s/helm-demo/my-nginx$ vi values-dev.yaml
2
3 environment: dev
4 server:
5   port: 8080
6 julin:
7   name: Julin Peng
8   age: 1
```

通过以下命令来指定 dev 环境的变量文件。这样渲染出来的结果就是 dev 的相关配置了。其它环境同理。

```
1 # 指定 dev 环境的变量文件(file): -f my-nginx/values-dev.yaml
```

```

2  anxin@node38:~/pengling/k8s/helm-demo$ helm template my-nginx/ --output-dir
  ./result -f my-nginx/values-dev.yaml
3  wrote ./result/my-nginx/templates/serviceaccount.yaml
4  wrote ./result/my-nginx/templates/configmap.yaml
5  wrote ./result/my-nginx/templates/service.yaml
6  wrote ./result/my-nginx/templates/deployment.yaml
7
8  ---
9  # Source: my-nginx/templates/tests/test-connection.yaml
10 apiVersion: v1
11 kind: Pod
12 metadata:
13   name: "RELEASE-NAME-my-nginx-test-connection"
14   labels:
15     helm.sh/chart: my-nginx-0.1.0
16     app.kubernetes.io/name: my-nginx
17     app.kubernetes.io/instance: RELEASE-NAME
18     app.kubernetes.io/version: "1.16.0"
19     app.kubernetes.io/managed-by: Helm
20   annotations:
21     "helm.sh/hook": test-success
22 spec:
23   containers:
24   - name: wget
25     image: busybox
26     command: ['wget']
27     args: ['RELEASE-NAME-my-nginx:80']
28   restartPolicy: Never

```

通过命令行设置变量

使用 `-set` 或 `-set-string`，使用如下：

```

1  # 设置 julin.website=https://www.pengling.com (注意: website 大小写敏感。)
2  anxin@node38:~/pengling/k8s/helm-demo$ helm template my-nginx/ --output-dir
  ./result -f my-nginx/values-dev.yaml --set
  julin.website=https://www.pengling.com
3  wrote ./result/my-nginx/templates/serviceaccount.yaml
4  wrote ./result/my-nginx/templates/configmap.yaml
5  wrote ./result/my-nginx/templates/service.yaml
6  wrote ./result/my-nginx/templates/deployment.yaml
7  ...

```

查看输出结果：`website: https://www.pengling.com`。

```

1  anxin@node38:~/pengling/k8s/helm-demo$ cat result/my-
  nginx/templates/configmap.yaml
2  ---
3  # Source: my-nginx/templates/configmap.yaml
4  apiVersion: v1
5  kind: ConfigMap
6  metadata:
7   name: julin-config-dev
8   namespace: default
9  data:
10  application.yaml: |-

```

```
11 server:
12     port: 8080
13 julin:
14     name: Julin Peng
15     age: 1
16     website: https://www.pengling.com
```

Helm vs. kubebuilder 总结

通过以上 Helm 的介绍和实践，结合我司业务场景，可知：Helm 可用于具有依赖关系的应用管理（如 web 需要 webapi、萤石代理等服务的支持），但存在一个比较明显的问题：values 配置项较为繁琐，尤其是项目依赖较多时尤为突出，而且发行 Chart 后，如果需要修改配置，需要本地下载并修改配置，然后本地进行测试，应用管理极为不便。

Helm 更适用于对项目的快速安装体验，而非应用。

相比而言，kubebuilder 对于项目依赖及参数配置管理更趋向于应用的定制化（模板化），通过 CRD 的 yaml 文件配置项目应用即可。