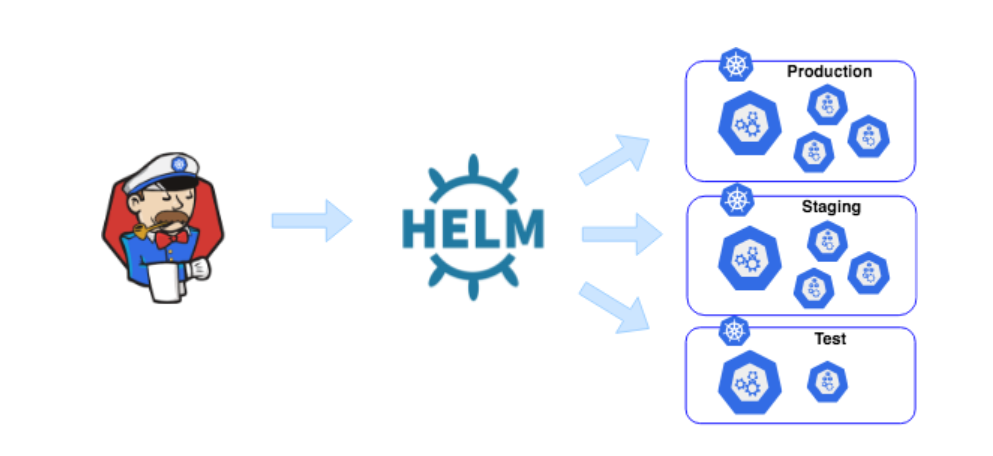
**HELM**

**Helm** là một package manager dành cho Kubernetes. Cũng giống như những package manager nổi tiếng của NodeJS như NPM hay YARN, Helm dùng để quản lý các Kubernetes application được dựng lên từ những Kubernetes resource. Hiểu nôm na là thay vì phải define một nùi Kubernetes resource để triển khai một ứng dụng, thì với Helm bạn chỉ việc gõ vài lệnh trong terminal và enter cái “bụp”, xong! . Helm là một trình quản lý gói ứng dụng cho Kubernetes, điều phối tải xuống, cài đặt và triển khai ứng dụng. Helm chart là cách để bạn xác định một ứng dụng bao gồm các tài nguyên nào trong Kubernetes.



Mô hình

Các khái niệm :

**Chart**: là tập hợp những file YAML template của những Kubernetes resource cần thiết để dựng lên một ứng dụng Kubernetes. Để dễ hình dung, Chart của Helm có thể được so sánh giống như một Image của Docker. Đương nhiên Helm cũng có Helm Hub nơi để tìm kiếm và chia sẻ Chart của những ứng dụng phổ biến.



* *Chart.yaml*: This is the main file that contains the description of our chart

Phần đầu tiên gồm thông số về API version mà chart này sẽ sử dụng. (bắt buôc). Tên của chart và mô tả của nó. Phần tiếp theo sẽ là loại chart (mặc định là application), tiếp theo là phiên bản chart mà ta sẽ triển khai và phiên bản ứng dụng này (appVersion).

* *values.yaml*: this is the file that contains the default values for our chart

Tiếp theo bạn đến với thông số **imagePullSecrets** – đây là cài đặt một khoá bí mật, mật khẩu hoặc API key để bạn có thể tải xuống image từ một Private Registy. Tiếp theo là **nameOverride** và **fullnameOverride**. 2 thông số này được hiểu là khi bạn chạy câu lệnh **heml create** thì đã đặt tên cho chart là **tel4vnchart** nhưng nếu bạn cần phải đổi tên chart sau khi bạn tạo nó thì bạn có thể thay đổi nó ở đây .

Trường**service accounts** cung cấp xác thực người dùng khi chạy một pod bên trong cluster. Nếu phần này Trống, nó sẽ được sinh tự động theo full name được chỉ định trong \_\_**helpers.tpl.** mình kiến nghị nên có service account được cài để ứng dụng có thể được liên kết với người dùng ngay trong chart.

Bạn có thể cài đặt pod security để giới hạn nhóm người hoặc người dùng nhất định được can thiệp vào pod. Hiểu được các thông số trong phần này sẽ giúp tăng khả năng bảo mật cho pod

Có 2 loại cài đặt khác nhau trong phần này có thể được chỉ định trong chart. Đầu tiên là sử dụng mạng local với **ClusterIP**. Khi đó ứng dụng sẽ được expose trên địa chỉ ip bên trong của Cluster. Với thông số này, ứng dụng của bạn chỉ có thể truy cập ở bên cluster (hoặc thông qua **ingress**, mặc định thông số **ingress** được đặt là **false**). Cài đặt khác là **NodePort**. Nó sẽ expose ứng dụng trên mỗi địa chỉ của từng node trong Kubernetes Cluster

Helm cho phép bạn chỉnh sửa thông số phần cứng (resources) sẽ cung cấp để chạy ứng dụng. Bạn có thể cấu hình giá trị lớn nhất và nhỏ nhất phần cứng sẽ cung cấp. Trong bài viết này, bạn sẽ giới hạn lại như bên dưới

**nodeSelector**: chỉ định rằng bạn sẽ triển khai ứng dụng trên các node nào trong cluster.

**taining**: chỉ định thông số để loại bỏ một node ra khỏi quá trình triển khai

**affinity:**chỉ định thông số để thêm ưu tiên chọn một node khi triển khai

* *templates*: This is the directory where Kubernetes resources are defined as templates

Phần quan trọng nhất của chart là thư mực template. Trong đó gồm tất cả các cài đặt mà ứng dụng của bạn sẽ triển khai trong cluster. Như bạn thấy ở bên dưới, ứng dụng này bao gồm các phần triển khai cơ bản nhất là ingress, service account, và service. Ngoài ra còn có thư mục test, bao gồm các kiểm tra dùng để check kết nối vào bên trong ứng dụng.

Một số giá trị trong template :

* indent là cách vào vd như muốn cách vào 5 ô thì indent 5
* nindent là cách vào và xuống dòng
* include là bao gồm một cái biến sẽ được định nghĩa trong \_helpers.tpl
* Values là giá trị đc lấy từ file values
* *charts*: This is an optional directory that may contain sub-charts

Trong đó bao gồm các chart khác mà ứng dụng của bạn cần để triển khai. Một số chart cần thêm chart khác để có thể bổ sung được thì các chart bổ sung đó sẽ được để vào đây

Mình sẽ có 2 cách , một là dung file chart.yml hoặc là bỏ cái dependency vào trong folder charts

# Chart.yaml option 1

dependencies:

- name: nginx

version: "1.2.3"

repository: "https://example.com/charts"

- name: memcached

version: "3.2.1"

repository: <https://another.example.com/charts>

# Chart.yaml option 2

dependencies:

- name: nginx

version: "1.2.3"

repository: "file://../dependency\_chart/nginx"

* *.helmignore*: This is where we can define patterns to ignore when packaging (similar in concept to .gitignore)

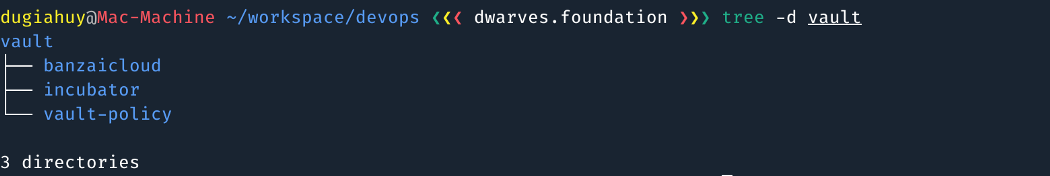
**Config**: nằm trong file values.yaml chứa những configuration dành riêng cho một bản release của Kubernetes application. Đó có thể là config cho service, ingress, deployment,… cho đến những application riêng biệt như Kafka, Consul, Vault, NATS-streaming,…



**Release**: là một version của K8s application đang chạy dựa trên Chart và kết hợp với một Config cụ thể.



**Repositories**: Helm Charts có thể được publish thông qua nhiều repo khác nhau. Nó có thể là những private repo chỉ dùng trong nội bộ công ty, hoặc public thông qua Helm Hub. Một số Chart có thể có nhiều phiên bản của từng công ty hoặc publisher khác nhau. Riêng những Chart trong repo Stable thì luôn phải đáp ứng được tiêu chí từ Technical Requirements của Helm.



**Tại sao Helm lại quan trọng đối với việc triển khai Kubernetes?**

-Each deployment of a chart is a Helm release. Helm automatically maintains a versioned history of your releases. If something goes wrong with a deployment, getting back to your previous state can be as simple as helm rollback RELEASE\_NAME.

-Helm also provides several CI/CD pipeline integration hooks, so you can configure actions to occur, for example, before installation begins, or after an upgrade has finished. You can even configure health checks for Helm to run and verify a deployment has completed successfully.

-Another distinguishing feature is the ability to provide application configuration during deployment. Not only can you specify the Kubernetes resources (deployments, services, etc.) that make up your application, but also environment-specific configuration for those resources. This allows the same Helm chart to be used across all of your environments. For example, when deploying to production, a high number of replicas and persistent storage may be required. Conversely, a developer working on their laptop may only need a single instance of the app and no storage.

*# Production*

helm install demo-app

*# Development*

helm install --set replicas=1 --set persistence.enabled=false demo-app

In the production example, Helm will set the application's configuration from the values.yaml that is part of every Helm chart. This file would likely be maintained by your DevOps team, and versioned with your SCM. The developer is able to use the exact same deployment process in their local environment, overriding configuration as needed with a few CLI parameters.

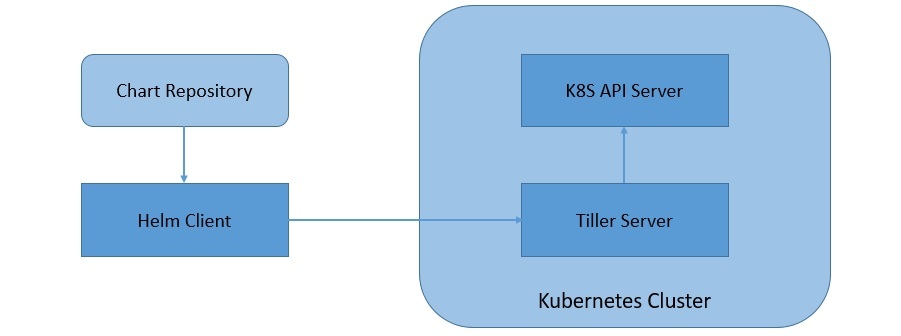
**Install helm :**

<https://helm.sh/docs/intro/install/>

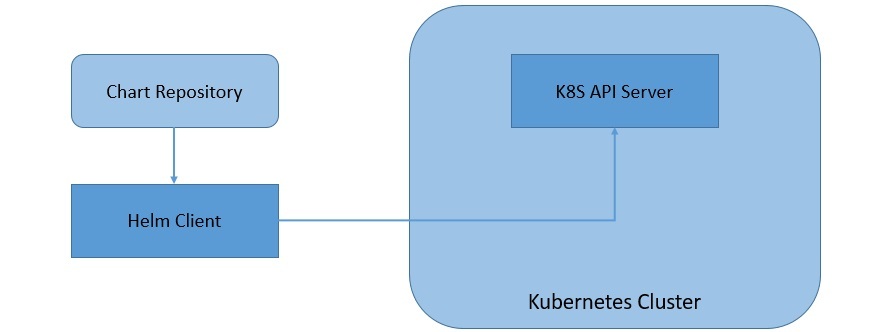
**Helm Architecture**

Helm has undergone a significant architecture uplift as part of Helm 3. It has some of the significant and long-awaited changes as compared to Helm 2. Apart from packing a new set of capabilities, Helm 3 also features changes in its internal plumbing. We'll examine some of these changes.

Helm 2 ( we will run helm init cmd to connect tiller server ):



Helm 3 ( don’t need run cmd ) :



**Step by step :**

1.create helm chart

*#first , go to your directory where you want to create chart*

*helm create <chart-name>*

2.create helm templates

*#templates will be auto generate when you create helm chart*

3.providing a values ( values.yaml )

4. Helm Lint ( this is a simple command that takes the path to a chart and runs a battery of tests to ensure that the chart is well-formed )

*helm lint ./<chart-name>*

5. Helm Templates ( check templates )

*helm template ./<chart-name>*

6. Helm Install

*helm install <name> ./<chart-name> --version <version>*

*helm install <name> <chart-name>/ --values <chart-name >/values.yaml*

7. Helm Get ( or check log kuber if helm installed but pod don’t up )

*helm ls –all or kubectl get events -w*

8. Helm Upgrade ( you can create new file with new values then use -f to direct to new file values )

*helm upgrade <name> ./<chart-name> --version <version>*

9.Helm RollBack

*helm rollback <chart-name> 1*

10.Helm Uninstall

*helm uninstall <name>*

11.Helm Package

*helm package ./ <chart-name>*

12.Helm Repo

*#create file index.yaml*

*helm repo index my-repo/ --url https://<username>.github.io/my-repo*

*#add repo*

*helm repo add my-repo https://my-pages.github.io/my-repo*

13.Helm Search

*helm search repo <KEYWORD>*

Repo public :

<https://artifacthub.io/>

<https://charts.bitnami.com/bitnami>

<https://www.freecodecamp.org/news/helm-charts-tutorial-the-kubernetes-package-manager-explained/>

<https://www.youtube.com/watch?v=LWW8_S0dbG8>

**Flow Control :**

Helm's template language provides the following control structures:

* if/else for creating conditional blocks

The basic structure for a conditional looks like this:

{{ if PIPELINE }}

# Do something

{{ else if OTHER PIPELINE }}

# Do something else

{{ else }}

# Default case

{{ end }}

* with to specify a scope
* range, which provides a "for each"-style loop

In addition to these, it provides a few actions for declaring and using named template segments:

* define declares a new named template inside of your template
* template imports a named template
* block declares a special kind of fillable template area

<https://helm.sh/docs/chart_template_guide/control_structures/>