Helm và các vấn đề mà nó giải quyết

Thông thường, với một hệ thống microservices lớn, mỗi dịch vụ đều cần nhiều manifest files riêng biệt như Deployment, Service, ConfigMap,... Điều này gây khó khăn trong việc duy trì và tái sử dụng. Ngoài ra, việc triển khai hay gỡ bỏ các file này đều cần thao tác thủ công qua các lệnh kubectl.

Helm giải quyết vấn đề này bằng cách cung cấp một định dạng đóng gói gọi là **Chart**. Mỗi **Helm chart** là một tập hợp các file mô tả các tài nguyên Kubernetes có liên quan. Các manifest files của nhiều dịch vụ có thể được gom lại trong cùng một chart, giúp dễ dàng triển khai toàn bộ hệ thống chỉ với một lệnh duy nhất.

Thêm vào đó, Helm hỗ trợ **các chart phụ thuộc (child/dependent charts)** tương tự như cấu trúc kế thừa trong lập trình hướng đối tượng. Điều này cho phép người dùng tích hợp các thành phần bên thứ ba (như cơ sở dữ liệu, Redis,...) vào hệ thống một cách linh hoạt và dễ bảo trì.

Một điểm nổi bật khác là Helm sử dụng các file **template YAML** để trừu tượng hóa các phần cấu hình có tính chất lặp lại. Các giá trị động (ví dụ: tên dịch vụ, cổng, label) được tách riêng trong file values.yaml. Điều này giúp việc tạo ra các file manifest cụ thể cho từng dịch vụ trở nên tự động và nhất quán hơn, đồng thời giảm thiểu sự trùng lặp không cần thiết.

Hướng dẫn cài đặt Helm

Helm là một công cụ quản lý package quan trọng trong hệ sinh thái Kubernetes, hỗ trợ triển khai và quản lý các ứng dụng Kubernetes một cách hiệu quả. Trong bài học này, người hướng dẫn đã trình bày chi tiết các bước để cài đặt Helm trên nhiều hệ điều hành khác nhau.

1. Điều kiện tiên quyết

Trước khi cài đặt Helm, người dùng cần đảm bảo:

- Đã có một cụm Kubernetes đang hoạt động trên máy cục bộ hoặc trên nền tảng điện toán đám mây.
- Cấu hình các yếu tố bảo mật nếu cần thiết (không bắt buộc).
- Cài đặt Helm trên hệ điều hành tương ứng.

2. Cài đặt Helm trên macOS

- Sử dụng Homebrew – trình quản lý package phổ biến trên macOS.
- Chạy lệnh: brew install helm.
- Sau khi cài đặt xong, kiểm tra phiên bản bằng lệnh helm version.

3. Cài đặt Helm trên Windows

- Trước tiên, cài đặt Chocolatey – trình quản lý package cho Windows.
    - Truy cập trang chủ chocolatey.org, chọn mục "Install", và làm theo hướng dẫn dành cho người dùng cá nhân.
    - Mở PowerShell với quyền Administrator.
    - Kiểm tra và thay đổi chính sách thực thi nếu bị hạn chế bằng lệnh:
        - Get-ExecutionPolicy
        - Nếu kết quả là Restricted, chạy: Set-ExecutionPolicy AllSigned
    - Chạy đoạn lệnh được cung cấp trên trang để cài đặt Chocolatey.
    - Kiểm tra cài đặt bằng lệnh: choco hoặc choco -?.
- Sau khi đã có Chocolatey, chạy lệnh sau để cài đặt Helm:
    - choco install kubernetes-helm

- Xác minh bằng lệnh: helm version.

Cài đặt và Hiểu Cấu Trúc Helm Chart

Trong quá trình triển khai các ứng dụng microservices trên Kubernetes, Helm là một công cụ quan trọng giúp quản lý và đóng gói các tài nguyên dưới dạng "Helm Chart". Cài đặt một Helm chart bên thứ ba – cụ thể là Helm chart của **WordPress** – từ kho lưu trữ Bitnami bằng lệnh:

helm repo add bitnami https://charts.bitnami.com/bitnami

helm install wordpress bitnami/wordpress

Lệnh này sẽ tạo ra một release mới tên là wordpress dựa trên chart có sẵn. Chúng em cũng đã kiểm tra quá trình triển khai thành công bằng các lệnh kubectl get pods, kubectl get svc, và sử dụng minikube service để truy cập vào giao diện web của WordPress chạy trên Kubernetes.

Tiếp theo, trong bài học **004**, chúng em tìm hiểu sâu hơn về **cấu trúc của một Helm chart**. Mỗi chart khi được cài đặt hoặc tạo ra đều tuân theo một cấu trúc thư mục chuẩn, bao gồm:

- **Chart.yaml**: chứa thông tin metadata của chart như tên, version, API version, dependencies, maintainer, source v.v...
- **values.yaml**: chứa các giá trị cấu hình động sẽ được sử dụng để render vào các file template Kubernetes.
- **Thư mục charts/**: chứa các Helm chart phụ mà chart chính phụ thuộc vào (ví dụ như mariadb, memcached, common...).

- **Thư mục templates/**: chứa các file template YAML cho các tài nguyên Kubernetes như deployment.yaml, service.yaml, v.v... Các file này sử dụng cú pháp template của Helm để tham chiếu đến giá trị trong values.yaml.

Thông qua ví dụ về chart WordPress, chúng ta đã thấy rõ mối liên hệ giữa values.yaml và các template – giá trị từ values.yaml sẽ được Helm render vào các template YAML để tạo ra các manifest hoàn chỉnh, sẵn sàng apply vào Kubernetes cluster.

Ngoài ra, lưu ý rằng khi làm việc với microservices riêng (như hệ thống **Easy Bank**), sẽ không có sẵn Helm chart từ bên thứ ba. Do đó, bước tiếp theo trong lộ trình là **tự tạo Helm chart** cho từng microservice theo đúng cấu trúc và nhu cầu triển khai thực tế. Việc này giúp dễ dàng tái sử dụng, quản lý cấu hình cho nhiều môi trường (dev, staging, production) bằng cách chỉ cần thay đổi values.yaml.

Tạo và sử dụng Helm Chart trong triển khai microservices trên Kubernetes

Tạo Helm Chart Mới

Đầu tiên, chúng ta tạo một thư mục mới để chứa các Helm Charts của dự án. Sau khi tạo thư mục section_16 và Helm, ta sử dụng lệnh helm create để tạo một Helm Chart mới có tên easybank-common. Chart này sẽ được sử dụng làm cơ sở để triển khai các microservices chung cho toàn bộ hệ thống.

helm create easybank-common

Kết quả là hệ thống tự động tạo ra một Helm Chart với cấu trúc mặc định, bao gồm các tệp cấu hình cho một ứng dụng web, ví dụ như tệp values.yaml, deployment.yaml, và các tệp khác. Tuy nhiên, chúng ta không sử dụng các tệp này vì chúng được tạo ra để triển khai một ứng dụng web (nginx).

Tùy Chỉnh và Cấu Hình Các Tệp

Sau khi tạo Helm Chart, chúng ta thực hiện một số thay đổi:

- **Xóa các tệp mẫu** trong thư mục templates vì chúng không cần thiết cho microservices của chúng ta.
- **Chỉnh sửa tệp values.yaml** để loại bỏ các giá trị mặc định liên quan đến nginx và thay thế bằng các thông số cần thiết cho microservices của hệ thống.
- **Cập nhật chart.yaml** với thông tin của Helm Chart, bao gồm tên, mô tả, và phiên bản ứng dụng.

Tạo Các Tệp Template Cần Thiết

Chúng ta đã tạo ba tệp template chính cho việc triển khai microservices:

- service.yaml: Định nghĩa các thông số cho dịch vụ Kubernetes.
- deployment.yaml: Định nghĩa các thông số cho việc triển khai các pod.
- configmap.yaml: Định nghĩa các cấu hình cần thiết cho microservices.

Các tệp này sử dụng cú pháp của Helm để tạo các cấu hình động, ví dụ như tên dịch vụ, cổng, và tên hình ảnh container, được lấy từ tệp values.yaml.

Sử Dụng Helm Template

Trong các tệp YAML này, chúng ta sử dụng cú pháp Helm template (Go template) để tạo ra các giá trị động. Ví dụ, trong service.yaml, tên dịch vụ và các cổng được lấy từ các giá trị được định nghĩa trong tệp values.yaml. Tương tự, trong deployment.yaml, các thông số như số lượng bản sao của pod và tên hình ảnh được cung cấp qua các khóa trong values.yaml.

Lợi Ích Của Việc Sử Dụng Helm Chart

Việc sử dụng Helm Chart giúp đơn giản hóa quá trình triển khai, quản lý và cập nhật microservices trên Kubernetes. Helm Chart cho phép tái sử dụng cấu hình cho nhiều microservices, giúp triển khai toàn bộ hệ thống chỉ với một lệnh đơn giản. Điều này không chỉ tiết kiệm thời gian mà còn giảm thiểu các sai sót trong quá trình cấu hình.

Demo của Lệnh helm template

Lệnh helm template là một công cụ hữu ích giúp bạn xem tất cả các Kubernetes manifest files mà Helm sẽ tạo ra từ chart của bạn, giúp bạn kiểm tra và xác nhận lại trước khi thực sự triển khai chúng lên Kubernetes.

Các bước thực hiện:

1. **Mở Terminal trong thư mục Helm Chart:** Đầu tiên, bạn cần mở terminal trong thư mục của Helm chart của mình. Sau đó, sử dụng lệnh: *helm template* .

2. **Xử lý Lỗi Trong Template:** Trong quá trình thực hiện lệnh, chúng tôi gặp lỗi do một biến trong file service.yaml bị sai, nơi chúng tôi đã vô tình nhập sai tên biến. Sau khi sửa lỗi này, lệnh helm template đã chạy thành công và hiển thị các Kubernetes manifest files.
3. **Tái biên dịch các Helm Charts:** Sau khi sửa lỗi, chúng tôi chạy lại lệnh helm dependencies build để tái biên dịch các chart có phụ thuộc vào easybank-common. Điều này cũng giúp tái biên dịch các Helm charts cho các môi trường khác như QA, prod.
4. **Xem Kết Quả Kubernetes Manifests:** Lệnh helm template cho phép chúng ta kiểm tra tất cả các Kubernetes manifest files, như deployment.yaml, service.yaml, v.v., mà Helm sẽ tạo ra cho microservices của chúng ta.
5. **Kiểm Tra Từng Môi Trường:** Sau khi chạy lệnh helm template cho môi trường sản xuất (prod), chúng ta cũng có thể chạy lệnh này cho các môi trường

khác (ví dụ: dev, qa) để kiểm tra cấu hình manifest của từng môi trường, với các thay đổi phù hợp.

6. **Tiếp Tục với Việc Triển Khai:** Sau khi xác nhận tất cả các Kubernetes manifest files, bước tiếp theo sẽ là triển khai các microservices lên Kubernetes bằng cách sử dụng Helm charts cho các môi trường cụ thể. Tuy nhiên, trước khi triển khai, bạn cần đảm bảo rằng các dịch vụ như **Keycloak**, **Kafka**, và **Grafana** đã được cài đặt và cấu hình trong Kubernetes cluster, vì đây là các thành phần cần thiết.

Triển khai Các Dịch vụ vào Kubernetes Cluster bằng Helm Chart

Cài đặt Keycloak trong Kubernetes Cluster sử dụng Helm Chart

Keycloak là một giải pháp mã nguồn mở để quản lý quyền truy cập và xác thực người dùng trong các ứng dụng web. Để triển khai Keycloak trên Kubernetes bằng Helm, bạn có thể làm theo các bước sau:

- **Bước 1:** Cài đặt Helm (nếu chưa cài đặt):

brew install helmbrew install helm

- **Bước 2:** Thêm kho Helm cho Keycloak:

helm repo add codecentric https://codecentric.github.io/helm-charts

helm repo update

- **Bước 3:** Cài đặt Keycloak:

helm install keycloak codecentric/keycloak --set keycloak.username=admin --set keycloak.password=admin123

Bước 4: Kiểm tra tình trạng của Keycloak:

kubectl get pods

Bước 5: Truy cập vào Keycloak qua URL được cung cấp và đăng nhập bằng thông tin quản trị viên (username: admin, password: admin123).

Cài đặt Kafka trong Kubernetes Cluster sử dụng Helm Chart

Kafka là một hệ thống phân tán giúp xử lý luồng dữ liệu lớn trong thời gian thực. Để triển khai Kafka trên Kubernetes, bạn có thể sử dụng Helm Charts như sau:

- **Bước 1:** Thêm kho Helm cho Kafka:

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

- **Bước 2:** Cài đặt Kafka với các tham số cơ bản:

helm install kafka bitnami/kafka --set persistence.enabled=true --set replicaCount=3

- **Bước 3:** Kiểm tra các pods của Kafka:

kubectl get pods

- **Bước 4:** Kết nối ứng dụng với Kafka thông qua Kafka Service URL (có thể lấy từ các dịch vụ trong Kubernetes).

Cài đặt MySQL trong Kubernetes Cluster sử dụng Helm Chart

MySQL là một cơ sở dữ liệu quan trọng trong hệ thống. Để triển khai MySQL vào Kubernetes với Helm, bạn thực hiện như sau:

- **Bước 1:** Thêm kho Helm cho MySQL:

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

- **Bước 2:** Cài đặt MySQL với cấu hình mật khẩu root:

helm install mysql bitnami/mysql --set auth.rootPassword=my-secret-pw

- **Bước 3:** Kiểm tra MySQL pods:

kubectl get pods

- **Bước 4:** Truy cập MySQL thông qua kubectl port-forward:

kubectl port-forward svc/mysql 3306:3306

- **Bước 5:** Kết nối MySQL thông qua client:

mysql -h 127.0.0.1 -u root -p

Cài đặt Redis trong Kubernetes Cluster sử dụng Helm Chart

Redis là một hệ thống lưu trữ dữ liệu key-value nhanh chóng, thường được sử dụng cho caching và quản lý session. Để triển khai Redis trên Kubernetes, bạn có thể làm theo các bước sau:

- **Bước 1:** Thêm kho Helm cho Redis:

helm repo add bitnami https://charts.bitnami.com/bitnami

helm repo update

- **Bước 2:** Cài đặt Redis với các cấu hình mặc định:

helm install redis bitnami/redis --set auth.password=my-redis-password

- **Bước 3:** Kiểm tra Redis pods:

kubectl get pods

- **Bước 4:** Truy cập Redis thông qua kubectl port-forward:

kubectl port-forward svc/redis-master 6379:6379

- **Bước 5:** Kết nối với Redis bằng Redis CLI:

redis-cli -h 127.0.0.1 -p 6379 -a my-redis-password

Quản lý và Cấu hình với Helm Charts

- **Quản lý các dịch vụ Kubernetes:** Sau khi triển khai, bạn có thể dễ dàng nâng cấp, thay đổi cấu hình hoặc gỡ bỏ các dịch vụ. Ví dụ:
    - **Nâng cấp dịch vụ:**

helm upgrade kafka bitnami/kafka --set replicaCount=5

    - **Gỡ bỏ dịch vụ:**

helm uninstall kafka

- **Quản lý với values.yaml:** Bạn có thể sử dụng file values.yaml để cấu hình các tham số cho từng dịch vụ. Ví dụ cấu hình cho Keycloak trong values.yaml:

keycloak:

username: admin

password: admin123

database:

provider: mysql

url: jdbc:mysql://mysql:3306/keycloak

Lệnh helm upgrade để cập nhật dịch vụ trong Kubernetes

Trong quá trình triển khai các microservices trên Kubernetes, sau khi đã cài đặt các dịch vụ bằng Helm charts, đôi khi chúng ta cần phải cập nhật dịch vụ để triển khai

các thay đổi mới, ví dụ như cập nhật phiên bản Docker image, thay đổi số lượng replicas, hoặc thay đổi các cấu hình khác của dịch vụ. Lệnh helm upgrade là công cụ hữu ích để thực hiện những thay đổi này mà không cần phải triển khai lại toàn bộ dịch vụ.

Các bước thực hiện cập nhật dịch vụ:

1. **Cập nhật cấu hình**:
    - Đầu tiên, bạn cần mở file values.yaml trong Helm chart của dịch vụ muốn cập nhật. Ví dụ, nếu muốn thay đổi tag của Docker image cho dịch vụ Gateway server, bạn chỉ cần cập nhật giá trị tag từ s14 sang tag mới (ví dụ: s12 để loại bỏ các yêu cầu xác thực).
    - Sau khi cập nhật xong, lưu lại file values.yaml.
2. **Xây dựng lại các dependencies của Helm**:
    - Sau khi thay đổi cấu hình, bạn cần xây dựng lại các dependencies của Helm để đảm bảo các thay đổi được áp dụng cho tất cả các thành phần liên quan. Để làm điều này, bạn chạy lệnh:

helm dependencies build

3. **Chạy lệnh helm upgrade**:
    - Khi đã xây dựng lại các dependencies, bạn sử dụng lệnh helm upgrade để áp dụng các thay đổi lên Kubernetes cluster. Lệnh sẽ được thực hiện như sau:

helm upgrade easybank prod-env

- Lệnh này sẽ cập nhật release có tên là easybank với chart prod-env. Helm sẽ tự động so sánh sự khác biệt giữa phiên bản hiện tại và chart mới, chỉ triển khai những thay đổi cần thiết.

4. **Xác minh việc cập nhật**:
   - Sau khi chạy lệnh helm upgrade, bạn có thể kiểm tra tiến trình cập nhật bằng cách truy cập vào Kubernetes dashboard, vào phần Pods và tìm kiếm các pod đã được cập nhật (ví dụ: pod Gateway Server).
   - Kiểm tra logs của pod để chắc chắn rằng dịch vụ đã khởi động thành công.
5. **Kiểm tra thay đổi**:
   - Sau khi dịch vụ được triển khai thành công, bạn có thể kiểm tra các thay đổi bằng cách sử dụng các công cụ kiểm tra API như Postman. Ví dụ, nếu bạn đã cấu hình để loại bỏ yêu cầu xác thực, bạn có thể thử gọi API mà không cần thông tin xác thực. Nếu cấu hình đúng, bạn sẽ nhận được phản hồi thành công.
6. **Sửa lỗi**:
   - Trong trường hợp bạn thay đổi sai tag hoặc cấu hình, bạn có thể dễ dàng sửa lại và thực hiện lại lệnh helm upgrade. Ví dụ, nếu bạn đã thay đổi tag thành s12 thay vì s11, bạn chỉ cần cập nhật lại và chạy lại lệnh:

helm upgrade easybank prod-env

Lệnh helm history

Lệnh helm history cho phép chúng ta xem lịch sử các lần cài đặt và nâng cấp (upgrade) của một release Helm. Mỗi lần bạn thực hiện một nâng cấp, Helm sẽ tạo ra một revision (phiên bản) mới cho dịch vụ đó.

Ví dụ, khi sử dụng lệnh helm history easybank, chúng ta có thể thấy lịch sử các thay đổi của dịch vụ Easy Bank. Lịch sử này sẽ bao gồm thông tin về các revision, bao gồm thông tin về những lần nâng cấp, ví dụ như:

- Revision 1: Cài đặt ban đầu.
- Revision 2: Nâng cấp với tag Docker mới.
- Revision 3: Cập nhật lại với tag Docker khác (s11).

Lệnh helm history giúp chúng ta dễ dàng theo dõi các thay đổi và nhận diện các phiên bản cụ thể.

2. Lệnh helm rollback

Lệnh helm rollback cho phép chúng ta quay lại một phiên bản trước đó của dịch vụ trong Helm, rất hữu ích khi một phiên bản mới không hoạt động như mong đợi hoặc có lỗi. Lệnh này sẽ phục hồi toàn bộ trạng thái của các dịch vụ trong cluster về phiên bản trước đó.

Ví dụ:

- Nếu bạn muốn quay lại revision 1 của easybank, bạn sẽ sử dụng lệnh:

helm rollback easybank 1

- Lệnh này sẽ quay lại dịch vụ easybank ở phiên bản revision 1 mà không cần phải thay đổi từng dịch vụ riêng biệt trong Kubernetes.

Sau khi chạy lệnh helm rollback, dịch vụ sẽ tự động được phục hồi, và chúng ta có thể kiểm tra lại trạng thái dịch vụ trong Kubernetes Dashboard. Ví dụ, khi quay lại

revision 1, Gateway server sẽ được bảo mật đúng cách như phiên bản ban đầu, và việc truy cập API sẽ trả về mã lỗi 401 nếu yêu cầu không có xác thực đúng.

3. Kiểm tra lại sau khi rollback

Sau khi rollback thành công, bạn có thể sử dụng lệnh helm history để kiểm tra lại lịch sử các phiên bản. Phiên bản mới nhất sẽ ghi nhận rằng lệnh rollback đã được thực hiện, ví dụ:

- Revision 4: Quay lại revision 1.

Lệnh helm uninstall để Gỡ Bỏ Các Microservice

Các Bước Thực Hiện:

1. **Liệt Kê Các Cài Đặt Đã Thực Hiện:** Trước khi gỡ bỏ các microservices, chúng ta cần kiểm tra tất cả các cài đặt Helm đã thực hiện trên cluster. Để làm điều này, sử dụng lệnh sau:

helm ls

Lệnh này sẽ liệt kê tất cả các releases đã được cài đặt trong Kubernetes cluster.

2. **Gỡ Bỏ Microservice:** Sau khi liệt kê được các ứng dụng đã cài đặt, ta có thể sử dụng lệnh helm uninstall để gỡ bỏ từng ứng dụng. Ví dụ, để gỡ bỏ ứng dụng "Easy Bank", ta thực hiện lệnh sau:

helm uninstall easy-bank

Giải thích:

- Lệnh trên sẽ xóa tất cả các tài nguyên liên quan đến release "easy-bank" như pods, deployments, services, config maps, secrets, v.v., trong Kubernetes.

3. **Gỡ Bỏ Các Ứng Dụng Khác:** Tiếp theo, ta sẽ gỡ bỏ các ứng dụng khác trong cluster như Grafana, Tempo, Loki, Prometheus, Kafka và Keycloak. Mỗi ứng dụng sẽ được gỡ bỏ bằng lệnh tương tự:

helm uninstall grafana

helm uninstall tempo

helm uninstall loki

helm uninstall prometheus

helm uninstall kafka

helm uninstall keycloak

4. **Kiểm Tra Sau Khi Gỡ Bỏ:** Sau khi thực hiện các lệnh gỡ bỏ, ta có thể kiểm tra lại các release đã bị xóa bằng lệnh:

helm ls

Kết quả mong đợi: Khi tất cả các dịch vụ đã bị gỡ bỏ, lệnh helm ls sẽ trả về kết quả trống, nghĩa là không còn release nào trong cluster.

5. **Kiểm Tra Trên Kubernetes Dashboard:** Để xác nhận rằng các dịch vụ đã bị gỡ bỏ hoàn toàn, bạn có thể truy cập Kubernetes Dashboard. Sau khi gỡ bỏ các ứng dụng, các tài nguyên như Pods, Deployments, Services sẽ không còn hiển thị. Tuy nhiên, các Persistent Volume Claims (PVCs) vẫn có thể tồn tại và cần được xóa thủ công để tránh lỗi khi triển khai lại các ứng dụng.

6. **Xóa Persistent Volume Claims:** Khi gỡ bỏ các microservices, PVC có thể không bị xóa tự động. Để tránh vấn đề khi triển khai lại các dịch vụ trong tương lai, bạn cần xóa các PVC thủ công. Bạn có thể xóa chúng qua giao diện Kubernetes Dashboard hoặc sử dụng lệnh kubectl:

kubectl delete pvc --all

Lưu ý: Xóa PVC sẽ đảm bảo rằng không có dữ liệu tồn đọng khi cài đặt lại các dịch vụ sau này.

Quick Revision of Important Helm Commands

1. **Helm create**

   Lệnh này giúp bạn tạo một chart Helm mới, với các thư mục mặc định như charts.yaml, values.yaml, charts và templates. Bạn có thể tạo một chart trống với tên do bạn chỉ định.

2. **helm dependencies build**

   Sử dụng lệnh này để xây dựng các phụ thuộc của một chart, nếu chart đó có các phụ thuộc. Các chart phụ thuộc sẽ được sao chép vào thư mục charts.

3. **helm install**

   Để triển khai một release, bạn sử dụng lệnh helm install. Cung cấp tên release và tên chart hoặc thư mục chứa chart. Helm sẽ cài đặt tất cả các hướng dẫn triển khai được định nghĩa trong chart.

4. **helm upgrade**

   Khi có thay đổi mới trong chart và bạn muốn cập nhật các microservices đã triển khai, bạn sử dụng helm upgrade để cập nhật release.

5. **helm history**

   Lệnh này cho phép bạn kiểm tra lịch sử các release đã được triển khai. Nó sẽ liệt kê tất cả các phiên bản trước của một release.

6. **helm rollback**

   Để quay lại một phiên bản trước của release, bạn sử dụng helm rollback và

chỉ định tên release cùng với số revision muốn quay lại. Nếu không chỉ định revision, Helm sẽ quay lại phiên bản trước ngay lập tức.

7. **helm uninstall**

   Để gỡ bỏ một release và xóa các microservices khỏi Kubernetes cluster, bạn dùng lệnh helm uninstall và chỉ định tên release bạn muốn gỡ bỏ.

8. **helm template**

   Lệnh này giúp bạn tạo ra các manifest Kubernetes mà Helm sẽ sử dụng khi cài đặt. Điều này giúp bạn kiểm tra và gỡ lỗi trước khi cài đặt chính thức.

9. **helm ls**

   Lệnh này sẽ liệt kê tất cả các release đã được cài đặt trong hệ thống Helm.