Helm and Kustomize

1. Helm

1.1 Helm 이란?

- https://helm.sh
- 애플리케이션을 쉽게 배포하고 관리하기 위한 쿠버네티스 패키지 매니저
- 애플리케이션의 설치, 업그레이드, 설정 관리를 단순화
- 패키지를 Kubernetes 리소스 정의 파일과 설정 정보가 포함된 "차트(Chart)"라는 형태로 관리

- **Helm의 주요 기능**
  - **애플리케이션 패키징**: Kubernetes 리소스들을 하나의 패키지(차트)로 묶어 관리
  - **애플리케이션 배포**: 차트를 사용해 Kubernetes 클러스터에 애플리케이션을 쉽게 배포
  - **버전 관리**: 애플리케이션의 다양한 버전을 관리하고, 필요 시 특정 버전으로 롤백
  - **설정 관리**: 애플리케이션의 설정 값을 쉽게 커스터마이징하여 배포
  - **업그레이드 및 롤백**: 배포된 애플리케이션을 손쉽게 업그레이드하거나 이전 상태로 롤백 가능
  - **차트 저장소 관리**: 공용 또는 사설 차트 저장소를 통해 차트를 공유하고 관리

- **Helm 설치**(https://helm.sh/docs/intro/install/)
  - From Script

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-
helm-3 chmod 700 get_helm.sh ./get_helm.sh
```

  - 명령어 도움말 보기

```
helm --help helm [COMMAND] --help # 자동완성 등록하기 echo
"source <(helm completion bash)" >> ~/.bashrc source
~/.bashrc
```

1.2 Helm 리포지토리 등록하기

- 리포지토리는 여러 차트들의 저장된 저장소로, 집합

- 대표적인 리포지토리
  - bitnami: https://bitnami.com/stacks/helm
    - 가장 인기 있고 널리 사용되는 Helm 리포지토리
    - 다양한 오픈소스 애플리케이션의 Helm 차트를 제공하며, 설치 및 사용이 간편하도록 미리 구성된 차트들이 포함
    - 유지보수가 잘 되어 있으며, 차트가 지속적으로 업데이트되고 있음
  - Artifact Hub : https://artifacthub.io/packages/search
    - Artifact Hub는 CNCF(Cloud Native Computing Foundation) 에서 호스팅하는 Public 저장소
    - 다양한 Helm 차트를 한 곳에서 검색할 수 있는 중앙 허브 역할
    - 여러 조직과 개인이 자신의 리포지토리를 등록할 수 있으며, 다양한 출처의 차트를 검색하고 설치할 수 있음
    - 개별 리포지토리를 직접 등록할 필요 없이, Artifact Hub 웹사이트를 통해 필요한 차트를 검색하고 사용
  - Repository 추가/삭제 명령어

명령어	설명	예시
helm search repo	리포지토리에서 차트를 검색	**helm search repo nginx**
helm repo add	새로운 Helm 차트 리포지토리를 추가	**helm repo add stable https://cha**
helm repo update	리포지토리 정보를 업데이트	**helm repo update**

  - ▶ LAB: 리포드토리 등록

1.3. helm 하위 명령어

- 기본 명령어

명령어	설명	예시
helm search repo	리포지토리에서 차트를 검색	helm search repo nginx
helm show chart	차트의 메타데이터 정보를 출력	helm show chart repo/nginx
helm show values	차트의 기본 설정 값을 출력	helm show values repo/nginx
helm install	새로운 릴리스를 설치	helm install my-nginx-release repo/ng
helm upgrade	기존 릴리스를 업그레이드	helm upgrade my-nginx-release repo
helm history	특정 릴리스의 버전 기록을 확인	helm history my-nginx-release
helm rollback	특정 릴리스를 이전 버전으로 롤백	helm rollback my-nginx-release 1
helm get values	특정 릴리스의 설정 값을 출력.	helm get values my-nginx-release
helm list	설치된 릴리스 목록을 확인	helm list
helm uninstall	릴리스를 삭제	helm uninstall my-nginx-release

▶ LAB : Helm을 이용한 nginx 컨테이너 배포

1.4. Helm의 구성 요소

```
<chart-name>/ ├── Chart.yaml # 차트 메타데이터 ├── values.yaml # 기본 설정 값 ├── charts/ # 차트 의존성 관리 ├── templates/ # 템플릿 파일들 │ ├── deployment.yaml │ ├── service.yaml │ └── _helpers.tpl # 템플릿 헬퍼 파일 └── README.md # 차트에 대한 설명 (선택 사항)
```

- **Chart**: Helm 패키지로, Kubernetes 애플리케이션, 툴, 또는 서비스 실행에 필요한 모든 Kubernetes 리소스 정의 파일과 메타데이터를 포함한 집합. Chart는 애플리케이션의 청사진 역할을 담당.
  - **Chart.yaml**: Chart의 메타데이터 파일로, Chart의 이름, 버전, 애플리케이션에 대한 설명, 차트 작성자 정보 등을 포함
  - **Values**: Chart에서 사용하는 설정 값들을 정의한 YAML 파일. 사용자는 이 값을 통해 Chart의 설정을 커스터마이징. .
  - **Templates**: Kubernetes 리소스 정의 파일에서 변수와 로직을 사용해 유연하게 생성되는 파일들. Helm은 템플릿 엔진을 통해 `Values` 파일에서 제공된 값들을 반영하여 최종 Kubernetes 리소스 파일을 생성한다.
  - **Release**: 특정 차트의 배포 단위로 릴리즈를 통해 특정 버전의 차트를 추적하고 관리할 수 있음.
- Repository : helm chart를 모아두고 공유하는 저장소 **원격 저장소(bitnami, Artifact HUB), 로컬저장소(Chart Museum)**
- Chart 관리 명령어

명령어	설명	예시
helm template	템플릿을 로컬에서 렌더링하여 Kubernetes 매니페스트를 생성	**helm template my-ng repo/nginx**
helm pull	차트를 다운로드하고 압축을 해제	**helm pull nginx --unt**

▶ LAB: Helm 차트 관리

❓기출문제 : Helm을 이용한 패키지 배포

kubectl config use-context k8s

Context

- Helm을 이용해 `nginx` 웹 서버를 배포하시오.

Task:

- helm repository : https://charts.bitnami.com/bitnami
- repo name : `bitnami`

- install chart: `bitnami/nginx`
- chart name : `cka-webserver`
- 서비스 동작 중인지 확인을 위해 `k8s-worker1` 노드로 서비스 연결해본다.

🔎 **Search keyword**: https://helm.sh/docs/

▶ 답안

2. Kustomize

2.1 Kustomize 개요

- Kustomize 란?
  - **쿠버네티스 리소스(yaml파일)를 변경하지 않고 필드를 재정의하여 새로운 쿠버네티스 리소스를 생성하는 도구**
  - **템플릿 없이 YAML을 직접 관리**
    - 기존 Kubernetes 리소스를 유지하면서 운영 환경별로yaml 매니페스트 설정을 간단하게 변경
    - **공통 YAML 중복 제거 :** 여러 환경(dev, staging, prod)에서 공통 부분을 유지하면서 차이점만 정의 가능
  - **kubectl과 통합** → 추가적인 설치 없이 사용 가능

```
# kubectl 명령에 포함되어 있음 kubectl kustomize DIR [flags] [options] kubectl kustomize --help # -k 옵션을 적용해서 실행가능 kubectl apply -k
```

  - 실무 중심의 use case
    - **ConfigMap, Secret 자동 생성**
    - **네임 prefix/suffix 추가 및 patch 적용**
    - **release 별 설정 관리 (Dev, Staging, Prod)**
  - **Helm보다 가벼운 사용 방식** → Helm은 패키징 방식이지만, Kustomize는 단순히 기존 YAML을 변형하는 방식

- 사용 문법
  - Kustomize는 `kustomization.yaml` 파일을 사용하여 매니페스트를 관리
  - 주요 키워드

키워드	설명
`resources`	적용할 기본 YAML 매니페스트 목록
`patches`	특정 리소스를 수정하는 패치 적용
`configMapGenerator`	ConfigMap 자동 생성
`secretGenerator`	Secret 자동 생성
`commonLabels`	모든 리소스에 공통 라벨 추가
`commonAnnotations`	모든 리소스에 공통 어노테이션 추가
`namePrefix`	모든 리소스 이름 앞에 접두어 추가
`nameSuffix`	모든 리소스 이름 뒤에 접미어 추가

- 실행 방법

```
# Kustomize 기본 실행 kubectl apply -k <디렉터리 경로> #Kustomize 변환 결과 확인 kubectl kustomize <디렉터리 경로> # Kustomize 별도 설치 후 실행 kustomize build <디렉터리 경로> | kubectl apply -f -
```

2. kubernetes 문법 이해

참고 : https://kubernetes.io/ko/docs/tasks/manage-kubernetes-objects/kustomization/

- **exam01 : yaml 파일을 커스터마이즈하기**
  - Kustomize의 `images` 기능을 사용하여 기존 `pod.yaml`을 수정하지 않고 컨테이너 수정 가능
  - `kubectl kustomize`로 적용될 내용을 미리 확인 가능

▶ CLI

- **exam02 : resources를 통한 여러 개의 yaml 적용**
  - `resources:` 에 `service.yaml` 추가하여 **Pod + Service 배포 가능**
  - `kubectl kustomize` 로 적용될 리소스 미리 확인 가능
  - `kubectl apply -k` 로 한 번에 배포 가능
  - ▶ CLI

- exam03: **configMapGenerator**
  - `configMapGenerator` 를 사용하면 **YAML 없이 ConfigMap을 자동 생성**
  - Pod에서 `configMapKeyRef` 를 이용하여 환경 변수로 설정 가능
  - ConfigMap 변경 시 **새로운 hash값이 붙은 ConfigMap이 생성되며, Pod가 자동 업데이트 됨**
  - 동적인 설정 관리에 유용
  - ▶ CLI

- exam04: Prefix, Suffix, Labels, Annotations 적용
  - 기존 `deployment.yaml` 에 Kustomize 설정을 적용하여 **네임스페이스 추가, 리소스 명 변경(Prefix & Suffix), 라벨 추가, 어노테이션 추가**

설정	설명
`namespace: my-namespace`	네임스페이스 자동 추가
`namePrefix: dev-`	모든 리소스 이름 앞에 `"dev-"` 추가
`nameSuffix: "-001"`	모든 리소스 이름 뒤에 `"-001"` 추가
`labels:`	모든 리소스에 `app: bingo` 추가, `includeSelectors: true` 로 selector에도 반영
`commonAnnotations:`	모든 리소스에 `oncallPager: 800-555-1212` 어노테이션 추가
`resources:`	`deployment.yaml` 을 관리 대상으로 지정

  - ▶ CLI

- exam05 : **patches** Strategic Merge
  - 기존 리소스(YAML)를 유지하면서 특정 부분을 **병합(override)하여 변경**
  - Kustomize는 `patchesStrategicMerge`와 `patchesJson6902`를 통해 서로 다른 패치 메커니즘을 지원
  - `patchesStrategicMerge`는 파일 경로들의 리스트로, 각각의 파일로 **전략적 병합 패치**
    - 기존 `deployment.yaml`을 수정하기 위해 `increase_replicas.yaml`과 `set_memory.yaml`을 적용하는 구조

  ▶ CLI

- exam06 : **JSON 형식의 변경 명령을 사용**해서 기존 매니페스트를 수정
  - Kustomize의 `patches` 기능을 사용하여 기존 `pod.yaml`을 수정하지 않고 **환경 변수 추가 가능**
  - 환경 변수를 동적으로 추가하거나 변경할 때 유용함
  - patches
    - **JSON 형식의 변경 명령을 사용**해서 기존 매니페스트를 수정