

# Helm 프로그래밍 기초



helm의 프로그래밍에 대해 기초부터 알려줘



Helm에서 “프로그래밍”이라고 부르는 건 사실 일반적인 코드 작성이 아니라 **템플릿 DSL(Go 템플릿 + Helm 함수/변수)** 을 다루는 걸 말합니다.

즉, Kubernetes YAML을 그대로 쓰는 대신, 변수·조건·반복을 써서 **동적으로 생성되는 YAML**을 만드는 방식이죠.

## 1. Helm 템플릿의 기본 구조

Helm 차트(Chart)는 보통 이런 디렉터리 구조를 가집니다:

```
mychart/
  Chart.yaml           # 차트 메타정보 (이름, 버전 등)
  values.yaml          # 사용자 정의값 기본 설정
  templates/           # 실제 쿠버네티스 매니페스트 템플릿
    deployment.yaml
    service.yaml
    _helpers.tpl       # 재사용 템플릿 함수 정의
```

## 2. 템플릿 문법의 핵심

Helm은 **Go** 템플릿 언어를 기반으로 하고, 여기에 Helm만의 함수와 파이프라인을 얹었습니다.

### 2.1 변수 사용

```
metadata:
  name: {{ .Values.appName }}
```

- `{{ ... }}`: 템플릿 표현식
- `.Values`: values.yaml 또는 `--set`으로 전달된 값 접근

## 2.2 파이프라인

```
metadata:
  name: {{ .Values.appName | lower }}
```

- `` 뒤에 함수를 이어서 적용
- 여기서는 appName 값을 소문자로 변환

## 2.3 조건문

```
{{- if .Values.ingress.enabled }}
ingress:
  ...
{{- end }}
```

- ``.Values.ingress.enabled``가 true일 때만 블록 출력
- ``{{-`` 와 ``-}}``: 공백 제거

## 2.4 반복문

```
{{- range .Values.ports }}
- containerPort: {{ . }}
{{- end }}
```

- ``.Values.ports`` 리스트를 반복하며 값 출력

## 2.5 기본값 처리

```
{{ .Values.replicas | default 1 }}
```

- ``.Values.replicas`` 값이 없으면 1 사용

## 2.6 재사용 가능한 헬퍼 함수

``\_helpers.tpl``:

```
{{- define "mychart.fullname" -}}  
{{ .Release.Name }}-{{ .Chart.Name }}  
{{- end -}}
```

다른 템플릿에서 호출:

```
metadata:  
  name: {{ include "mychart.fullname" . }}
```

---

### 3. values.yaml 예시

```
appName: MyApp  
replicas: 2  
ports:  
  - 8080  
  - 9090  
ingress:  
  enabled: true
```

---

### 4. 실전 예제: Deployment 템플릿

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: {{ include "mychart.fullname" . }}  
spec:  
  replicas: {{ .Values.replicas | default 1 }}  
  selector:  
    matchLabels:  
      app: {{ .Values.appName }}  
  template:  
    metadata:  
      labels:  
        app: {{ .Values.appName }}  
    spec:  
      containers:  
        - name: {{ .Values.appName | lower }}
```

```
image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
ports:
{{- range .Values.ports }}
  - containerPort: {{ . }}
{{- end }}
```

## 5. 실행 흐름

1. `values.yaml`에 기본 값 작성
2. `helm install myapp ./mychart --set appName=TestApp`
3. Helm이 템플릿을 values 값과 합쳐서 **순수 YAML** 생성
4. 생성된 YAML이 `kubectl apply`처럼 Kubernetes에 배포

원하시면 제가 **Go 템플릿 문법 + Helm 함수 사전**을 정리해서,  
minikube에서 바로 테스트 가능한 샘플 차트까지 만들어 드릴 수 있습니다.  
그렇게 하면 지금 WSL2 환경에서 바로 실습 가능하실 거예요.  
그렇게 진행해 드릴까요?



응



12s 동안 생각함

- 안녕하세요! 템플릿 및 Helm 기능에 대한 핵심 정보를 드리겠습니다. Go 템플릿 문법, 파이프라인, 함수 사용, 자주 쓰이는 Helm Sprig 함수들 (예: `default`, `quote`, `toYaml`, `indent` 등)을 포함할 예정이고, WSL2와 Minikube에서의 테스트 명령까지 포함할게요. 별도 요청 없으면 zip 파일 제공은 생략할게요. 간결하지만 상세하게 준비할게요!

🕒 완료

좋습니다. 바로 손에 잡히게 정리해 드릴게요.

(WSL2 + Minikube 기준, Helm 템플릿 DSL 핵심 → 실습용 차트 → 테스트 명령 순서)

# 1) Helm 템플릿 DSL 치트시트 (필수만 쓱쓱)

## 기본 바인딩

- ``.`` : 현재 컨텍스트(대개 최상위에서 릴리스/차트/값 전부 접근 가능)
- ``.Values`` : `values.yaml` + `--set`` 값
- ``.Chart``, ``.Release``, ``.Capabilities`` : 차트/릴리스/클러스터 정보

## 변수, 파이프라인

```
{{- $name := .Values.app.name -}}
metadata:
  name: {{ $name | lower | trunc 63 | trimSuffix "-" }}
```

## 조건

```
{{- if .Values.ingress.enabled }}
# ... 인그레스 리소스
{{- else }}
# ... 대안 리소스
{{- end }}
```

## 반복

```
{{- range $k, $v := .Values.labels }}
  {{ $k }}: {{ $v | quote }}
{{- end }}
```

## 자주 쓰는 함수(진짜 자주 씬)

- ``default x`` : 값이 비었을 때 기본값
- ``quote`` : 따옴표 감싸기
- ``toYaml`` : 맵/리스트 → YAML 문자열
- ``nindent N`` : 개행 후 N칸 들여쓰기
- ``include "name" .`` : ``_helpers.tpl``에 정의한 블록 호출

- ``required "msg" VALUE`` : 필수값 강제
- ``tpl`` : 문자열 안의 템플릿 다시 렌더링(고급)
- ``printf``, ``replace``, ``b64enc``, ``sha256sum`` 등: 문자열/해시 유틸

## 헬퍼 정의 & 사용

`_helpers.tpl`

```
{{- define "hello.fullname" -}}
{{- printf "%s-%s" .Release.Name .Chart.Name | trunc 63 | trimSuffix "-" -}}
{{- end -}}
```

사용

```
metadata:
  name: {{ include "hello.fullname" . }}
```

## YAML 들여쓰기 콤보(실전 최다)

```
labels:
{{- toYaml .Values.podLabels | nindent 2 }}
```

``toYaml``로 문자열화 → ``nindent``로 들여쓰기까지 한 번에.

## 2) Minikube에서 바로 돌리는 실습 차트

아래 그대로 파일 만들면 됩니다. 차트 이름은 ``helloapp``.

```
helloapp/
  Chart.yaml
  values.yaml
  templates/
    _helpers.tpl
    deployment.yaml
    service.yaml
    ingress.yaml
    hpa.yaml
  NOTES.txt
```

## Chart.yaml

```
apiVersion: v2
name: helloapp
description: A minimal app for Helm+Minikube practice
type: application
version: 0.1.0          # 차트 버전
appVersion: "1.0.0"    # 실제 앱(이미지) 버전 표기
```

## values.yaml

```
app:
  name: hello
image:
  repository: nginx
  tag: "1.25-alpine"
  pullPolicy: IfNotPresent

replicaCount: 1

service:
  type: NodePort      # Minikube는 NodePort가 편합니다
  port: 80
  nodePort: 30080     # 고정 노드포트(충돌시 변경)

ingress:
  enabled: false      # Minikube에서 ingress addon 켜면 true로
  className: ""       # "nginx" 등 컨트롤러 클래스
  hosts:
    - host: hello.local
      path: /
  tls: []             # 예: [{ hosts: ["hello.local"], secretName: "hello-tls"
}]

resources: {}
nodeSelector: {}
tolerations: []
affinity: {}

podLabels:
  tier: web
```

```
probes:
  enabled: true
  path: /
  port: http
  initialDelaySeconds: 3
  periodSeconds: 10

hpa:
  enabled: false
  min: 1
  max: 5
  cpu: 80
```

## templates/\_helpers.tpl

```
{{- define "helloapp.fullname" -}}
{{- printf "%s-%s" .Release.Name .Chart.Name | trunc 63 | trimSuffix "-" -}}
{{- end -}}

{{- define "helloapp.labels" -}}
app.kubernetes.io/name: {{ .Chart.Name }}
app.kubernetes.io/instance: {{ .Release.Name }}
app.kubernetes.io/version: {{ .Chart.AppVersion }}
app.kubernetes.io/managed-by: {{ .Release.Service }}
{{- end -}}
```

## templates/deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "helloapp.fullname" . }}
  labels:
    {{ include "helloapp.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount | default 1 }}
  selector:
    matchLabels:
      app: {{ .Values.app.name }}
  template:
```



```

    metadata:
      labels:
        app: {{ .Values.app.name }}
    {{- if .Values.podLabels }}
    {{ toYaml .Values.podLabels | nindent 8 }}
    {{- end }}
    spec:
      containers:
        - name: {{ .Values.app.name }}
          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy | default "IfNotPresent" }}
      ports:
        - name: http
          containerPort: 80
    {{- if and .Values.probes.enabled (.Values.probes.path) }}
      readinessProbe:
        httpGet:
          path: {{ .Values.probes.path }}
          port: {{ .Values.probes.port }}
          initialDelaySeconds: {{ .Values.probes.initialDelaySeconds }}
          periodSeconds: {{ .Values.probes.periodSeconds }}
      livenessProbe:
        httpGet:
          path: {{ .Values.probes.path }}
          port: {{ .Values.probes.port }}
          initialDelaySeconds: {{ .Values.probes.initialDelaySeconds }}
          periodSeconds: {{ .Values.probes.periodSeconds }}
    {{- end }}
  }}

```

## templates/service.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: {{ include "helloapp.fullname" . }}
  labels:
    {{ include "helloapp.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type | default "ClusterIP" }}
  selector:

```

```

    app: {{ .Values.app.name }}
ports:
  - name: http
    port: {{ .Values.service.port }}
    targetPort: 80
    {{- if eq .Values.service.type "NodePort" }}
    nodePort: {{ .Values.service.nodePort | default 0 }}
    {{- end }}

```

## templates/ingress.yaml

```

{{- if .Values.ingress.enabled }}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "helloapp.fullname" . }}
  labels:
    {{ include "helloapp.labels" . | nindent 4 }}
    {{- if .Values.ingress.className }}
  annotations:
    kubernetes.io/ingress.class: {{ .Values.ingress.className }}
    {{- end }}
spec:
  {{- if .Values.ingress.className }}
  ingressClassName: {{ .Values.ingress.className }}
  {{- end }}
  rules:
    {{- range .Values.ingress.hosts }}
    - host: {{ .host }}
      http:
        paths:
          - path: {{ .path }}
            pathType: Prefix
            backend:
              service:
                name: {{ include "helloapp.fullname" $ }}
                port:
                  number: {{ $.Values.service.port }}
    {{- end }}
    {{- if .Values.ingress.tls }}
  tls:

```

```
{{ toYaml .Values.ingress.tls | nindent 4 }}
{{- end }}
{{- end }}
```

## templates/hpa.yaml

```
{{- if .Values.hpa.enabled }}
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {{ include "helloapp.fullname" . }}
  labels:
    {{ include "helloapp.labels" . | nindent 4 }}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {{ include "helloapp.fullname" . }}
  minReplicas: {{ .Values.hpa.min }}
  maxReplicas: {{ .Values.hpa.max }}
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: {{ .Values.hpa.cpu }}
{{- end }}
```

## templates/NOTES.txt

- 1) Minikube에서 노드포트 바로 확인:  
minikube service {{ include "helloapp.fullname" . }} --url -n {{ .Release.Namespace }}
- 2) 인그레스 사용 시:
  - minikube addons enable ingress
  - /etc/hosts에 {{ (index .Values.ingress.hosts 0).host }} 추가 후 접속

### 3) 설치/테스트 명령 (WSL2 + Minikube)

```
# 네임스페이스 준비
kubectl create ns demo

# 린트 + 템플릿 출력(적용 전 미리 보기)
helm lint ./helloapp
helm template hello ./helloapp -n demo | head -n 80

# 설치 (NodePort 노출)
helm install hello ./helloapp -n demo

# 서비스 URL 확인(가장 쉬움)
minikube service hello-helloapp --url -n demo
# 출력된 URL로 접속 (curl 또는 브라우저)

# 상태 점검
kubectl get all -n demo
kubectl describe deploy,svc -n demo
kubectl logs deploy/hello-helloapp -n demo --tail=100

# 값 변경 후 롤링 업데이트
helm upgrade hello ./helloapp -n demo --set replicaCount=2

# (옵션) 인그레스 실습
minikube addons enable ingress
helm upgrade hello ./helloapp -n demo --set ingress.enabled=true \
  --set ingress.className=nginx \
  --set ingress.hosts[0].host=hello.local --set ingress.hosts[0].path=/
# /etc/hosts에 127.0.0.1 hello.local 추가 후 접속(터널 사용 시)
# 또는: minikube tunnel & # 관리자 권한 콘솔에서

# 제거
helm uninstall hello -n demo
kubectl delete ns demo
```

Minikube에선 **Pod IP 직접 curl**보다

`minikube service ... --url` 이 안정적이고 쉬워요.

인그레스는 애드온 켜 둔 뒤 클래스/호스트 세팅까지 맞아야 Health가 정상 나옵니다.

## 4) 다음 스텝(선택)

- Post-renderer로 Kustomize 섞기
- ``tpl``로 values 안 문자열 재렌더링
- ``required``로 필수 파라미터 강제
- ``helm test`` 혹은 간단한 e2e 체크 추가
- ArgoCD와 연계(Repo 구조/Value 계층/Sync Policy)

원하시면 위 차트를 **GitOps용 리포 구조**(예: ``charts/``, ``environments/``)로 재배치하고, ArgoCD Application 매니페스트까지 붙여드릴게요.

지금은 여기까지 해보시고, 막히는 지점 찍어 주시면 바로 거기부터 깔끔히 이어가겠습니다.