APUNTES HELM

Manuel Vergara
2022

Requísitos del curso:

- Conocimientos de Linux o Windows
- Conocimientos básicos de contenedores y Docker
- Conocimientos básicos de Kubernetes

Índice

Este documento contiene los apuntes tomados en el curso «Helm 3: Despliega aplicaciones en Kubernetes» impartido por Apasoft Training en diciembre de 2022. El curso udemy consta de 5 horas aproximadamente de vídeo-tutoriales. Las prácticas aquí contenidas tuvieron una duración de alrededor de unas 15 horas.

Los apuntes no fueron pensados para compartirlos, por ello pueden tener lagunas de información o contenido adicional respecto al curso, ya que se redactaron para recordar procedimientos y conceptos que el autor creyó relevantes. Teniendo un documento, a mi parecer, tan completo y entendiendo que el conocimiento debe ser libre se decidió compartirlo.

Si te parece útil este documento puedes agradecerlo a través de las vías de contacto de la web https://vergaracarmona.es

Recuerda,

> *"Quien se corta su propia leña se calienta dos veces"*

TEMA 1 - Introducción a Helm

Helm te ayuda a administrar las aplicaciones de Kubernetes. Podrás definir, instalar y actualizar incluso la aplicación de Kubernetes más compleja. Las Charts son fáciles de crear, versionar, compartir y publicar.

Web: https://helm.sh/ **Documentación**: https://helm.sh/docs/

Algunos conceptos:

- Helm es proyecto graduado de la CNCF y es mantenido por Comunidad de Helm.
- Se puede categorizar como un gestor de paquetes para Kubernetes.
- Se considera un estándar dentro del mundo de Kubernetes.
- Podemos compararlo con otros gestores de paquetes como apt, yum, homebrew, etc
- De este modo se simplifica mucho el proceso de crear, gestionar y desplegar aplicaciones.
- Todo se almacena en un paquete de instalación. Así no lo tenemos repartidos en ficheros.
- Permite gestionar versiones
- Permite hacer rollback de una forma rápida y sencilla.

1.1. - Componentes

La arquitectura de Helm es muy sencilla.

Chart – el paquete de Helm. Un objeto compuesto de ficheros y aplicaciones para desplegar la aplicación concreta.

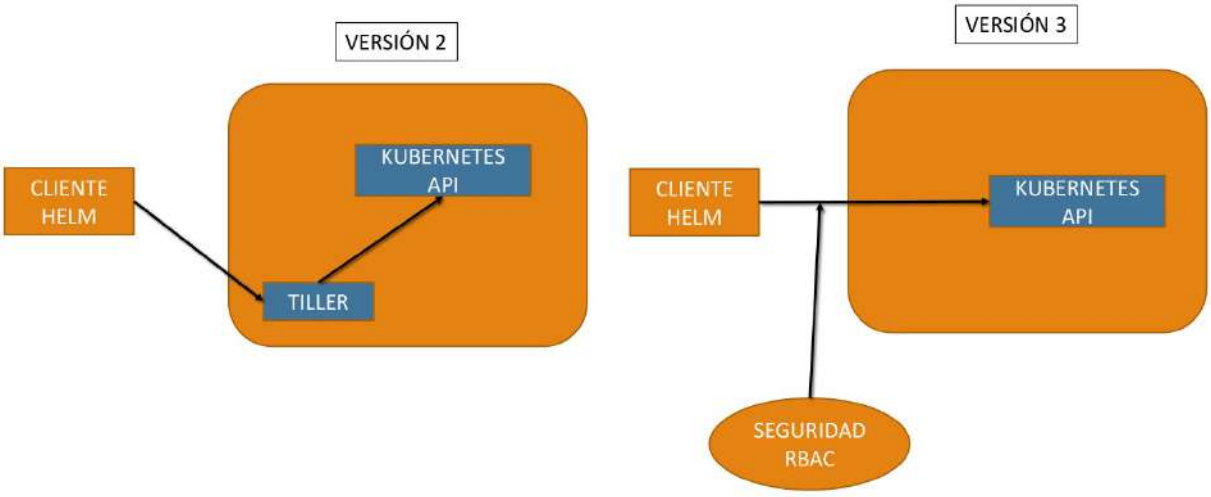
Repositorio – Donde se puede depositar los Charts para poder distribuirlos.

Release – Es la ejecución de un Chart dentro de un clúster Kubernetes. Por cada despliegue tendríamos una release distinta.

1.2. - Comparando la Versión 2 con la versión 3

En la versión 2 se utilizaba un servidor llamado «Tiller» y el problema que tenía es que había que configurar la seguridad para conectarse con Kubernetes. Ahora RBAC está integrado con Kubernetes, con lo cuál evitamos todas la configuración.

Seguridad RBAC (Role based access control): Este modelo de seguridad permite asignar funciones y autorizaciones en la infraestructura informática de una organización. El término “basado en roles” es clave para entender cómo funciona el RBAC, ya que lo distingue de otros conceptos de seguridad, como el mandatory access control. En este modelo, el administrador del sistema asigna un nivel y una categoría de seguridad a cada usuario y objeto en dependencia del rol que cumple. El sistema operativo enlaza automáticamente los dos niveles y luego concede o deniega el acceso.



1.3. - Descargar e instalar Helm

La instalación es sencilla según lo explican en la página oficial: https://helm.sh/

Documentación: https://helm.sh/docs/intro/quickstart/

Guía básica: https://gitea.vergaracarmona.es/man-linux/Guia-Tutorial-kubernetes/src/branch/main/guias/05-helm.md#instalaci%C3%B3n-de-helm

TEMA 2 - Repositorios

Un repositorio HELM es un sitio HTTP que contiene un conjunto de charts o paquetes Helm.

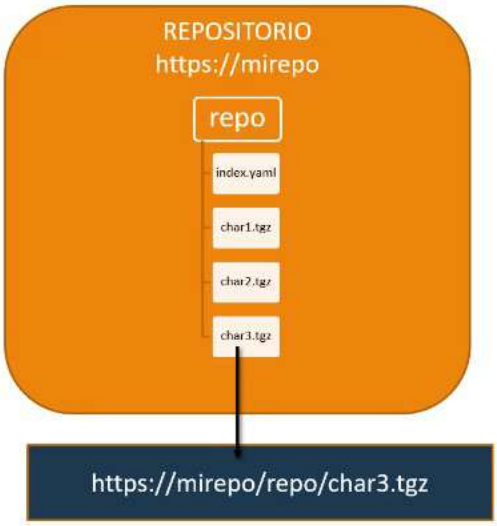
El repositorio contiene un archivo «índice» que detalla el contenido del repositorio. Se llama index.yaml. (En Helm se utiliza también yaml.)

Los charts aparecen empaquetados como .tar.gz. Es un como un sitio web que sirve los repositorios.

«Helm repo» nos permite añadir repositorios, indexarlos, listarlo, borrarlo, actualizarlos.

Algunas de las opciones donde podemos crear un repositorio son:

- Servidor web, Servicio de almacenamiento como GitHub, Google Cloud Storage (GCS) bucket, Amazon S3 bucket...



Para gestionar un repositorio puedo utilizar el comando:

helm repo

Podemos ver la ayuda si lo ponemos como se indica anteriormente. Para añadir un repositorio estable de utilizamos:

helm repo add stable https://charts.helm.sh/stable

Podemos ver los repositorios instalados con

helm repo list

Podemos buscar en el repo stable con

helm search repo stable

si hacemos una tubería con wc -l sabremos el número de charts que tiene el repo.

Podemos ver los contextos que utiliza helm en este momento

helm env

```
❯ helm env
HELM_BIN="/snap/helm/353/helm"
HELM_CACHE_HOME="/home/v/.cache/helm"
HELM_CONFIG_HOME="/home/v/.config/helm"
HELM_DATA_HOME="/home/v/.local/share/helm"
HELM_DEBUG="false"
HELM_KUBEAPISERVER=""
HELM_KUBEASGROUPS=""
HELM_KUBEASUSER=""
HELM_KUBECAFILE=""
HELM_KUBECONTEXT=""
HELM_KUBETOKEN=""
HELM_MAX_HISTORY="10"
HELM_NAMESPACE="default"
HELM_PLUGINS="/home/v/.local/share/helm/plugins"
HELM_REGISTRY_CONFIG="/home/v/.config/helm/registry.json"
HELM_REPOSITORY_CACHE="/home/v/.cache/helm/repository"
HELM_REPOSITORY_CONFIG="/home/v/.config/helm/repositories.yaml"
```

Al igual que docker, existe un aglutinador de repositorio oficial de helm: https://artifacthub.io/ Aquí podemos almacenar y descargar toda clase de repositorios. Siempre es mejor descargar versiones oficiales. Cada chart contiene el repositorio al que pertenece, más el nombre del chart más la versión. En cada chart de la web nos dará el código para añadir o instalar la chart.

Podemos extraer el index.yaml de los repositorios. Por ejemplo, añadir bitnami deberíamos usar el comando

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

Si añadimos index.yml a la dirección y entramos en un navegador podríamos ver todos los datos respecto a las charts incluídas. También se puede descargar con el comando:

```
curl https://charts.bitnami.com/bitnami/index.yaml > index.yaml
```

Normalmente, no utilizaremos este fichero para buscar porque lo podemos hacer mediante el comando «helm» ayudado con «grep», por ejemplo, pero conviene estudiarlo para saber los datos que contiene.

Para buscar paquetes yaml en los repositorios podemos usar:

En la nube

```
helm search hub
```

En local

```
helm search repo
```

Podemos poner la salida con -o <json, table>

Si añadimos -l podremos ver las versiones y no solo la última.

Con --version puedes escoger la versión concreta del paquete.

Para borrar un repositorio local lo hacemos con este comando:

```
helm repo remove <nombre-repo>
```

Realmente lo único que hacemos es borrar un puntero al repositorio real que está en la nube.

TEMA 3 - Charts. Trabajar con paquetes

3.1. - Ejemplo instalar apache

Vamos a instalar una aplicación mediante una chart. Buscaremos con los comandos anteriores las charts de Apache en el hub. Para tener la url entera utilizaré el comando de una manera especial:

```
helm search hub "Apache http" -o json | jq
```

Ahora instalaremos la bitnami, que ya la tenemos en local.

```
helm install apache1 bitnami/apache
```

Podemos ver todos los release con

```
helm list
```

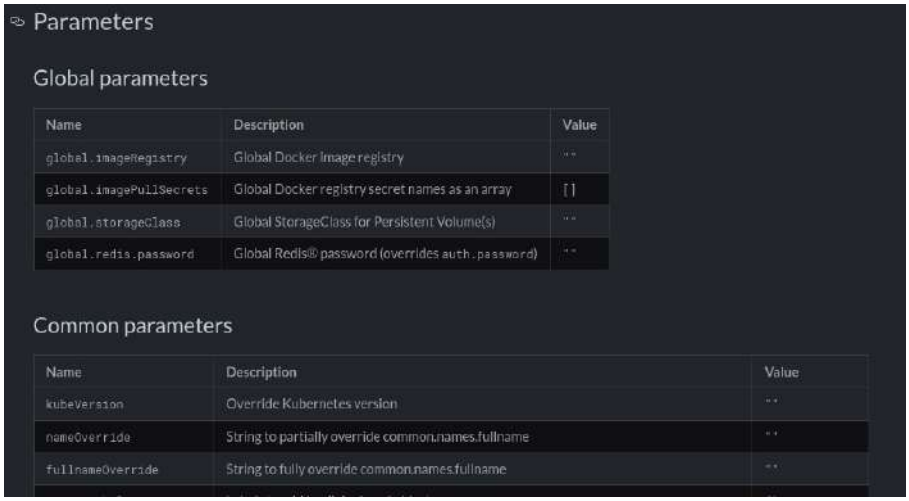
Y con la opción -A nos aseguramos que liste de todos los namespaces.

Con esa instalación de bitnami, en concreto, nos despliega un deployment con replicaset, service y un pod.

3.2. - Ejemplo instalar redis

En la página concreta de la chart podemos encontrar información extra para añadir a nuestro release y saber que es lo que contiene la chart: https://artifacthub.io/packages/helm/bitnami/redis

Por ejemplo, nos indica los parámetro que podemos añadirle:



Parameters

Global parameters

Name	Description	Value
global.imageRegistry	Global Docker image registry	""
global.imagePullSecrets	Global Docker registry secret names as an array	[]
global.storageClass	Global StorageClass for Persistent Volume(s)	""
global.redis.password	Global Redis® password (overrides auth.password)	""

Common parameters

Name	Description	Value
kubeVersion	Override Kubernetes version	""
nameOverride	String to partially override common.names.fullname	""
fullnameOverride	String to fully override common.names.fullname	""

En este caso nos dan el nombre, la descripción y el valor de cada parámetro.

Lanzamos redis

```
helm install redis1 bitnami/redis
```

En este caso despliega 2 statefulset (uno con 3 replicas), 3 servicios, 4 pods (1 master y 3 replicas), 4 pv.

3.3. - Información sobre releases

Podemos ver información de las releases que tenemos con

```
helm list -A
```

También podemos pedir información de una release concreta

```
helm status redis1
```

Otra información que podemos extraer es con el comando get. Tenemos estas opciones:

all	Descargar toda la información de las release
hooks	download all hooks for a named release
manifest	download the manifest for a named release
notes	download the notes for a named release
values	download the values file for a named release

Por ejemplo, podríamos ver toda la información del manifest con

```
helm get manifest apache1
```

También lo podríamos ver en un archivo con

```
helm get manifest apache1 > apache1.yaml
```

Podríamos recuperar toda la información con el all

```
helm get all apache1
```

3.4. - Información sobre CHARTS

Podemos ver la información sobre los Charts que tenemos con el comando show. Tiene estas opciones:

all	mostrar toda la información sobre Charts
chart	show the chart's definition
crds	show the chart's CRDs
readme	show the chart's README
values	show the chart's values

Por ejemplo

helm show readme bitnami/apache

También podemos guardarlo en un fichero para verlo tranquilamente, incluso con un visualizador de Markdown lo podríamos ver mejor. En principio, la información que descarguemos con show será la misma que tengamos en la url de la Chart.

Igual que con get podemos ver toda la información con all.

3.5. - Upgrade

Podemos actualizar una release lanzada. Por ejemplo, del apache anterior podríamos descargar de nuevo la versión del repositorio

helm upgrade apache1 bitnami/apache

Para cambiar cosas podríamos ir a buscarlo con show, por ejemplo

helm show readme bitnami/apache | grep port

Con el anterior comando nos dará todos los parámetros del README con «port». También podríamos ver en el values el puerto que instala

helm show values bitnami/apache | grep port

Para cambiar el puerto lo podríamos hacer con upgrade con la opción --set:

helm upgrade --set service.port=8080 apache1 bitnami/apache

Ahora podemos utilizar get para ver los valores que tiene la release

helm get values apache1

También podemos ver el manifest creado

helm get manifest apache1

Se pueden cambiar más cosas a la vez con set

helm upgrade --set service.ports.http=8080 --set service.ports.https=8443 --set service.type=NodePort apache1 bitnami/apache

Realmente, para cambiar valores es mejor crear un fichero para cambiarlo todo y no hacerlo a través de helm.

Podemos descargar los valores por defecto en un fichero yaml.

Helm show values bitnami/apache > valores.yaml

Por ejemplo, podemos cambiar las replicas, el puerto, etc Luego tendremos que utilizar el comando añadiendo el fichero de configuración:

helm upgrade -f valores.yaml apache1 bitnami/apache

3.6. - Rollback

Esto se podría especificar en nuestros manifiestos de kubernetes. Con helm, podríamos hacerlo de otra manera. Podemos ver el historial de la release con history

helm history apache1

En este caso tenemos varias versiones que hemos creado

```
❯ helm history apache1
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec  4 14:01:52 2022        superseded      apache-9.2.7    2.4.54          Install complete
2               Sun Dec  4 18:12:34 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
3               Sun Dec  4 18:33:22 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
4               Sun Dec  4 18:41:43 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
5               Sun Dec  4 18:42:00 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
6               Sun Dec  4 18:45:27 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
7               Sun Dec  4 18:57:44 2022        deployed        apache-9.2.7    2.4.54          Upgrade complete
```

Le vamos a decir de volver a la versión 3.

helm rollback apache1 3

Si miramos el history de nuevo veremos que tenemos una nueva versión que en la descripción nos dice que fue creada a través de rollback

```
❯ helm rollback apache1 3
Rollback was a success! Happy Helming!
~/Documents/projectes/helm                                              ⎈ kind-kind ⊘ ES 19:02:44
❯ helm history apache1
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec  4 14:01:52 2022        superseded      apache-9.2.7    2.4.54          Install complete
2               Sun Dec  4 18:12:34 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
3               Sun Dec  4 18:33:22 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
4               Sun Dec  4 18:41:43 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
5               Sun Dec  4 18:42:00 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
6               Sun Dec  4 18:45:27 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
7               Sun Dec  4 18:57:44 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
8               Sun Dec  4 19:02:44 2022        deployed        apache-9.2.7    2.4.54          Rollback to 3
```

Esta es la genialidad de helm, el control de versiones.

3.7. - Borrar

Muy fácil:

helm uninstall apache1

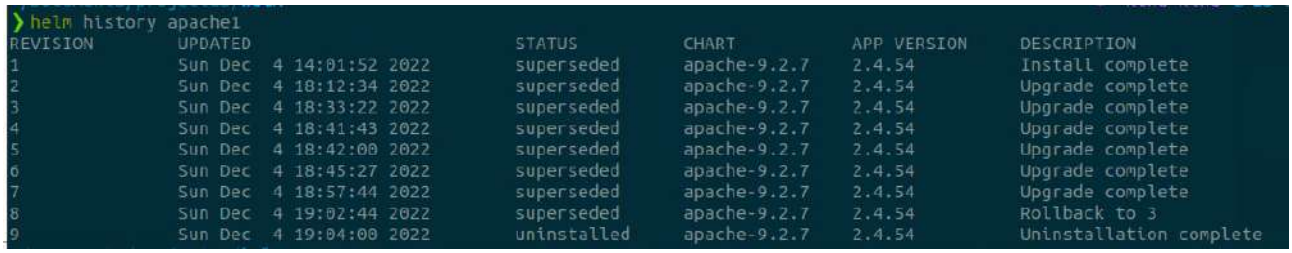
Antes podríamos ver lo que hay dentro con la opción --dry-run

helm uninstall --dry-run apache1

También podríamos guardar el historico con --keep-history

helm uninstall --keep-history apache1

Entonces, en el historial podríamos ver incluso la desinstalación. Esto nos servirá para mantener un histórico de lo que hemos ido haciendo.



```
❯ helm history apache1
REVISION        UPDATED                         STATUS          CHART           APP VERSION     DESCRIPTION
1               Sun Dec  4 14:01:52 2022        superseded      apache-9.2.7    2.4.54          Install complete
2               Sun Dec  4 18:12:34 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
3               Sun Dec  4 18:33:22 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
4               Sun Dec  4 18:41:43 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
5               Sun Dec  4 18:42:00 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
6               Sun Dec  4 18:45:27 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
7               Sun Dec  4 18:57:44 2022        superseded      apache-9.2.7    2.4.54          Upgrade complete
8               Sun Dec  4 19:02:44 2022        superseded      apache-9.2.7    2.4.54          Rollback to 3
9               Sun Dec  4 19:04:00 2022        uninstalled     apache-9.2.7    2.4.54          Uninstallation complete
```

TEMA 4 - Creación de Charts

4.1. - Ficheros y directorios de un Chart

DIRECTORIO	DESCRIPCIÓN
Chart.yaml	Fichero YAML que contiene información sobre el chart
LICENSE	Opcional: fichero de texto con la licencia del chart
README.md	Opcional: fichero README
values.yaml	Valores por defecto para este chart
values.schema.json	Opcional: Esquema JSON que determina la estructura del fichero values.yaml
charts/	Directorio que contiene los charts de los que depende este chart
templates/	Directorio de plantillas que combinado con los valores generará los ficheros de Manifes de Kubernetes.
templates/NOTES.txt	Opcional: Fichero plano con las notas sobre el uso del chart

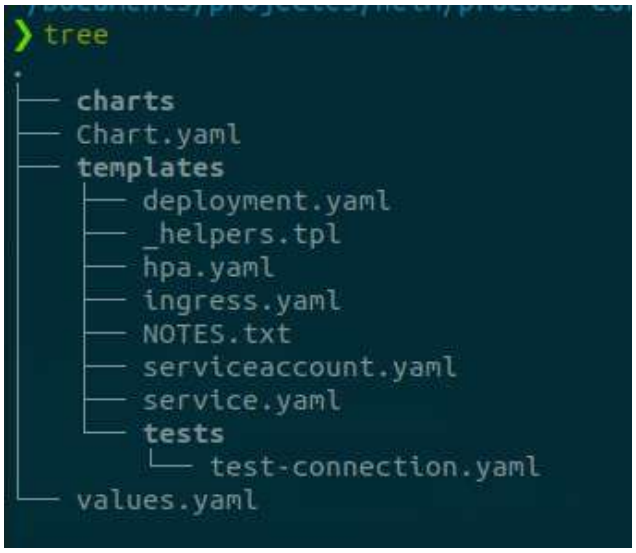
El lenguaje que usa Helm es Go con algunos cambios.

4.2. - Crear la estructura

Creamos un directorio para la charts. Entramos y vamos a utilizar la base que ofrece Helm con el comando.

```
Helm create chart1
```

Esto nos creará una estructura mínima con ejemplos.



4.3. - Editamos los ficheros de la estructura por defecto

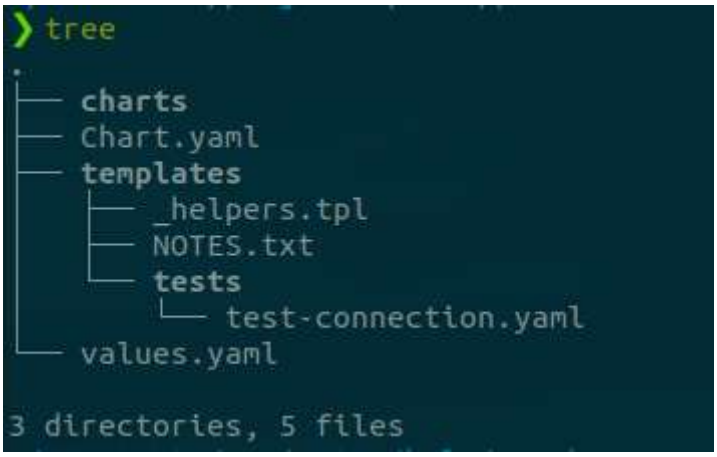
Primero editamos en fichero Chart.yaml, teniendo en cuenta https://semver.org queda así:

```
apiVersion: v2
name: chart1
description: Mi primera prueba de chart
type: application
version: 0.1.0
appVersion: "1.0.0"
```

Ahora borramos todos los yaml del directorio templates:

```
rm templates/*.yaml
```

Dejando el árbol así:



Y añadimos un yaml llamado nginx.yaml que tendrá el siguiente contenido:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx1
  labels:
    zone: prod
    version: v1
spec:
  containers:
   - name: nginx
     image: apasoft/nginx:v1
```

4.4. - Instalamos la chart

Ahora instalamos con

helm install app-nginx .

Se puede ver si se ha desplegado con

helm list

y puedo ver el pod desde kubectl

kubectl get pod

Lo normal no es instalar un chart con valores definidos, suelen ser con valores ad hoc o runtime.

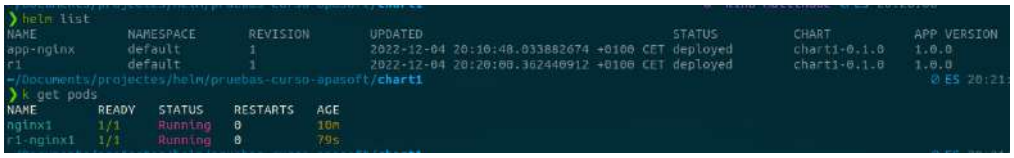
Cambiamos el nombre del pod y lo cambiamos por un valor con variable:

```
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Release.Name }}-nginx1
  labels:
    zone: prod
    version: v1
spec:
  containers:
   - name: nginx
     image: apasoft/nginx:v1
```

Y vamos a crear una nueva release.

helm install r1 .

Ahora tendremos la release anterior más la nueva y el nombre del pod no lo repite



4.5. - Comprobar pre-instalación

Podemos comprobar plantillas antes de instalar con la opción --dry-run

helm install --dry-run

O tambień con el comando

helm template .

Además, en cualquier de los comandos anteriores podemos ver detalles para debug con --debug

helm install --dry-run --debug

4.6. - Ignorar ficheros y directorios

Podemos ignorar determinados ficheros cuando hacemos la instalación con el fichero oculto .helmignore. Por defecto tiene estos ficheros y directorios:

```
.DS_Store
.git/
.gitignore
.bzr/
.bzrignore
.hg/
.hgignore
.svn/
*.swp
*.bak
*.tmp
*.orig
*~
.project
.idea/
*.tmproj
.vscode/
```

4.7. - Objetos

Los objetos se pasan a una plantilla desde el motor de plantillas.

Los objetos pueden ser simples y tener un solo valor. O pueden contener otros objetos o funciones.

Release

Values

Chart

Files

Capabilities

Template

Podemos añadir unos cuantos valores en anotaciones. Quedando el manifest así:

```
apiVersion: v1
kind: Pod
metadata:
  name: {{ .Release.Name }}-nginx1
  labels:
    zone: prod
    version: v1
  annotations: {
    fecha: tiempo-{{.Release.Time}} ,
    namespace: nombre-{{.Release.Namespace}},
    actualizacion: Actualizacion-{{.Release.IsUpgrade}},
    instalacion: Instalacion-{{.Release.IsInstall}},
    revision: Revision-{{.Release.Revision}},
spec:
  containers:
   - name: nginx
     image: apasoft/nginx:v1
```

Todos estos valores son predeterminados en Helm. Podemos mirar más detalles en la documentación. En la release quedaría así:

```
  annotations: {
    fecha: tiempo- ,
    namespace: nombre-default,
    actualizacion: Actualizacion-false,
    instalacion: Instalacion-true,
    revision: Revision-1,
```

El valor {{.Release.Time}} es Deprecated en la versión 3 y por eso ha aparecido nulo.

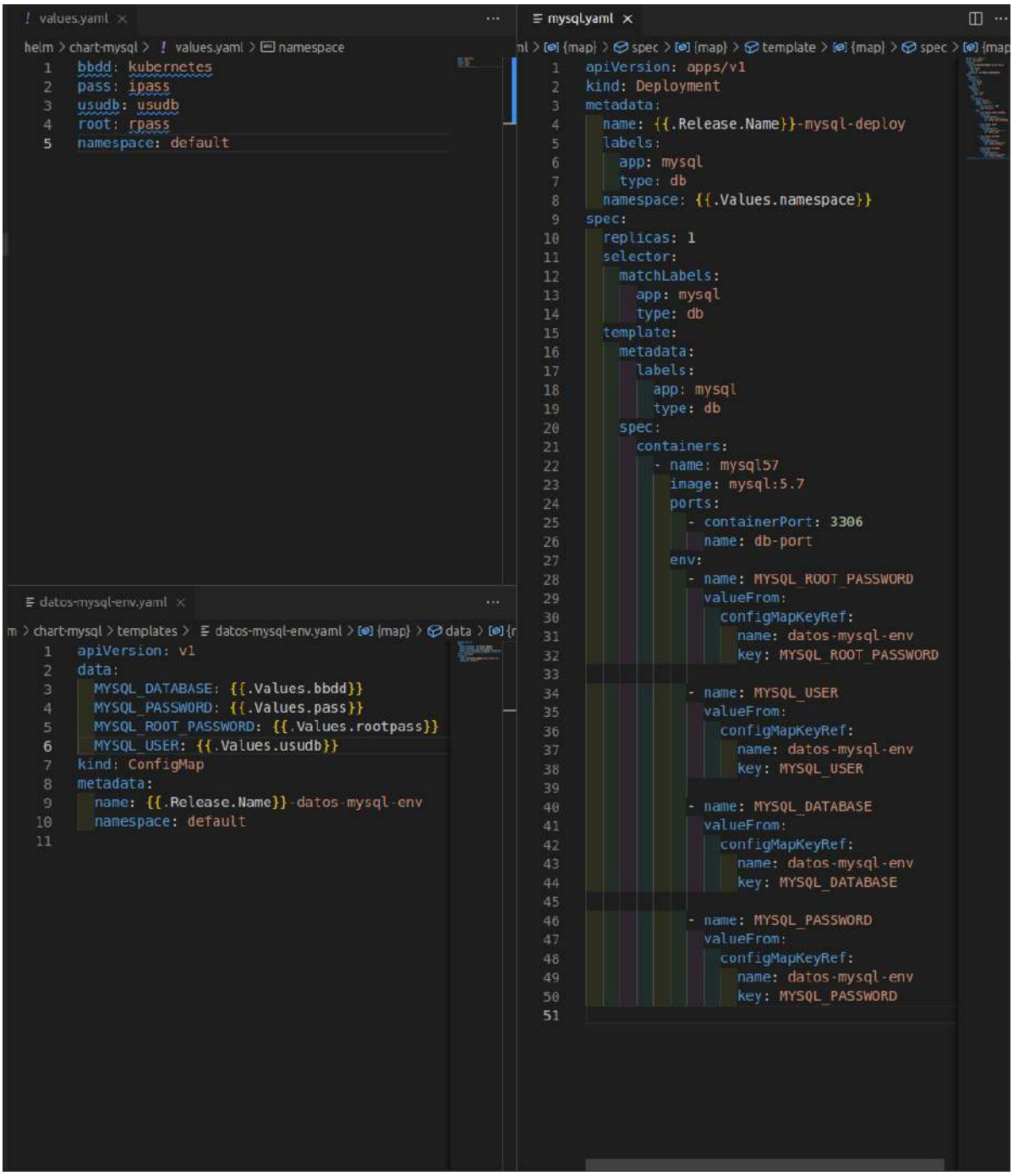
Hay que tener cuidado con los nulos, por ejemplo, si hubiesemos tenido la anotación solo con el value:

```
        fecha-ejec: {{.Release.Time}}
```

Hubiese dado un error porque yaml necesita que cada key tenga un valor: key-valor.

TEMA 5 - Values

Vamos a sustituir datos del configmap en unas variables que tendrá los valores en values.



values.yaml

```
bbdd: kubernetes
pass: ipass
usudb: usudb
root: rpass
namespace: default
```

mysql.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{.Release.Name}}-mysql-deploy
  labels:
    app: mysql
    type: db
  namespace: {{.Values.namespace}}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
      type: db
  template:
    metadata:
      labels:
        app: mysql
        type: db
    spec:
      containers:
        - name: mysql57
          image: mysql:5.7
          ports:
            - containerPort: 3306
              name: db-port
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: datos-mysql-env
                  key: MYSQL_ROOT_PASSWORD

            - name: MYSQL_USER
              valueFrom:
                configMapKeyRef:
                  name: datos-mysql-env
                  key: MYSQL_USER

            - name: MYSQL_DATABASE
              valueFrom:
                configMapKeyRef:
                  name: datos-mysql-env
                  key: MYSQL_DATABASE

            - name: MYSQL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: datos-mysql-env
                  key: MYSQL_PASSWORD
```

datos-mysql-env.yaml

```
apiVersion: v1
data:
  MYSQL_DATABASE: {{.Values.bbdd}}
  MYSQL_PASSWORD: {{.Values.pass}}
  MYSQL_ROOT_PASSWORD: {{.Values.rootpass}}
  MYSQL_USER: {{.Values.usudb}}
kind: ConfigMap
metadata:
  name: {{.Release.Name}}-datos-mysql-env
  namespace: default
```

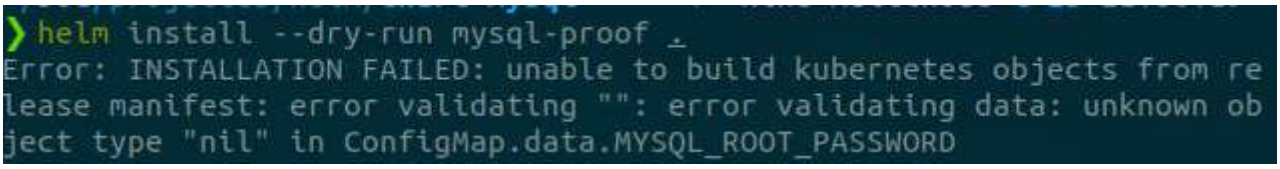
También modificaremos NOTES.txt con el texto:

Has instalado una base de datos mysql

Y promero hacemos una prueba de instalación desde la carpeta de chart:

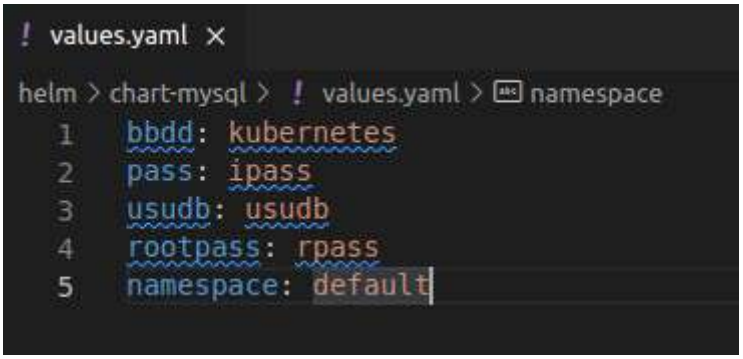
```
helm install --dry-run mysql-proof .
```

Nos da error



Este error se debe a que hay la variable MYSQL_ROOT_PASSWORD tiene un valor inexistente. Se puede ver en la captura de más arriba.

Lo corrijo cambiando root por rootpass



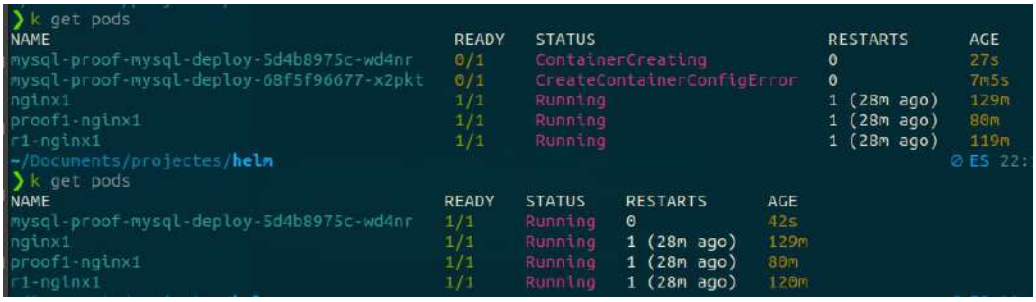
Si lanzo de nuevo el comando para probar la instalación saldrá todo correcto y al final sale la nota que introduccimos en NOTES.txt

Pero, si comprobamos el estado del pod, nos da un error de configuración. Esto se debe a que la llamada del valueFrom en las variables de entorno del deploy no estaba el valor correcto. Hay que cambiarlo en las 4 variables

Ahora relanzamos con upgrade para obtener la versión 2, espero que la correcta.

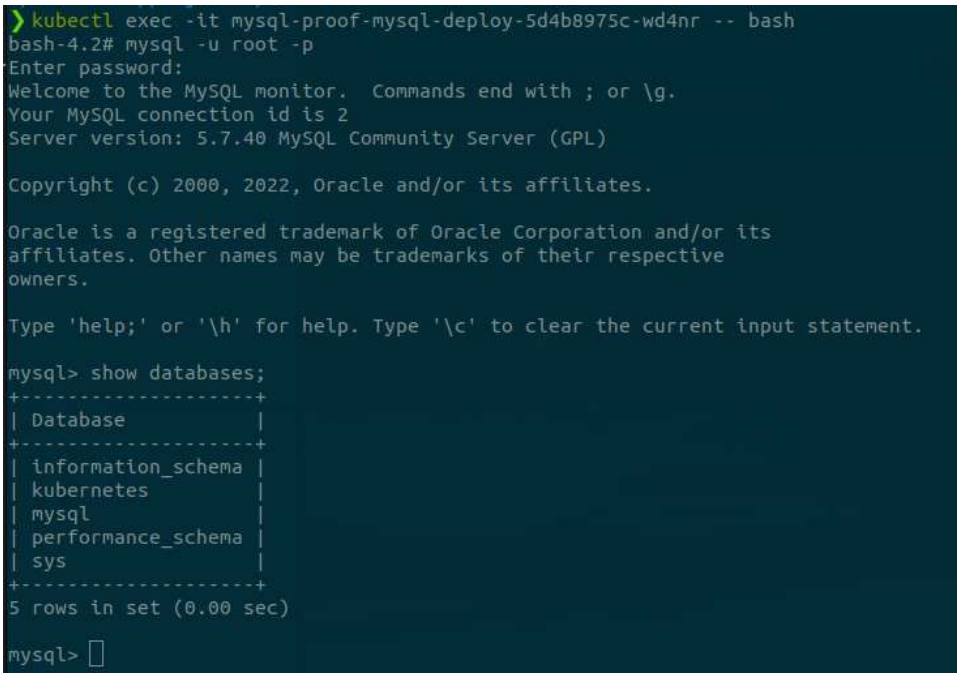
helm upgrade mysql-proof .

Esto borrará la anterior release (pod incluido) y creará otro



Podemos probar a entrar con kubectl con el comando

kubectl exec -it mysql-proof-mysql-deploy-5d4b8975c-wd4nr -- bash

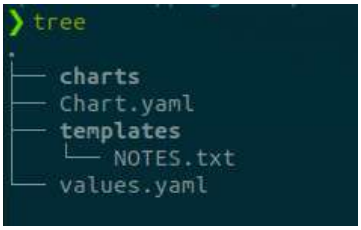


Vamos a añadir valores compuestos, que tengan más de una propiedad en la plantillas.

Crearemos otro chart

helm create chart-composite-values

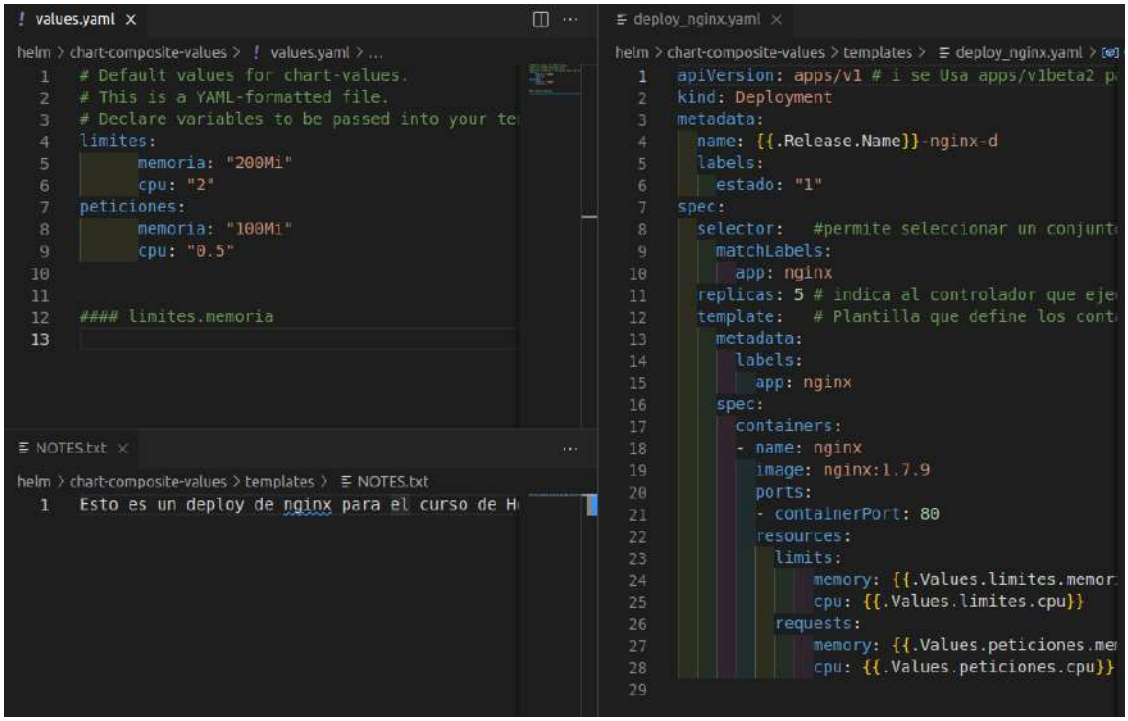
Ahora borramos lo que no necesitamos quedando el árbol así:



El values lo vaciamos, podemos utilizar el siguiente comando:

cat /dev/null > values.yaml

Ahora editamos NOTES.txt, añadimos valores a values.yaml y en templates creamos un deploy de nginx.



Si probamos vemos que todo es correcto:

helm install --dry-run value1 .

Lo mismo que antes, si todo va bien instalamos:

helm install value1 .

Comprobamos la release

helm list

Y comprobamos el despliegue de pods

kubectl get pods.

Hay que tener en cuenta, que si utilizamos las tres opciones que hemos visto para pasar valores, el **orden de carga** de los valores será el siguiente:

1. Parámetro pasados en el install o upgrade con --set
2. El fichero de valores pasado en el install o upgrade con la opción -f
3. El fichero values.yaml del chart

Vamos a ver como prioriza. Utilizaremos la última release llamada value1. Si vemos como está desplegada actualmete

helm get manifest value1

```
❯ helm get manifest value1
---
# Source: chart-composite-values/templates/deploy_nginx.yaml
apiVersion: apps/v1 # i se Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: value1-nginx-d
  labels:
    estado: "1"
spec:
  selector:   #permite seleccionar un conjunto de objetos que cumplan las cond
icione
    matchLabels:
      app: nginx
  replicas: 5 # indica al controlador que ejecute 5 pods
  template:   # Plantilla que define los containers
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
              memory: 200Mi
              cpu: 2
          requests:
              memory: 100Mi
              cpu: 0.5
```

Y vamos a añadir a nuestros directorio de chart el fichero values1.yaml con datos cambiados de limite de memoria y peticiones de memoria:

Ahora vamos a lanzar un upgrade añadiendo el fichero con la opción -f a ver que pasa

helm upgrade value1 . -f values1.yaml

Si pedimos el manifest veremos que ha hecho

```
❯ helm get manifest value1
---
# Source: chart-composite-values/templates/deploy_nginx.yaml
apiVersion: apps/v1 # i se Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: value1-nginx-d
  labels:
    estado: "1"
spec:
  selector:   #permite seleccionar un conjunto de objetos que cumplan las cond
icione
    matchLabels:
      app: nginx
  replicas: 5 # indica al controlador que ejecute 5 pods
  template:   # Plantilla que define los containers
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
              memory: 100Mi
              cpu: 2
          requests:
              memory: 50Mi
              cpu: 0.5
```

Con lo cual, el fichero que pasamos con -f tiene más prioridad que los valores de values.yaml de la chart.

Si repetimos el anterior upgrade pero añadiendo otro valor con --set veremos que tendrá más prioridad que el resto.

helm upgrade value1 . -f values1.yaml --set limites.memoria="150Mi"

Si pedimos el manifest:

```
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          limits:
              memory: 150Mi
              cpu: 2
          requests:
              memory: 50Mi
              cpu: 0.5
```

Lo que esperabamos. Pero es mejor no usar esta opción, porque al siguiente upgrade deberemos añadir también el set. La mejor práctica es dejarlo en el values de la Chart.

TEMA 6 - Condiciones y Bucles

6.1. - Variable

Vamos a utilizar una release con el siguiente árbol:

```
❯ tree chart-variables
chart-variables
├── Chart.yaml
├── templates
│   └── tomcat.yaml
└── values.yaml
```

En este ejemplo tan solo se ha modificado el tomcat.yaml de la carpeta templates, el resto es la instalación por defecto.

tomcat.yaml

```
les > templates > tomcat.yaml > {map} > sp
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: tomcat
5    labels:
6      estado: "desarrollo"
7      responsable: "juan"
8  spec:
9    containers:
10     {{ $version := "9.0"}}
11     - name: tomcat1
12       image: tomcat:{{$version}}
13
```

Aquí tenemos la variable declarada (A modo de constante, algo absurdo pero sirve de ejemplo) y en la versión de la imagen se llama a la variable. Probamos

```
❯ helm install --dry-run variables1 .
NAME: variables1
LAST DEPLOYED: Sun Dec  4 23:13:43 2022
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
MANIFEST:
---
# Source: chart-variables/templates/tomcat.yaml
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
  labels:
    estado: "desarrollo"
    responsable: "juan"
spec:
  containers:

  - name: tomcat1
    image: tomcat:9.0
```

El valor de la variable se ha añadido donde correspondía.

6.2. - Comentarios

Vamos a poner comentarios. Tenemos dos tipos de comentarios:

- Los comentarios del tipo yaml. Con un almohadilla. Se deberían utilizar para el propio pod, para la parte de Kubernetes

  # Esto es un comentario yaml

- Los comentarios dentro de llaves de tipo Helm se usan para componentes, para algo que tiene que quedar en el yaml de Helm y no pasará a Kubernetes.

  {{- /*
  Todo lo que hay aquí en medio es comentario
  */}}

6.3. - Condicionales

Como se ha comentado, el lenguaje de Helm se basa en Go y también en Split (sobretodo las funciones). Vamos a ver **una condición**:

ESTRUCTURA IF
==========
{{ if CONDICION }}
  # PROCESO
{{ else if CONDICION }}
  # HACER OTRA COSA
{{ else }}
  # OPCIÓN POR DEFECTO
{{ end }}

Cada clausula del if se debe poner con doble llave. Siempre hay que cerrar el bloque con end. Estas librerías son muy estrictas, si no ponemos alguno de los valores nos dará error.

Documentación Helm: https://helm.sh/docs/chart_template_guide/control_structures/

6.4. - Operadores lógicos

Operadores lógicos. En realidad son funciones que le llaman **Control Flow Functions**.

eq ne lt gt ge le and or not

default empty failed .....

EJEMPLOS:

if operador .Arg1 .Arg2

{{ if eq .Values.favorite.drink "coffee" }}mug: true{{ end }}

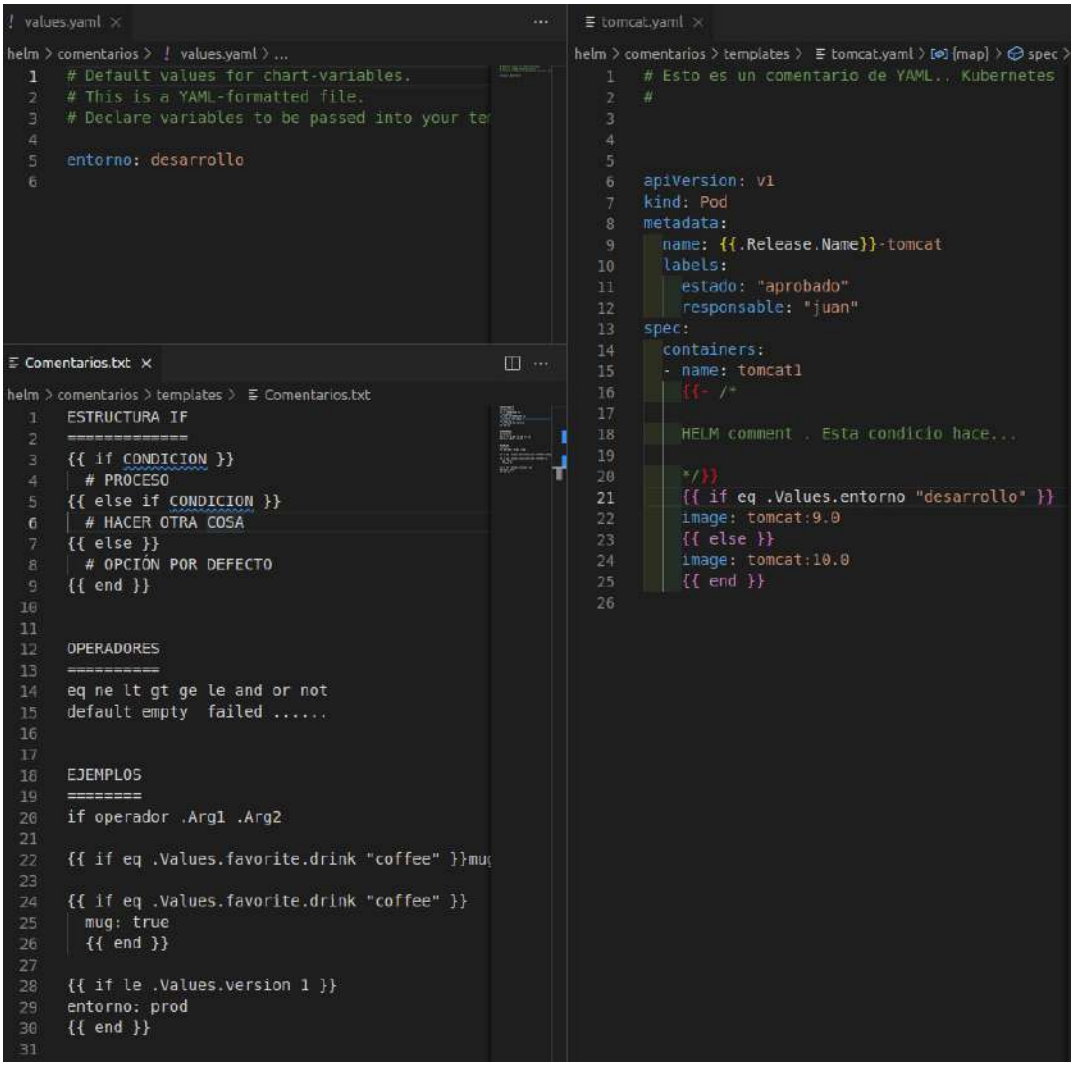
{{ if eq .Values.favorite.drink "coffee" }}
  mug: true
  {{ end }}

{{ if le .Values.version 1 }}
entorno: prod
{{ end }}

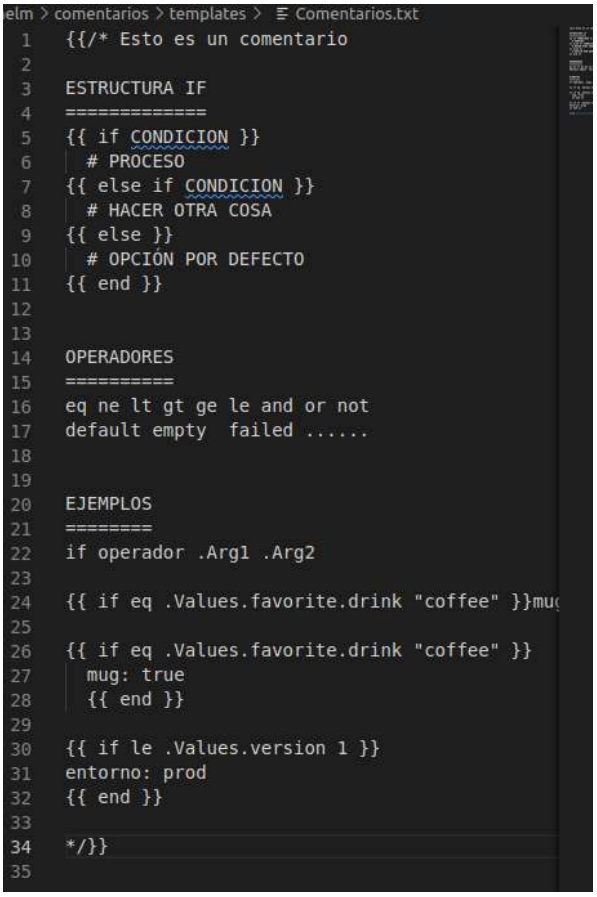
Es IMPORTANTE tener en cuenta la identación. Siendo un fichero YAML si no es correcta nos dará error.

6.5. - Práctica condicional

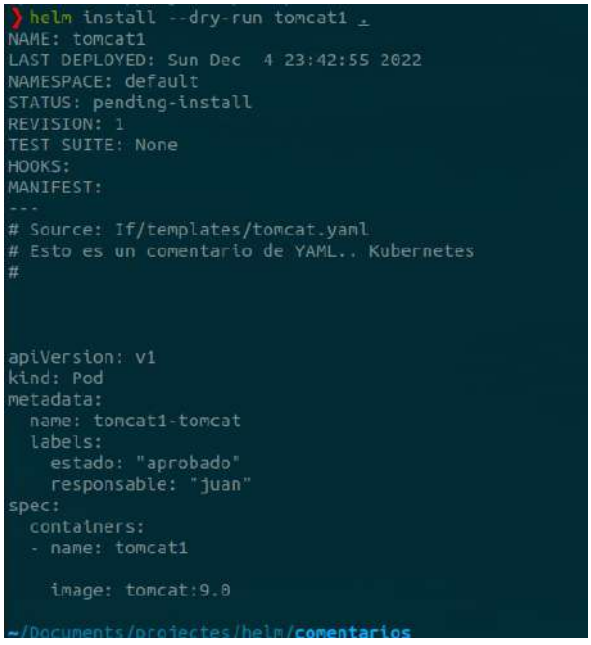
Vamos a dar las condiciones Si entorno es desarrollo tomcat será la versión 9 y sino será la versión 10.



Si lo probamos nos dará error porque el documento txt guardado en templates lo ha intentado interpretar aunque la extensión no corresponda. Vamos a comentarlo

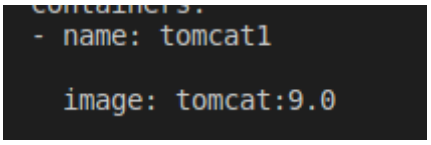


Ahora ya no protesta. Y el tomcat que nos despliega es el tomcat versión 9.0

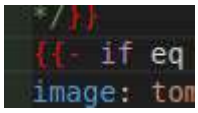


6.6. - Espacios en blanco en los bloques

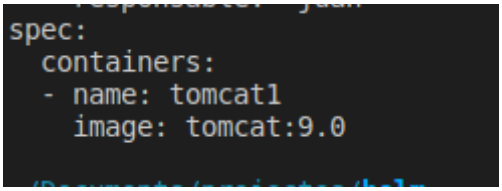
Para evitar el espacio en blanco que ha dejado, provocado porque la línea del if forma parte del manifiesto aunque no aparezca,



debemos poner un guión al inicio de las llaves del if. De esta manera



aparece todo junto



6.7. - AND y OR

Ejemplo de sintaxis:

```
{{- if and  (eq .Values.entorno  "desarrollo") ( eq .Values.departamento "RRHH" )  }}
  image: httpd:2.4
  {{ else}}
  image: httpd:2.2
 {{ end }}
```

«and» va justo después del «if» y para indicar «igual que» se añade «eq» antes de cada valor.

6.8. - Bucle

Con el operador «range» podremos iterar en un array. Por ejemplo, el array en values.yaml

```
departamentos:
  - rrhh
  - sales
  - TI
  - Marketing
```

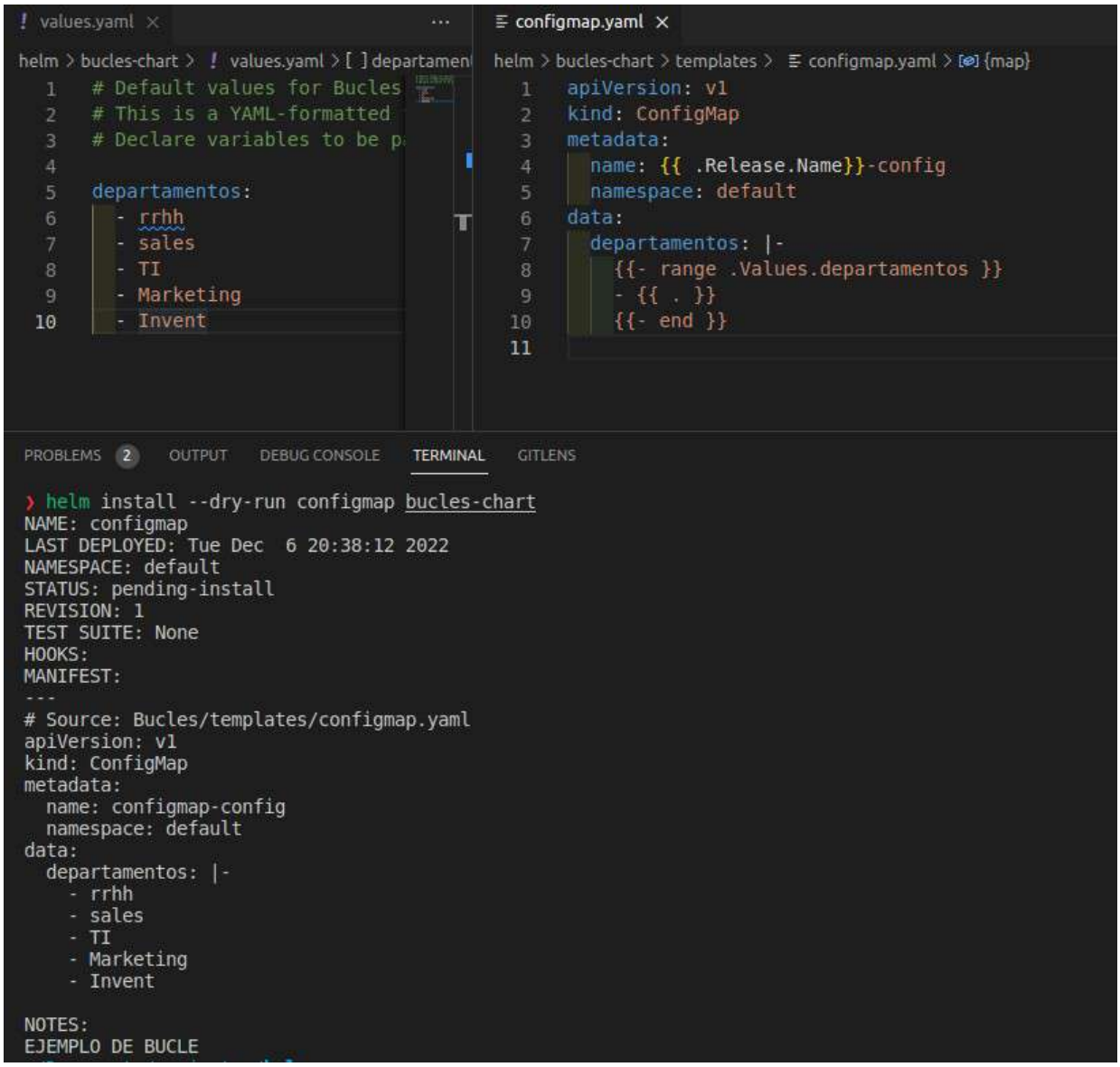
Lo podemos extraer con

```
data:
  departamentos: |-
    {{- range .Values.departamentos }}
    - {{ . }}
    {{- end }}
```

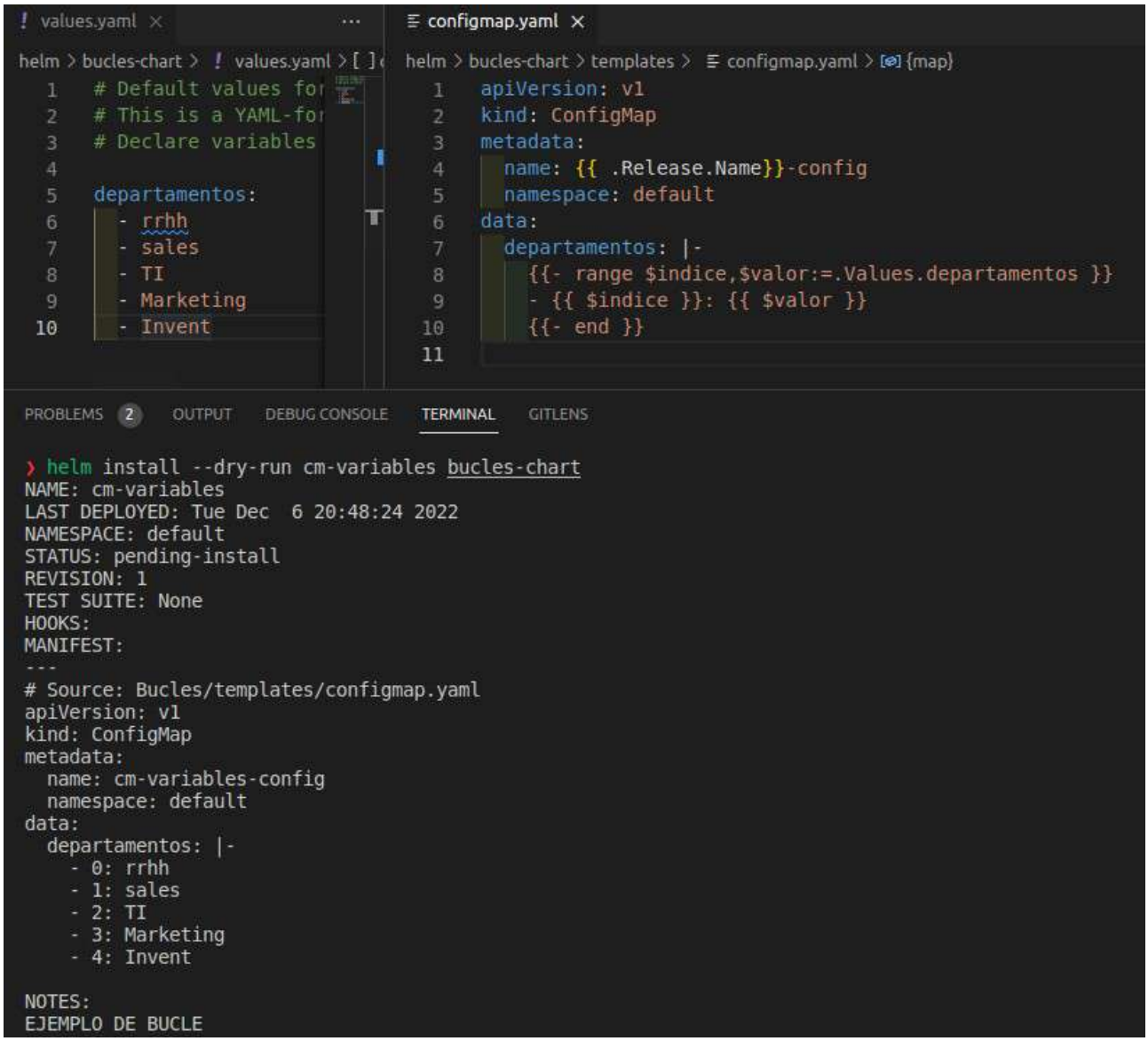
|- Esta clausula indica que tendrá más de un valor.

range Es un operador que en su forma más básica nos permite hacer un bucle, reiterar.

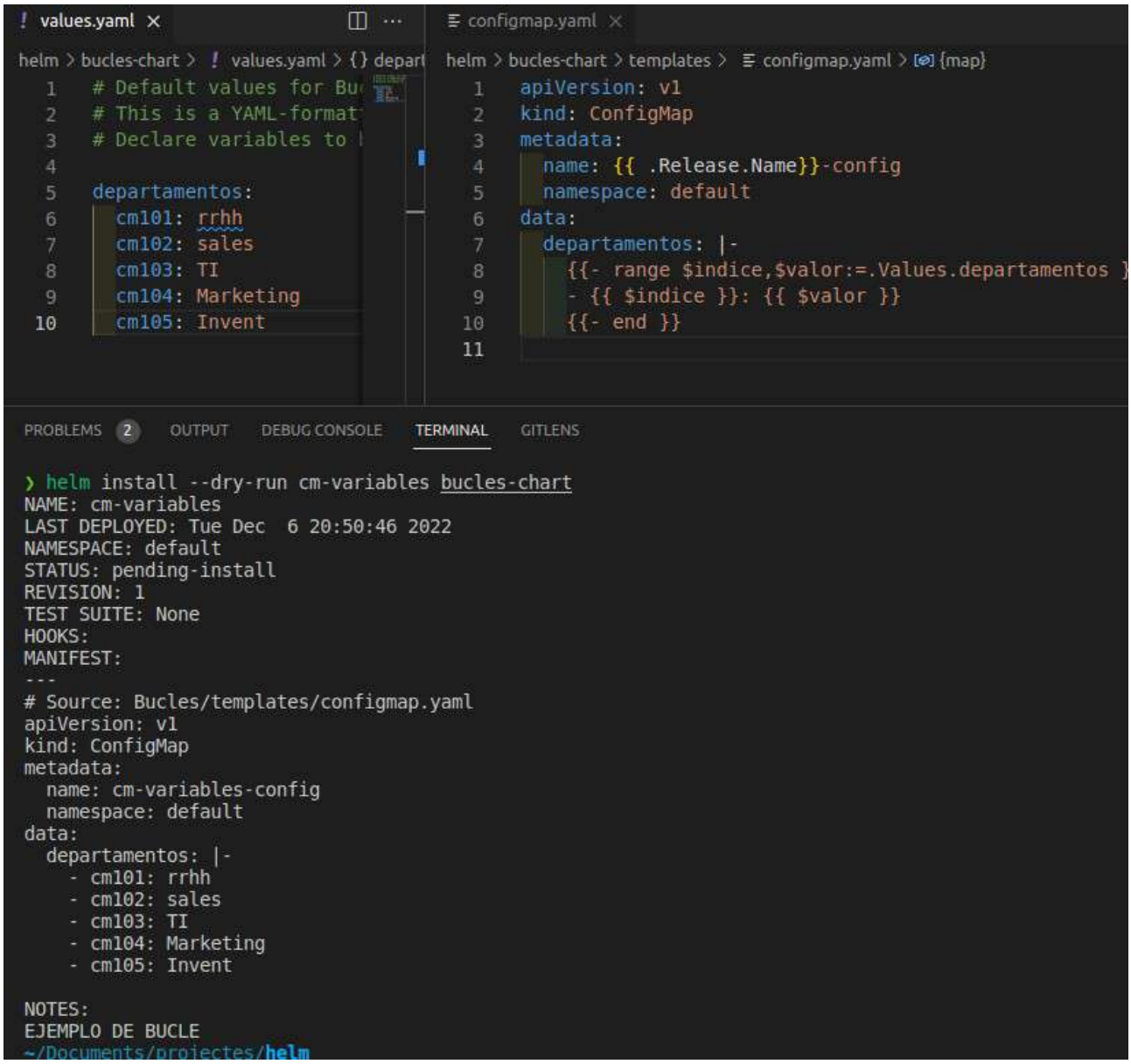
. El punto hace referencia al objeto que va cogiendo del array.



Podemos utilizar variables como receptores y como índices de un bucle de tipo range. Será igual que el anterior pero modificando el range para que coja el indice del array y el valor:

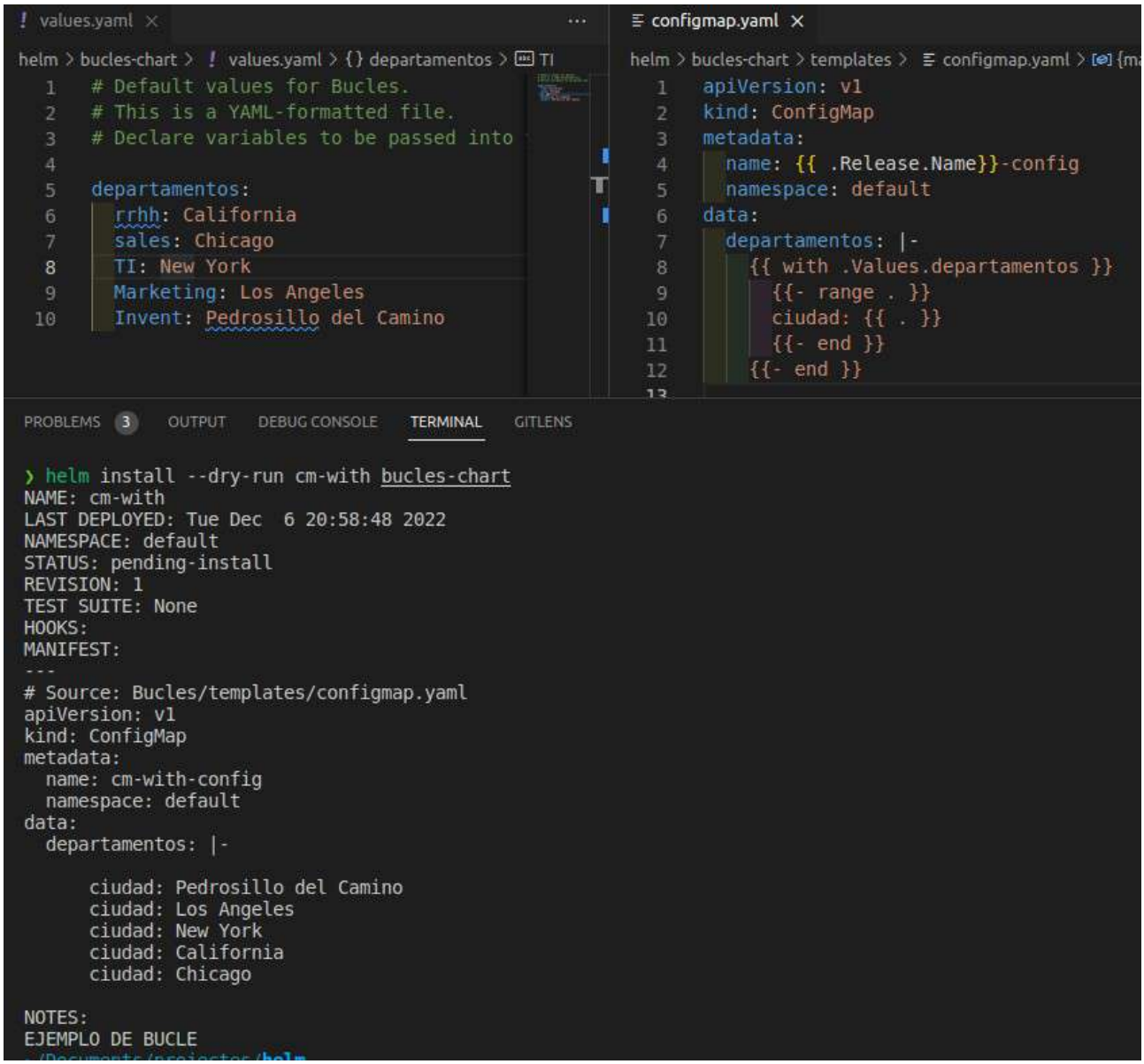


En values.yaml podríamos concretar el índice (key=value) para que la cogiera la variable en el configmap:

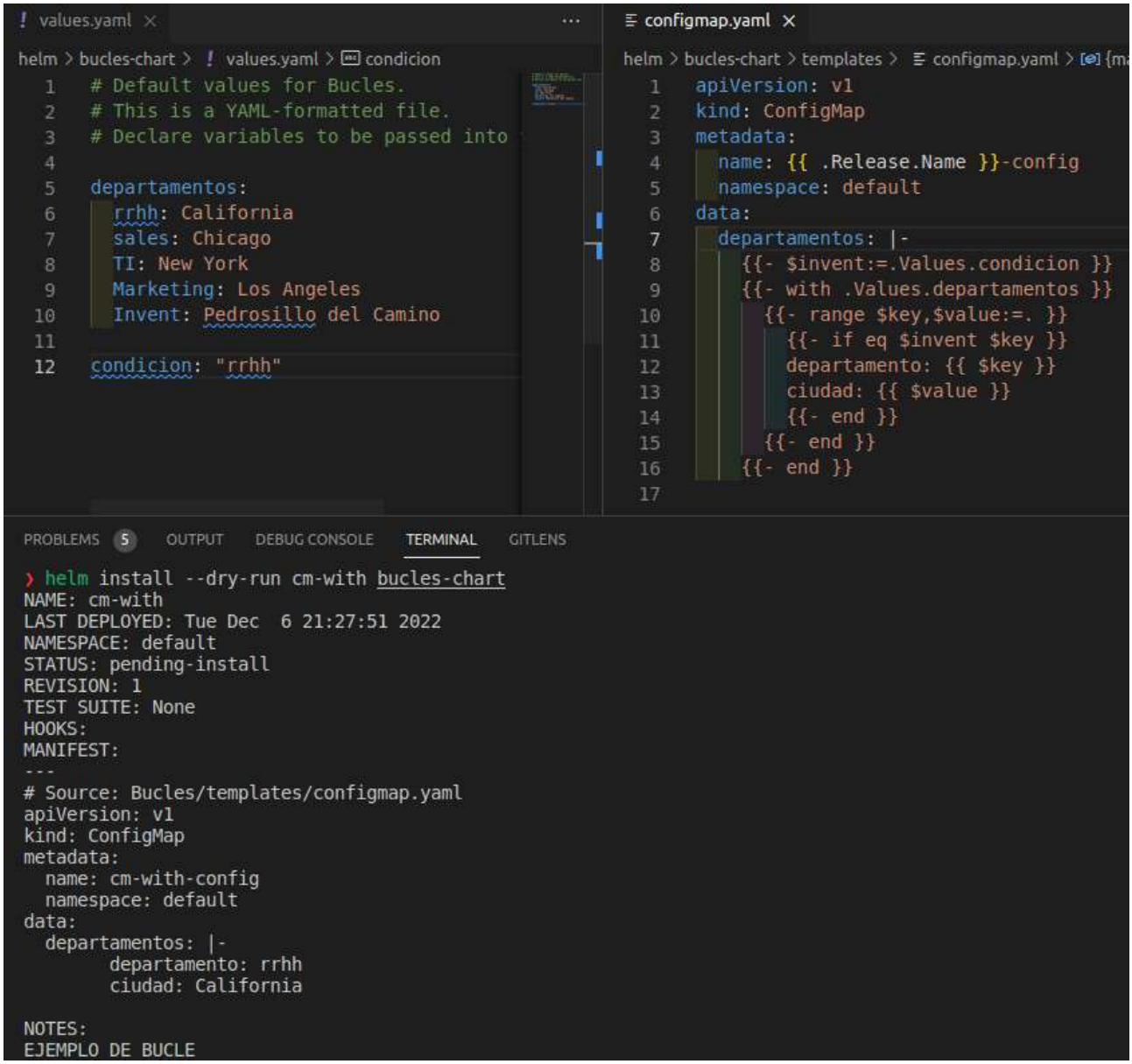


6.9. - WITH

Con la clausula WITH podemos escoger un ámbito dentro de las plantillas. Por ejemplo, podemos marcar el ámbito donde iterará el range anterior.



Para complicarlo más, vamos a declarar una variable con un nuevo valor en values.yaml donde escogeremos uno de los departamentos y, dentro del bucle range, daremos la condición de que se cumpla esa variable para que lo muestre.



TEMA 7 - Funciones y pipelines

Con las funciones podremos personalizar nuestros charts al detalle. Documentación: https://helm.sh/docs/chart_template_guide/function_list/

Los pipelines son identicos a los que podemos utilizar en un entorno como Linux. Documentación: https://helm.sh/docs/chart_template_guide/functions_and_pipelines/

7.1. - Ejemplos de funciones

Vamos a jugar con el valor

city: ampolla

y utilizando de nuevo el configmap veremos algunas funciones:

```
data:
  quote: {{ quote .Values.city }}
  upper: {{ upper .Values.city }}
  title: {{ title .Values.city }}
  shuffle: {{ shuffle .Values.city }}
  now: {{ now }}
  substr: {{ substr 2 7 .Values.city }}
  network: {{ getHostByName "curso" }}
  network: {{ getHostByName "www.google.com" }}
```

```
❯ helm install --dry-run functions Funciones
NAME: functions
LAST DEPLOYED: Tue Dec  6 22:01:58 2022
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
MANIFEST:
---
# Source: Funciones/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: functions-config
  namespace: default
data:
  quote: "ampolla"
  upper: AMPOLLA
  title: Ampolla
  shuffle: laaplom
  now: 2022-12-06 22:01:58.599311241 +0100 CET m=+0.067241579
  substr: polla
  network: 10.0.151.12
  network: 216.58.215.164

NOTES:
EJEMPLO DE FUNCIONES
```

En network tenemos el nombre de una máquina y una dirección web.

Podemos dar formato a la fecha con date
https://www.pauladamsmith.com/blog/2011/05/go_time.html

now: {{ now | date "Mon, 2 Jan 2006 15:04:05" }}

Quedando así:

now: Tue, 6 Dec 2022 22:08:36

7.2. - Ejemplo de Pipelines

Con el now ya hemos hecho una pipeline pero podemos transformar otras de ejemplo:

data:
quote: {{ quote .Values.city | upper }}
upper: {{ upper .Values.city | repeat 3 }}
now: {{ now | htmlDate }}
network: {{ getHostByName "victus" | substr 0 3}}

```
❯ helm install --dry-run pipelines Pipelines
NAME: pipelines
LAST DEPLOYED: Tue Dec  6 23:10:51 2022
NAMESPACE: default
STATUS: pending-install
REVISION: 1
TEST SUITE: None
HOOKS:
MANIFEST:
---
# Source: Pipelines/templates/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: pipelines-config
  namespace: default
data:
  quote: "CALIFORNIA"
  upper: CALIFORNIACALIFORNIACALIFORNIA
  now: 2022-12-06
  network: 192

NOTES:
EJEMPLO DE PIPELINES
```

TEMA 8 - SubPlantillas. Partials

Podemos guardar en un fichero bloques completos que podemos reutilizar en otras templates.

Cuando creamos un nuevo chart con create estamos creando una subplantilla.

Creamos una plantilla

```
helm create Plantilla1
```

Y observamos el árbol

❯ tree Plantilla1

Plantilla1

```
├── charts
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml
```

3 directories, 10 files

En el fichero _helpers.tpl podemos poner subplantillas, bloques o componentes que se podrán utilizar en otros ficheros.

El nombre debería empezar con un Guión bajo _ y acabar con la extensión .tpl.

Cuando se crea un .tpl estamos diciendo que este fichero no debe ir a Kubernetes. Es un fichero de ayuda con subplantillas para tareas más livianas. Solo se utiliza para el proceso de trabajo.

Para crear un bloque se utiliza la clausula define. Por ejemplo:

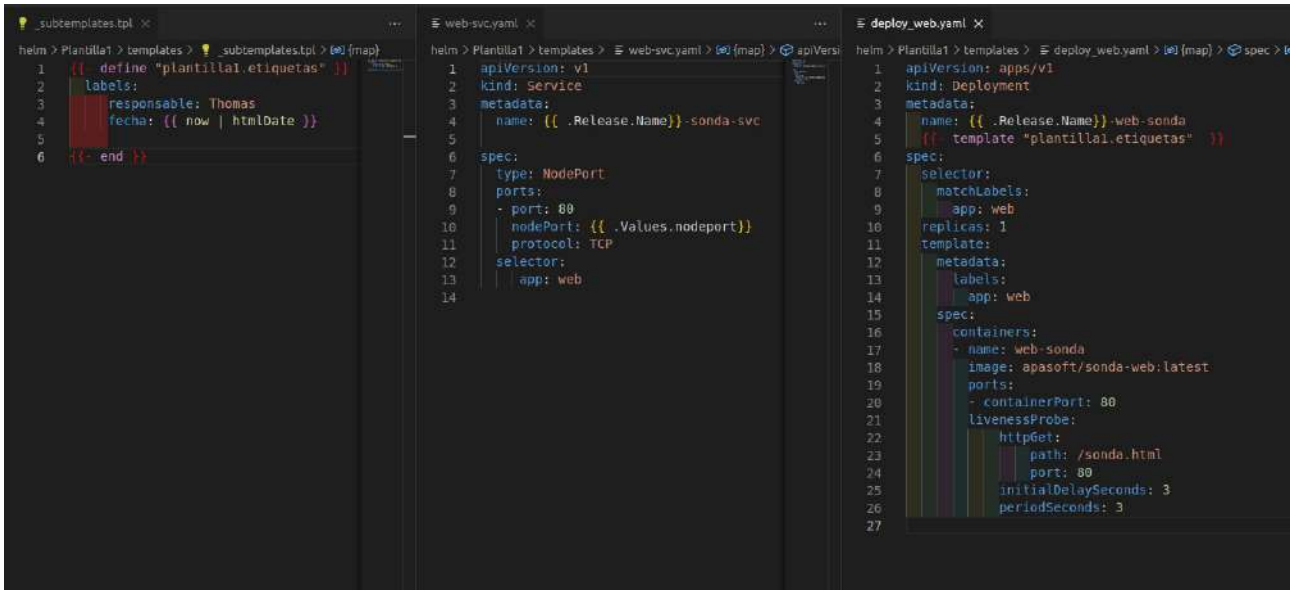
```
{{- define "Plantilla1.name" -}}
{{- default .Chart.Name .Values.nameOverride | trunc 63 | trimSuffix "-" }}
{{- end }}
```

Con el valor que se de a define lo podremos cargar donde queramos.

8.1. - Ejemplo de subplantilla

Es buena política llamarle en «define» con el nombre de la plantilla donde pertenece.

```
{{- define "plantilla1.etiquetas" }}
  labels:
    responsable: Thomas
    fecha: {{ now | htmlDate }}
{{- end }}
```



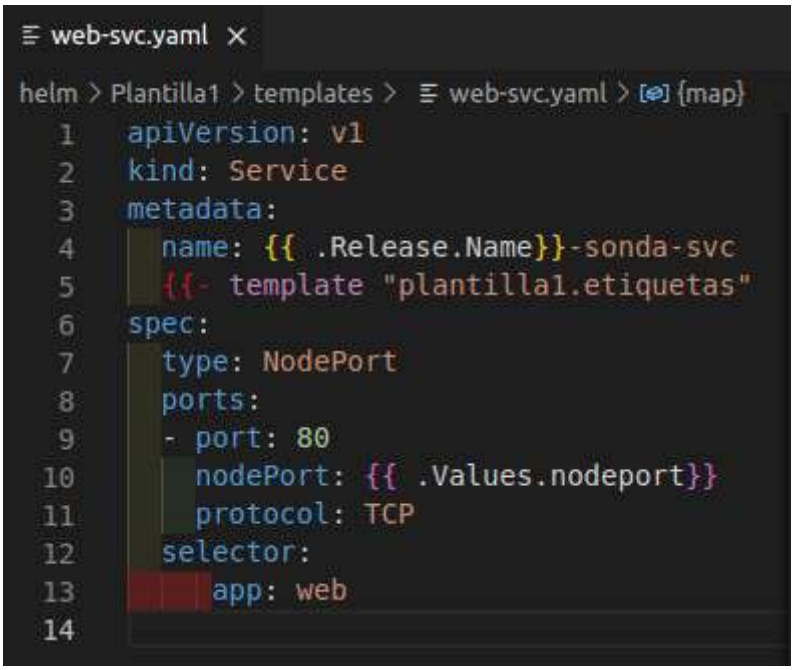
Hacemos una subplantilla con las labels que incluimos en el deploy con la función «template»:

```
---
# Source: Plantilla2/templates/deploy_web.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: plantilla1-web-sonda
  labels:
     responsable: Thomas
     fecha: 2022-12-06
spec:
  selector:
    matchLabels:
      app: web
  replicas: 1
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - name: web-sonda
        image: apasoft/sonda-web:latest
        ports:
        - containerPort: 80
        livenessProbe:
            httpGet:
                path: /sonda.html
                port: 80
            initialDelaySeconds: 3
            periodSeconds: 3

NOTES:
EJEMPLO DE SUBPLANTILLA
```

El mayor problema que dan las plantillas es el identado, porque con un error de un espacio más o menos ya no nos dejará instalar la Chart.

El template también lo podemos pasar al servicio creado y así lo tendríamos igual.

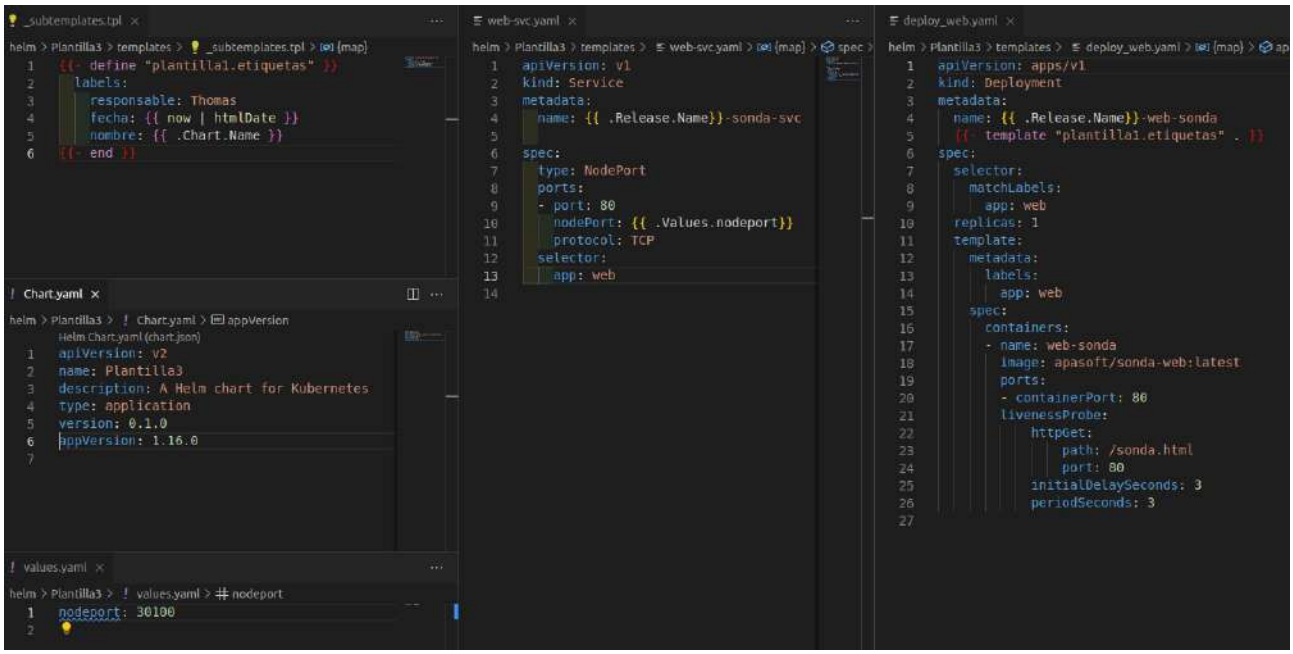


Si lo probamos lo podemos ver.

```
---
# Source: Plantilla2/templates/web-svc.yaml
apiVersion: v1
kind: Service
metadata:
  name: plantilla1-sonda-svc
  labels:
     responsable: Thomas
     fecha: 2022-12-06
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30100
    protocol: TCP
  selector:
     app: web
```

8.2. - Contexto de las subplantillas

Para que una subtemplate puede recoger valores del resto de fichero le debemos indicar el contexto. Por ejemplo, para que pueda recoger el nombre del chart



Debemos indicarle en la clausula «template» donde es el contexto. En este caso con un punto . Le indicariamos la raíz.

Es decir, el segundo argumento en la clausula «template» es el contexto.