Migración de una aplicación a Kubernetes

Israel Pavón Maculet Sergio Arroutbi Braojos

18 de marzo de 2019

Índice

1 Introducción **2**

2 Descripción del despliegue en Kubernetes **3**
2.1 Despliegue 3
2.1.1 Vista física 3
2.1.2 Vista de despliegue 4
2.1.3 Vista a nivel de servicio 5
2.1.4 Vista de datos 5
2.2 Helm 8
2.2.1 templates 9
2.2.2 values.yaml 14
2.2.3 Chart.yaml 16
2.3 Docker 16

3 Instrucciones para el despliegue de la aplicación **21**
3.1 Instrucciones paso a paso 21
3.2 Descripción de herramientas software utilizadas 24

Índice de figuras

1 Vista física 3
2 Vista de Despliegue 4
3 Vista a nivel de servicio 5
4 Vista de datos: datos persistentes 6
5 Vista de datos: acceso a base de datos 7
6 Vista de datos: gestión de datos en memoria compartida 8
7 Alerta sobre conexión no segura 22
8 Confirmar Excepción de Seguridad 23
9 Página de inicio 24

1. Introducción

Para esta práctica se ha utilizado, como aplicación distribuida de base para la práctica, el proyecto conocido como "Guardarpunto". Esta aplicación esta disponible en el siguiente enlace de Github:

https://github.com/mfms5/guardarpunto

En cuanto a realizar la implementación de los diversos ficheros que permitan el despliegue de dicha aplicación en un entorno basado en el sistema de orquestación de contenedores Kubernetes, se ha optado por realizar un "fork" del proyecto anterior en el siguiente repositorio de "github":

https://github.com/sarroutbi/guardarpunto

El repositorio anterior, básicamente, contiene una serie de carpetas adicionales respecto al proyecto original que se enumeran a continuación:

1. **docker** : Esta carpeta contiene los ficheros Dockerfile que permiten la construcción de los distintos contenedores que sustituyen a las máquinas virtuales del proyeto original.
2. **helm** : Bajo este directorio aparecen los ficheros necesarios para realizar el despliegue basado en **helm**, el gestor de paquetes de Kubernetes que permite una mejor automatización, reuso, gestión y versionado de los ficheros de despliegue de Kubernetes.
3. **doc** : Contiene los ficheros necesarios para la elaboración de la presente documentación.

En cuanto a las tecnologías utilizadas, el despliegue se ha basado, como ya se comentó anteriormente, en **helm** como gestor de paquetes de Kubernetes. Por otra parte, como sistema base Kubernetes se ha optado por **minikube**.

En la sección 2 se realizará una descripción detallada del despliegue realizado, así como de los pasos seguidos para generar los contenedores apropiados que sustituyan a las máquinas virtuales originales de la aplicación ("dockerización") así como los ficheros helm que permiten el despliegue y orquestación de dichos contenedores de aplicación en la infraestructura Kubernetes.

Mientras, la sección 3, enumerará los diversos pasos que se han de seguir para el despliegue de la aplicación. De igual forma, se enumerarán las principales herramientas software utilizadas, así como sus versiones y las diversas opciones y plugins o addons que se han habilitado.

2. Descripción del despliegue en Kubernetes

2.1. Despliegue

Esta sección describe las características del despliegue de Kubernetes llevado a cabo en el desarrollo de la práctica. A través de diversas vistas en modo de diagramas, se detallarán los servicios, despliegues pods y reglas de Ingress que se han decidido implementar para el correcto despliegue de la aplicación.

2.1.1. Vista física

Esta vista "física" pretende representar los containers que estarán corriendo realmente en el cluster una vez la aplicación distribuída haya sido desplegada. Será la herramienta de orquestación Kubernetes la encargada de que siempre estén corriendo estos 5 pods, cada uno de ellos formado por un container, por simplicidad.
En un despliegue con minikube, todos los containers estarán corriendo en el mismo host, pero en un cluster real de Kubernetes, formado por más hosts, cada pod podría estar corriendo en cualquier host de forma que incluso podrían ser movidos de un host a otro dependiendo de las circunstancias. Por ejemplo, las dos instancias del servicio "internalsvc" podrían estar corriendo en distintos hosts cada una.
Kubernetes es el encargado de garantizar la conectividad entre todos los pods, basada en nombres de servicio que serán accesibles a través del servicio de DNS que Kubernetes ofrece, por lo que el código de las aplicaciones ha debido ser modificado para que no haga referencia a IPs, sino a nombres de dominios, dado que las IPs efectivas que tienen asociados los distintos containers, pueden considerarse efímeras, ya que pueden cambiar cada vez que Kubernetes redespliega los containers, o bien cuando se realice una migración de pods, etc.

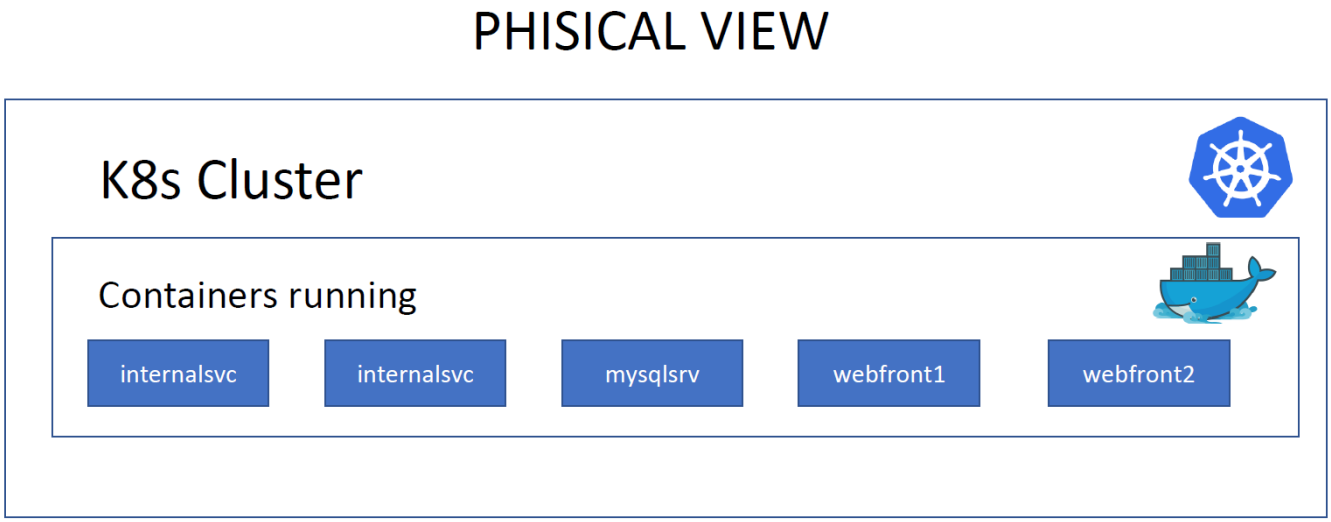

Figura 1: Vista física

2.1.2. Vista de despliegue

Con esta vista de despliegue se pretende describir la composición de servicios, deployments y pods que se ha seguido a la hora de desplegar la aplicación distribuída en Kubernetes. De esta forma, se puede observar, de forma anidada, los distintos elementos existentes en el cluster a nivel de servicio, despliegue y pod, con las réplicas existentes para cada uno de estos últimos:

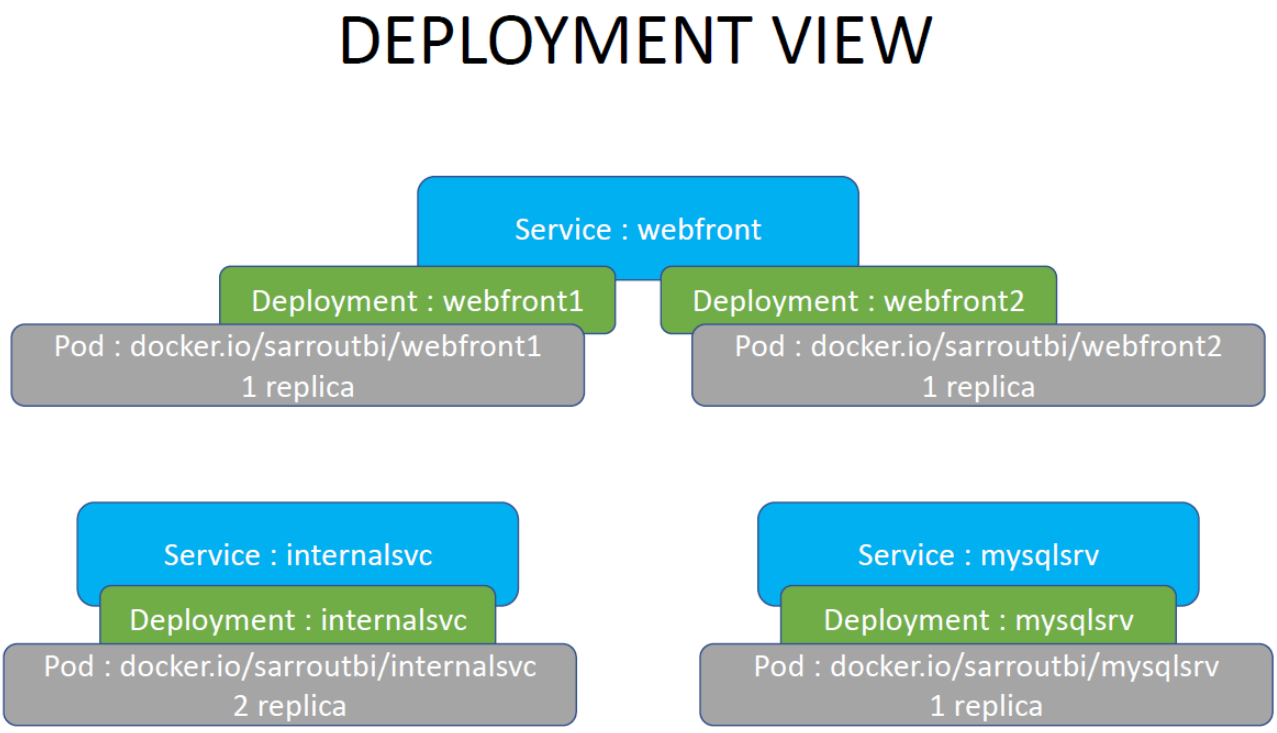

Figura 2: Vista de Despliegue

Como puede observarse, existen tres servicios:

- **mysqlsrv**: Es un servicio que ofrece acceso a BBDD. Tiene asociado un deployment, que configura una réplica de una imagen Docker con una base de datos MySql dockerizada. Es un servicio del tipo ClusterIP, con lo que sólo es accesible dentro del cluster Kubernetes.
- **internalsvc**: Es un servicio interno con una API REST, que implementa el envío de un mensaje de bienvenida ante el registro de un nuevo usuario. Tiene asociado un deployment que configura 2 replicas de una imagen Docker con la aplicación Java "MailRestPost" dockerizada. Es un servicio del tipo ClusterIP, con lo que sólo es accesible dentro del cluster Kubernetes.
- **webfront**: Es el servicio que ofrece la NBI de la aplicación Helm/Kubernetes. Para que pueda tener acceso fuera del cluster, se define como un servicio del tipo LoadBalancer, al que se le ha asociado un ingress, para que a través de el, las peticiones puedan llegar al servicio, que a su vez balanceará entre los dos deployments que tiene asociado. Cada deployment configura 1 réplica de una imagen Docker con la aplicación Java "guardar_punto" dockerizada. La imagen de cada deployment varía únicamente en el usuario que accederá a la BBDD y en que una tiene el flag de creación de creación del schema de base de datos para "Hibernate".

2.1.3. Vista a nivel de servicio

Esta vista describe la configuración realizada a nivel de servicio para permitir las comunicaciones internas de los diversos servicios implementados como, también, las reglas necesarias para acceder de forma externa al cluster, a través del mecanismo conocido como Ingress. La solución elegida para esto último ha sido nginx:

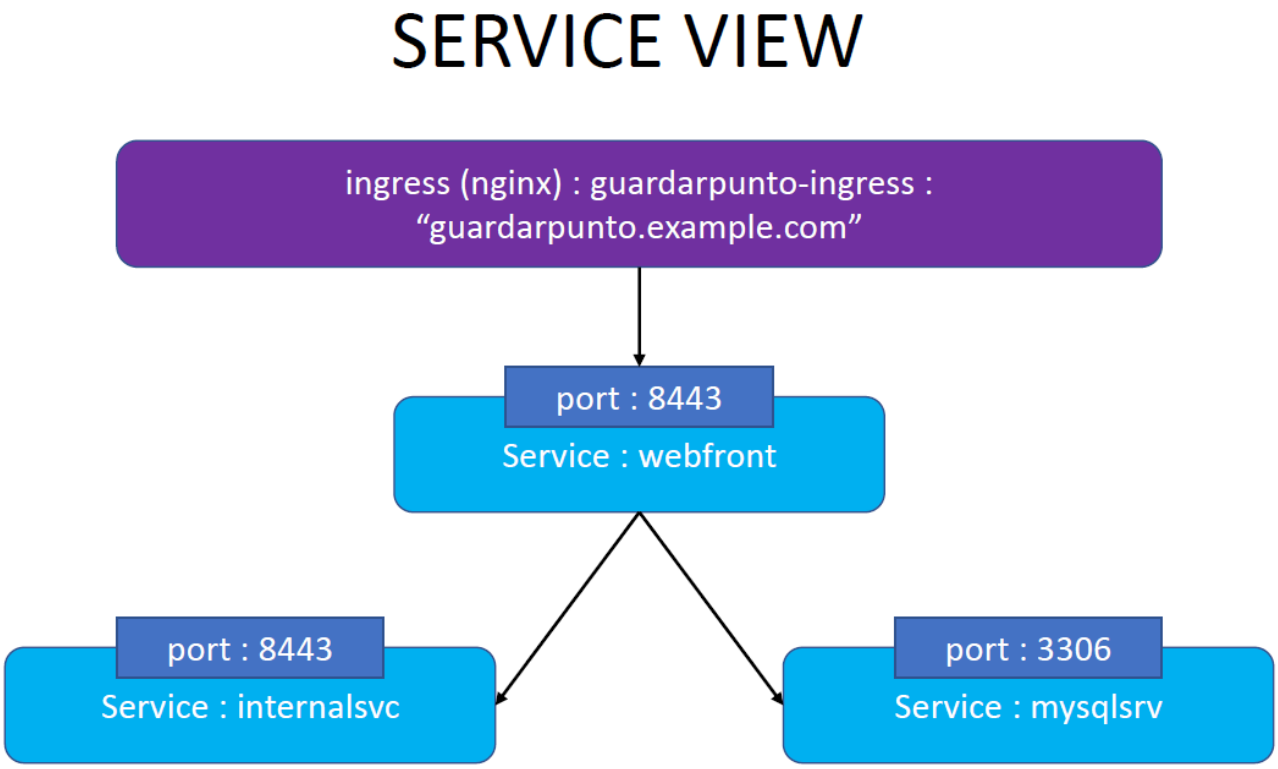

Figura 3: Vista a nivel de servicio

Como puede observarse, los puertos expuestos por cada servicio son los siguientes:

- **mysqlsrv**: Puerto **3306**, puerto standard de la base de datos Mysql.
- **internalsvc**: Puerto **8443**, puerto standard de arranque de Spring Boot con uso de HTTPS.
- **webfront**: Puerto **8443**, puerto standard de arranque de Spring Boot con uso de HTTPS.

2.1.4. Vista de datos

Este apartado describe la aproximación llevada a cabo para el despliegue de la base de datos y el acceso, por parte de las aplicaciones que actúan como frontales Web, a los mismos. Se presenta, a continuación, la declaración del volumen persistente de datos que se ha llevado a cabo:

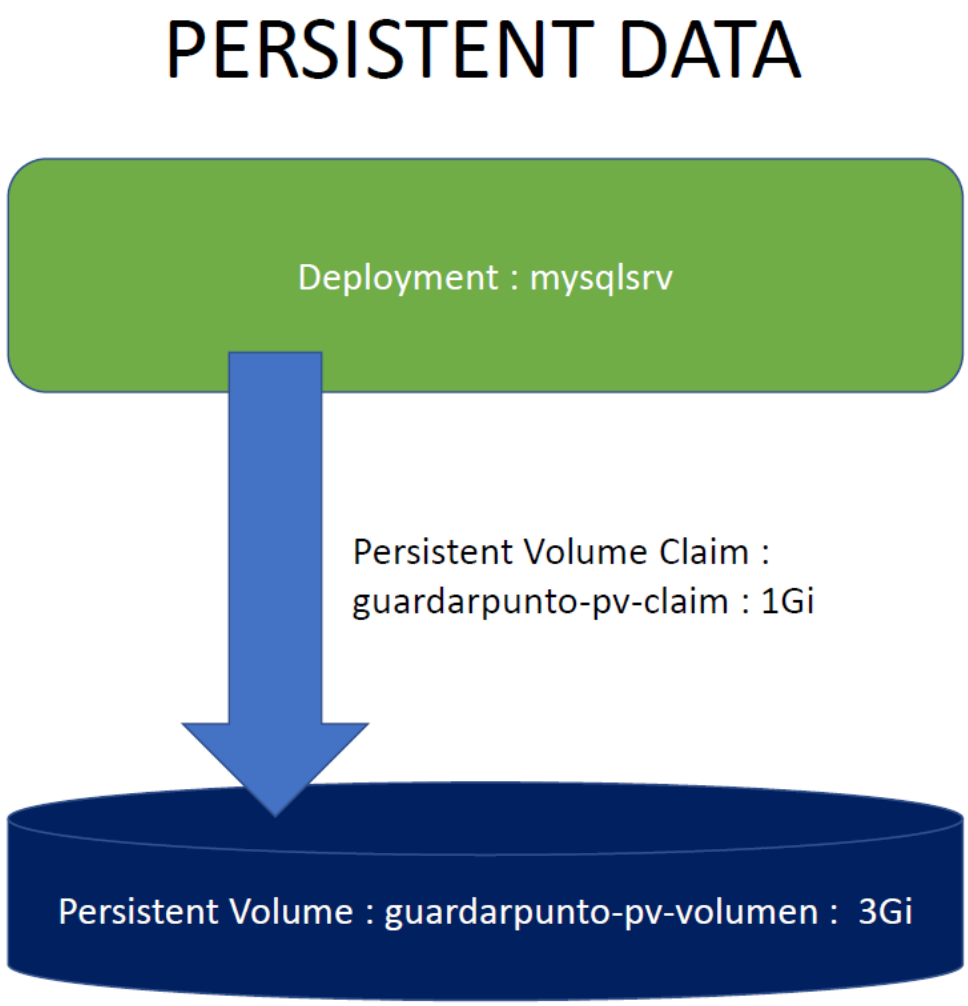

Figura 4: Vista de datos: datos persistentes

Como puede observarse en la Figura 4, se declara un "persistent volume" de 3 Gigas, que permite tener datos persistentes en el cluster. El servicio de base de datos (mysqlsrv) será quien haga uso de este sistema de persistencia. Para ello, en su deployment, se debe declarar un PVC, "persistent volume claim", que será utilizado por el deployment "mysqlsrv" para los datos. De este modo, el contenido de la base de datos es persistente, y puede ser recuperado en el caso de que Kubernetes requiera el redespliegue del pod correspondiente, que en este caso es el container con la aplicación mysql asociado al servicio "mysqlsrv".

Mientras, a nivel de acceso a la base de datos, en este proyecto, existe un servicio web que balanceará a entre dos aplicaciones idénticas que tan sólo difieren en el usuario que accede a la BBDD, única y exclusivamente por adaptación a la práctica original. En esta adaptación a Kubernetes, se ha optado por un esquema de un servicio de tipo "Load Balancer" con dos deployments.
Cada deployment tiene configurado un pod con un único contenedor Docker que difiere de la imagen del otro frontal web en el usuario de acceso a Mysql. Aparte de la diferencia del usuario de acceso a la BBDD, en una de las imágenes se configura un flag de Spring Boot para que Hibernate haga un "create" del schema en la base de datos final ("–spring.jpa.hibernate.ddl-auto=create"). Por tanto, a nivel de acceso a la base datos, la configuración queda como se muestra en la Figura 5:

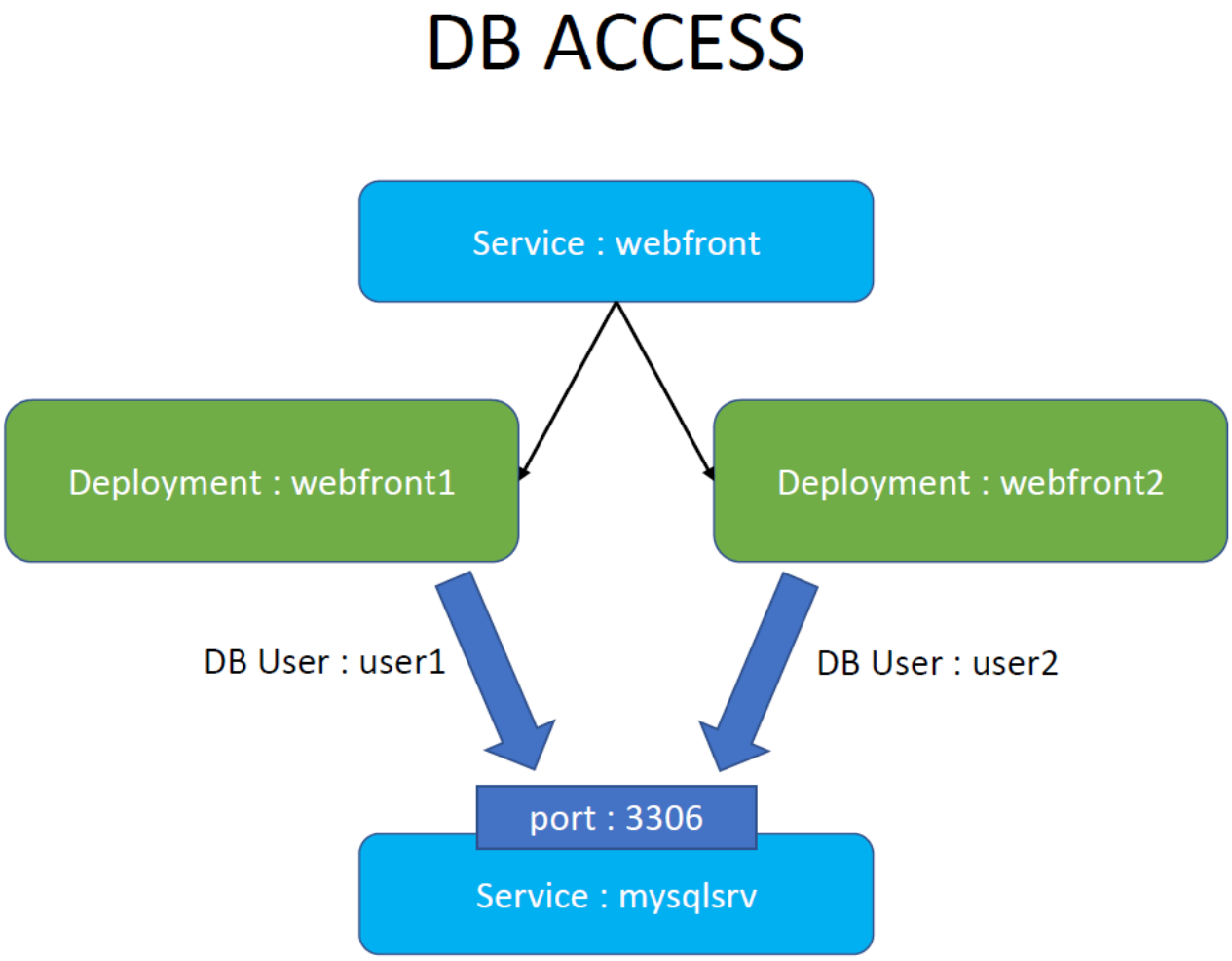

Figura 5: Vista de datos: acceso a base de datos

Para que ambos deployments puedan procesar peticiones balanceadas de cualquier sesión, la aplicación hace uso de un Grid de memoria Hazelcast. Para ello, hay que configurar el puerto hazelcast para que el Grid tenga conectividad entre las distintas instancias, según se muestra en la Figura 6:

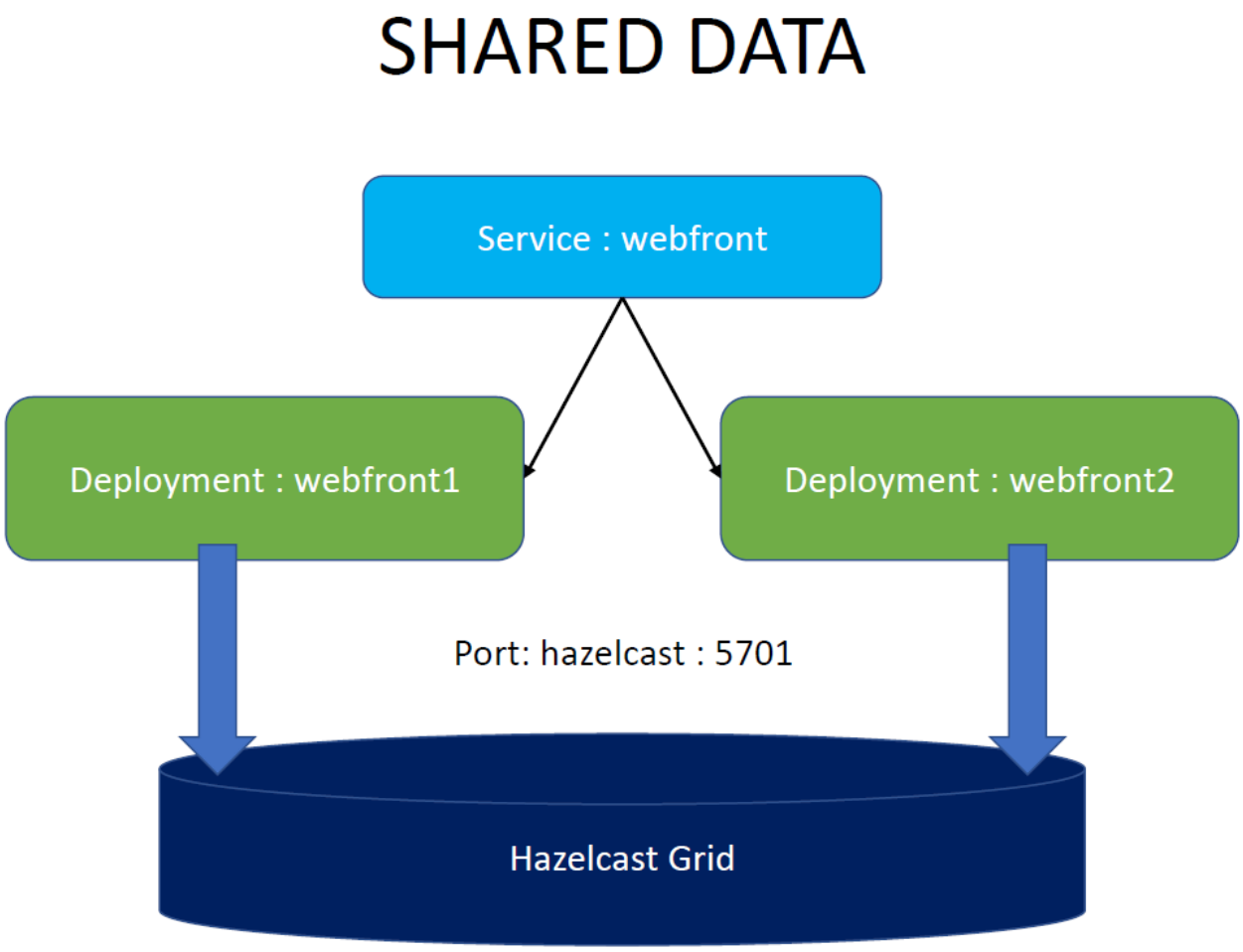

Figura 6: Vista de datos: gestión de datos en memoria compartida

En las siguientes subsecciones se describirán las implementaciones necesarias para llevar a cabo la materialización del diseño del despliegue anterior. Por un lado, la sección 2.2 presentará la especificación helm (tanto a nivel de templates como de valores) llevada a cabo para el despliegue, mientras que la sección 2.3 recoge los mecanismos que se han seguido para conseguir la "contenerización" de las distintas imágenes de la aplicación necesarias para sustituir las máquinas virtuales diseñadas en la práctica original.

2.2. Helm

Se recogen en este apartado los distintos ficheros "Helm" implementados para materializar el diseño de la aplicación distribuída migrada a Kubernetes según lo descrito en la sección 2. La selección de esta herramienta ha sido básicamente las facilidades que ofrece ya que permite gestionar de forma simplificada aplicaciones de Kubernetes.
Mediante el uso de "Charts" se facilita la definión, instalación, gestión y actualización de cualquier tipo de aplicación Kuberentes, independientemente de la complejidad inherente de la misma. A su vez, los "Charts" son un tipo de ficheros que resultan sencillos a la hora de ser creados, versionados, compartidos y publicados, y que permiten evitar el efecto "copy & paste" por el reuso que permiten.

En lo relativo a esta aplicación particular, se ha definido la estructura helm como sigue:

```
./guardarpunto
./guardarpunto/templates
./guardarpunto/templates/guardapunto.yaml
./guardarpunto/templates/ingress.yaml
./guardarpunto/values.yaml
./guardarpunto/Chart.yaml
```

No se va a realizar una descripción muy extensa de los ficheros helm implementados, más allá la enumeración de cada uno de las carpetas con los ficheros existentes. La justificación de la utilización en la práctica de cada uno de los deployments, servicios, ingress, etc. se ha realizado anteriormente en la sección 2.

2.2.1. templates

Por simplicidad, no se ha considerado necesario dividir en distintos ficheros los diversos elementos del despliegue de Kubernetes. Por ello, básicamente, se han definido dos ficheros, uno para la aplicación distribuída completa y otro para definir las reglas de ingreso en el cluster:

- **guardapunto.yaml**: Es el esqueleto de la aplicación, con la definición de los Pods, Deployments y Servicios según lo descrito en la sección 2. El contenido del fichero se muestra a continuación:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: {{ .Values.pv.pvolume.name }}
  labels:
    type: {{ .Values.pv.pvolume.type }}
spec:
  storageClassName: {{ .Values.pv.pvolume.storageClassName }}
  capacity:
    storage: {{ .Values.pv.pvolume.capacity }}
  accessModes:
    - {{ .Values.pv.pvolume.accessMode }}
  hostPath:
    path: {{ .Values.pv.pvolume.hostPath }}
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: {{ .Values.pv.pvolumeclaim.name }}
spec:
  storageClassName: {{ .Values.pv.pvolumeclaim.storageClassName }}
  accessModes:
    - {{ .Values.pv.pvolumeclaim.accessMode }}
  resources:
    requests:
```

```
      storage: {{ .Values.pv.pvolumeclaim.request.storage }}
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Values.service.webfront1.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  replicas: {{ .Values.replicas.webfront1.replicaCount }}
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: {{ .Values.service.webfront.name }}
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: {{ .Values.image.webfront1.repository }}/
               {{ .Values.image.webfront1.name }}:
               {{ .Values.image.webfront1.tag }}
        imagePullPolicy: {{ .Values.image.webfront1.pullPolicy }}
        ports:
        - containerPort: {{ .Values.service.webfront1.port }}
        env:
        - name: LOGLEVEL
          value: {{ .Values.env.logLevel | quote}}
        resources:
{{ toYaml .Values.resources | indent 10 }}
      restartPolicy: Always
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Values.service.webfront2.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  replicas: {{ .Values.replicas.webfront2.replicaCount }}
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
```

```
        app: {{ .Values.service.webfront.name }}
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: {{ .Values.image.webfront2.repository }}/
               {{ .Values.image.webfront2.name }}:
               {{ .Values.image.webfront2.tag }}
        imagePullPolicy: {{ .Values.image.webfront2.pullPolicy }}
        ports:
        - containerPort: {{ .Values.service.webfront2.port }}
        env:
        - name: LOGLEVEL
          value: {{ .Values.env.logLevel | quote}}
        resources:
{{ toYaml .Values.resources | indent 10 }}
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.service.webfront.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  type: {{ .Values.service.webfront.type }}
  ports:
  - name: http-spring
    port: {{ .Values.service.webfront.port }}
  - name: hazelcast
    port: {{ .Values.service.webfront1.hazelcast.port }}
    targetPort: {{ .Values.service.webfront1.hazelcast.port }}
  selector:
    app: {{ .Values.service.webfront.name }}
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Values.service.mysqlsrv.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  replicas: {{ .Values.replicas.mysqlsrv.replicaCount }}
  strategy:
    type: Recreate
  template:
```

```
    metadata:
      labels:
        app: {{ .Values.service.mysqlsrv.name }}
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      volumes:
      - name: {{ .Values.pv.vol.name }}
        persistentVolumeClaim:
          claimName: {{ .Values.pv.pvolumeclaim.name }}
      containers:
      - name: {{ .Chart.Name }}
        image: {{ .Values.image.mysqlsrv.repository }}/
               {{ .Values.image.mysqlsrv.name }}:
               {{ .Values.image.mysqlsrv.tag }}
        imagePullPolicy: {{ .Values.image.mysqlsrv.pullPolicy }}
        ports:
        - containerPort: {{ .Values.service.mysqlsrv.port }}
        volumeMounts:
          - mountPath: {{ .Values.pv.vol.volumeMounts.mountPath }}
            name: {{ .Values.pv.vol.name }}
        env:
        - name: LOGLEVEL
          value: {{ .Values.env.logLevel | quote}}
        resources:
{{ toYaml .Values.resources | indent 10 }}
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.service.mysqlsrv.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  type: {{ .Values.service.mysqlsrv.type }}
  ports:
  - name: mysql-port
    port: {{ .Values.service.mysqlsrv.port }}
    targetPort: {{ .Values.service.mysqlsrv.port }}
  selector:
    app: {{ .Values.service.mysqlsrv.name }}
---
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: {{ .Values.service.internalsvc.name }}
```

```
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  replicas: {{ .Values.replicas.internalsvc.replicaCount }}
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: {{ .Values.service.internalsvc.name }}
      annotations:
        sidecar.istio.io/inject: "false"
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: {{ .Values.image.internalsvc.repository }}/
               {{ .Values.image.internalsvc.name }}:
               {{ .Values.image.internalsvc.tag }}
        imagePullPolicy: {{ .Values.image.internalsvc.pullPolicy }}
        ports:
        - containerPort: {{ .Values.service.internalsvc.port }}
        env:
        - name: LOGLEVEL
          value: {{ .Values.env.logLevel | quote}}
        resources:
{{ toYaml .Values.resources | indent 10 }}
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.service.internalsvc.name }}
  labels:
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
spec:
  type: {{ .Values.service.internalsvc.type }}
  ports:
  - name: http-spring
    port: {{ .Values.service.internalsvc.port }}
  selector:
    app: {{ .Values.service.internalsvc.name }}
```

- **ingress.yaml**: Este fichero contiene las reglas de ingress de tráfico en el cluster Kubernetes para hacer llegar el tráfico a la aplicación.

```
apiVersion: extensions/v1beta1
kind: Ingress
```

```
    metadata:
      name: {{ .Values.ingress.name }}
      annotations:
        kubernetes.io/ingress.class: {{ .Values.ingress.class }}
        nginx.ingress.kubernetes.io/rewrite-target: "/"
        nginx.ingress.kubernetes.io/backend-protocol:
          {{ .Values.ingress.backend.protocol }}
    spec:
      tls:
        - hosts:
          - {{ .Values.ingress.host.name }}
          secretName: {{ .Values.ingress.secret.name }}
      rules:
      - host: {{ .Values.ingress.host.name }}
        http:
          paths:
          - path: {{ .Values.ingress.path }}
            backend:
              serviceName: {{ .Values.service.webfront.name }}
              servicePort: {{ .Values.service.webfront.port }}
```

2.2.2. values.yaml

Este fichero permite definir los valores con los que se van a rellenar los templates anteriores. Se describen dichos valores a continuación:

```
replicas:
  webfront1:
    replicaCount: 1
  webfront2:
    replicaCount: 1
  mysqlsrv:
    replicaCount: 1
  internalsvc:
    replicaCount: 2

resources: {}

image:
  webfront1:
    repository: docker.io/sarroutbi
    name: webfront1
    tag: sergioarroutbibraojos
    pullPolicy: Always
  webfront2:
    repository: docker.io/sarroutbi
    name: webfront2
    tag: sergioarroutbibraojos
```

```
    pullPolicy: Always
  mysqlsrv:
    repository: docker.io/sarroutbi
    name: mysqlsrv
    tag: sergioarroutbibraojos
    pullPolicy: Always
  internalsvc:
    repository: docker.io/sarroutbi
    name: internalsvc
    tag: sergioarroutbibraojos
    pullPolicy: Always

service:
  webfront:
    name: webfront
    type: LoadBalancer
    port: 8443
    hazelcast:
      port: 5701
  webfront1:
    name: webfront1
    type: ClusterIP
    port: 8443
    hazelcast:
      port: 5701
  webfront2:
    name: webfront2
    type: ClusterIP
    port: 8443
    hazelcast:
      port: 5701
  mysqlsrv:
    name: mysqlsrv
    type: ClusterIP
    port: 3306
  internalsvc:
    name: internalsvc
    type: ClusterIP
    port: 8443

ingress:
  class: nginx
  name: guardarpunto-ingress
  backend:
    protocol: HTTPS
  host:
    name: guardarpunto.example.com
```

```
  secret:
    name: guardarpunto-tls-cert
  path: "/*"

pv:
  pvolume:
    name: guardarpunto-pv-volume
    type: local
    storageClassName: standard
    accessMode: ReadWriteOnce
    capacity: 3Gi
    hostPath: /mnt/data
  pvolumeclaim:
    name: guardarpunto-pv-claim
    storageClassName: standard
    accessMode: ReadWriteOnce
    request:
      storage: 1Gi
  vol:
    name: guardarpunto-pv-storage
    volumeMounts:
      mountPath: /mnt/data/
```

2.2.3. Chart.yaml

Fichero que contiene la versión de la aplicación, su nombre, descripción y la versión principal de la API de helm utilizada:

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart that allows deploying guardarpunto\
distributed application
name: guardarpunto
version: 0.1.0
```

2.3. Docker

En cuanto a la generación de los contenedores, se ha creado una carpeta "docker" en la raíz del proyecto descrito anteriormente. Dicha carpeta contiene diversos subdirectorios con los distintos contenedores que se han utilizado como sustitución de las máquinas virtuales del proyecto inicial, y que se enumeran a continuación:

- **webfront1**. Esta carpeta contiene el Dockerfile necesario para, partiendo de una imagen base Ubuntu 14.04 (la utilizada en el proyecto original), instalar el JDK de Java 8 apropiado, clonar el código de la aplicación frontal web y recoger las instrucciones de arranque de la misma. Se ha optado por hacer una aplicación principal con un POD distinto al otro frontal

web por el hecho de que en la aplicación original las opciones de arranque y el usuario a utilizar son distintos respecto a la aplicación secundaria / backup. A continuación se incluye el **"Dockerfile"** utilizado:

```
# Pull base image
FROM ubuntu:14.04

# Variables
ENV REPOSITORY https://github.com/sarroutbi/guardarpunto.git
ENV WEB_APP_PATH guardarpunto/guardar_punto_2/target/
guardar_punto-0.0.1-SNAPSHOT.jar
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle
ENV PATH $PATH:$JAVA_HOME/bin

# Update latest software and install java modules
# software-properties-common needed for add-apt-repository command
RUN sudo apt-get update && \
    sudo apt-get install -y git && \
    sudo apt-get install -y software-properties-common && \
    sudo apt-add-repository ppa:webupd8team/java && \
    sudo add-apt-repository ppa:openjdk-r/ppa && \
    sudo apt-get update && \
    echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" \
    | sudo debconf-set-selections && \
    sudo apt-get install -y oracle-java8-installer && \
    sudo apt-get install -y openjdk-8-jre

# Clone software source and copy bin file to this path
RUN git clone -v ${REPOSITORY} && \
    cp -v ${WEB_APP_PATH} .

# Run web application
# TODO: Parametrize IPs, usernames, passwords, etc
CMD ["java", "-jar",
     "guardarpunto/guardar_punto_2/target/guardar_punto-0.0.1-SNAPSHOT.jar",
     "--spring.datasource.url=jdbc:mysql://mysqlsrv:3306/guardarpuntodb?
verifyServerCertificate=false&useSSL=true",
     "--spring.datasource.username=user1",
     "--spring.datasource.password=enjutomojamuto",
     "--spring.jpa.hibernate.ddl-auto=create"]
```

- **webfront2**. Esta carpeta contiene el Dockerfile necesario para generar una aplicación similar a la anterior, pero con las opciones de arranque cambiadas según estaban en el proyecto original (distinto usuario de conexión a base de datos y distintas opciones de arranque de la aplicación Spring Boot): A continuación se incluye el **"Dockerfile"** utilizado:

```
# Pull base image
```

```
FROM ubuntu:14.04

# Variables
ENV REPOSITORY https://github.com/sarroutbi/guardarpunto.git
ENV WEB_APP_PATH \
  guardarpunto/guardar_punto_2/target/guardar_punto-0.0.1-SNAPSHOT.jar
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle
ENV PATH $PATH:$JAVA_HOME/bin

# Update latest software and install java modules
# software-properties-common needed for add-apt-repository command
RUN sudo apt-get update && \
    sudo apt-get install -y git && \
    sudo apt-get install -y software-properties-common && \
    sudo apt-add-repository ppa:webupd8team/java && \
    sudo add-apt-repository ppa:openjdk-r/ppa && \
    sudo apt-get update && \
    echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" \
| sudo debconf-set-selections && \
    sudo apt-get install -y oracle-java8-installer && \
    sudo apt-get install -y openjdk-8-jre

# Clone software source and copy bin file to this path
RUN git clone -v ${REPOSITORY} && \
    cp -v ${WEB_APP_PATH} .

# Run web application
# TODO: Parametrize IPs, usernames, passwords, etc
CMD ["java", "-jar",
     "guardarpunto/guardar_punto_2/target/guardar_punto-0.0.1-SNAPSHOT.jar",
     "--spring.datasource.url=jdbc:mysql://mysqlsrv:3306/guardarpuntodb?
verifyServerCertificate=false&useSSL=true", "--spring.datasource.username=user2",
     "--spring.datasource.password=enjutomojamuto"]
```

- **mysqlsrv**. Bajo este directorio se encuentra el Dockerfile necesario para generar el contenedor de base de datos, partiendo de una versión de Ubuntu 14.04, de igual forma que en la práctica original, y con las opciones de instalación de la base de datos utilizada ("mysql" en este caso), creación de la base de datos, configuración de usuarios, privilegios de los mismos, tablas necesarias y parámetros de conexión necesarios, a parte de las distintas personalizaciones necesarias para este tipo de aplicaciones, como pueda ser el ajuste de la dirección a la que hacer "bind", exposición de los puertos utilizados por mysql, etc. A continuación se incluye el **"Dockerfile"** utilizado:

```
# Pull base image
FROM ubuntu:14.04

# Update latest software and install java modules
```

```
# software-properties-common needed for add-apt-repository command
RUN sudo apt-get update && \
    sudo rm -fr /var/lib/mysql && \
    sudo mkdir -p /mnt/mysql/data && \
    sudo ln -s /mnt/mysql/data /var/lib/mysql && \
    sudo apt-get -q -y install mysql-server && \
    sudo apt-get install -y software-properties-common && \
    sudo apt-add-repository ppa:openjdk-r/ppa && \
    sudo apt-get update && \
    sudo apt-get install -y openjdk-8-jre && \
    sudo sed -i 's/^bind-address/#bind-address/' /etc/mysql/my.cnf && \
    sudo service mysql start && \
    mysql -u root -e \
     "CREATE DATABASE guardarpuntodb;\
     create user 'user1'@'localhost' identified by 'enjutomojamuto';\
     create user 'user2'@'localhost' identified by 'enjutomojamuto';\
     grant all on guardarpuntodb.* to 'user1'@'localhost'\
identified by 'enjutomojamuto';\
     grant all on guardarpuntodb.* to 'user2'@'localhost'\
identified by 'enjutomojamuto';\
     grant all on guardarpuntodb.* to 'user1'@'%'\
identified by 'enjutomojamuto';\
     grant all on guardarpuntodb.* to 'user2'@'%'\
identified by 'enjutomojamuto';\
     flush privileges;"

COPY start_mysql.sh /usr/bin

# Start with just a shell, mysql will be running appropriately
EXPOSE 3306 33060
CMD ["/usr/bin/start_mysql.sh"]
```

Como puede verse anteriormente, además del fichero "Dockerfile", bajo esta carpeta se encuentra también un sencillo script "start_mysql.sh", que realiza, básicamente, dos operaciones:

- Copiar, en primer arranque, los ficheros de base de datos al directorio definido para almacenamiento permanente a través de PVC.
- Arrancar la base de datos con las opciones necesarias en cuanto a resolución de nombres y directorio de datos utilizado.

A continuación se muestra el contenido del script de arranque de la base de datos, "start_mysql.sh":

```
#!/bin/bash
# 1 - Copy data to pvc disk if it is the first time
test -f /mnt/data/copied
if [ $? -eq 1 ];
then
  cp -arfv /var/lib/mysql/ /mnt/data/
```

```
  touch /mnt/data/copied
fi

# 2 - Start database
mysqld --skip-name-resolve --datadir=/mnt/data/mysql/
```

- **internalsvc**. Finalmente, un directorio que engloba el fichero "Dockerfile" necesario para generar el contenedor utilizado para la aplicación que sirve como servicio interno. En este caso, se trata de un servicio de mail que permite el envío de correos en el proceso de registro. Partiendo de una imagen base Ubuntu 14.04 (la utilizada en el proyecto original), instalar el JDK de Java 8 apropiado, clonar el código de la aplicación interna y recoger las instrucciones de arranque de la misma. A continuación se incluye el **"Dockerfile"** utilizado:

```
# Pull base image
FROM ubuntu:14.04

# Variables
ENV REPOSITORY https://github.com/sarroutbi/guardarpunto.git
ENV INTERNAL_SRVC_PATH
    ./guardarpunto/MailRestPost/target/MailRestPost-0.0.1-SNAPSHOT.jar
ENV JAVA_HOME /usr/lib/jvm/java-8-oracle
ENV PATH $PATH:$JAVA_HOME/bin

# Update latest software and install java modules
# software-properties-common needed for add-apt-repository command
RUN sudo apt-get update && \
    sudo apt-get install -y git && \
    sudo apt-get install -y software-properties-common && \
    sudo apt-add-repository ppa:webupd8team/java && \
    sudo add-apt-repository ppa:openjdk-r/ppa && \
    sudo apt-get update && \
    echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" \
| sudo debconf-set-selections && \
    sudo apt-get install -y oracle-java8-installer && \
    sudo apt-get install -y openjdk-8-jre

# Clone software source
RUN git clone -v ${REPOSITORY} && pwd && \
    cp -v ${INTERNAL_SRVC_PATH} .

# Run web application
# TODO: Substitue ile with INTERNAL_SRVC_PATH env variable
CMD ["java", "-jar",
     "./guardarpunto/MailRestPost/target/MailRestPost-0.0.1-SNAPSHOT.jar"]
```

3. Instrucciones para el despliegue de la aplicación

3.1. Instrucciones paso a paso

- **Configuración del /etc/hosts**. Con motivo de acceder a la aplicación distribuida de forma externa a través de nginx, se ha de configurar una dirección de host, en concreto "guadarpunto.example.com" en el "/etc/hosts" de la máquina, configurando la IP de minikube como IP de destino. para ello:

  - Consultar la IP de minikube:

    ```
    $ minikube ip
    192.168.99.100
    ```

  - Editar el fichero "/etc/hosts" para que contenga el nombre de dominio apropiado:

    ```
    $ head /etc/hosts
    192.168.99.100  guardarpunto.example.com
    ...
    ```

- **Creación del directorio asociado a PVC**. Para que los datos de la base de datos sean permanetes, se deberá crear un directorio determinado, /mnt/data, en el host, en este caso, en minikube. Por lo tanto:

  - Acceder a minikube por ssh y crear el directorio /mnt/data:

    ```
    $ minikube ssh
                            _                 _
                _          _ ( )           ( )
      ___ ___  (_)  ___  (_)| |/')  _   _ | |_      __
    /' _ ` _ `\| |/' _ `\| || , <  ( ) ( )| '_`\  /'__`\
    | ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )(  ___/
    (_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/''\____)

    $ mkdir /mnt/data/
    $ exit
    logout
    ```

    En el caso que el directorio /mnt/data estuviese ya creado, se ruega borrar por completo y recrearlo para evitar problemas en el arranque de la base de datos.

- **Clonar el código de la aplicación**:

  ```
  $ git clone https://github.com/sarroutbi/guardarpunto.git
  ```

- **Acceder a la raíz del proyecto, y desplegar la aplicación con "Helm"**:

  ```
  $ cd guardarpunto
  $ helm install --name guardarpunto helm/guardarpunto/
  ```

Posteriormente, comprobar que los diversos PODs se encuentran arrancados:

```
$ kubectl get pods
NAME                              READY     STATUS    RESTARTS   AGE
internalsvc-5479b96564-jzhz6      1/1       Running   0          19m
internalsvc-5479b96564-s4xmp      1/1       Running   0          19m
mysqlsrv-845b6cd78b-s9bqn         1/1       Running   0          19m
webfront1-95d579447-vj2xt         1/1       Running   0          19m
webfront2-5787f49c88-d5m4v        1/1       Running   0          19m
```

NOTA: La primera vez que se despliegue la aplicación, los contenedores se bajarán de forma completa desde dockerhub, luego se tardará varios minutos hasta que los pods estén desplegados.

- **Acceder con el navegador a la página principal del proyecto:**
  http://guardarpunto.example.com
  Es probable que, en el primer acceso, el navegador bloquee el acceso a la página por la gestión de los certificados en el acceso por HTTPS. Si es este el caso se deberá, en primer lugar:

  1. Presionar sobre "Advanced"
  2. Una vez abierto el submenú avanzado, seleccionar "Add Exception"

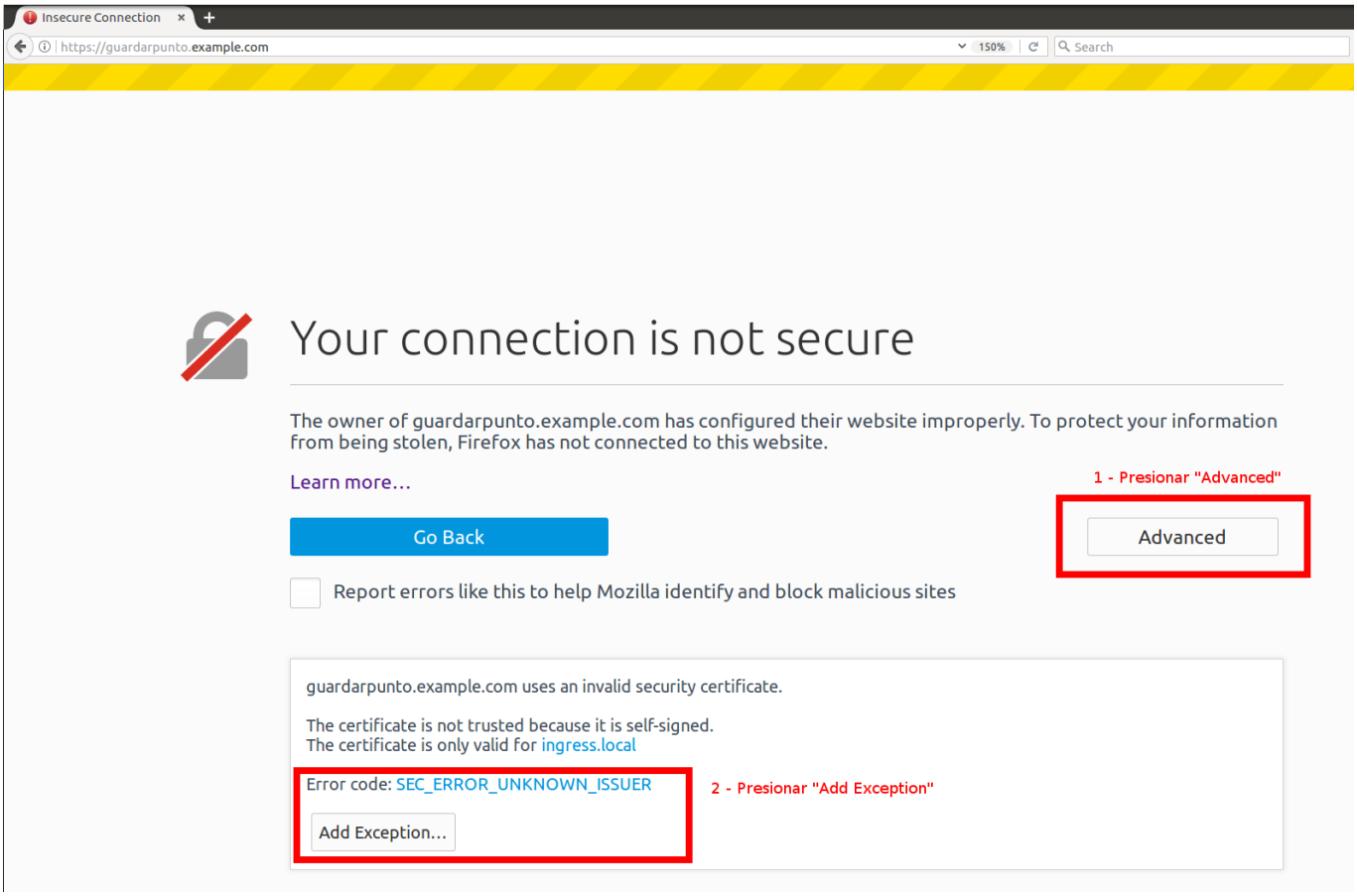

Figura 7: Alerta sobre conexión no segura

3. Finalmente, se deberá presionar la opción “Confirm Security Exception”

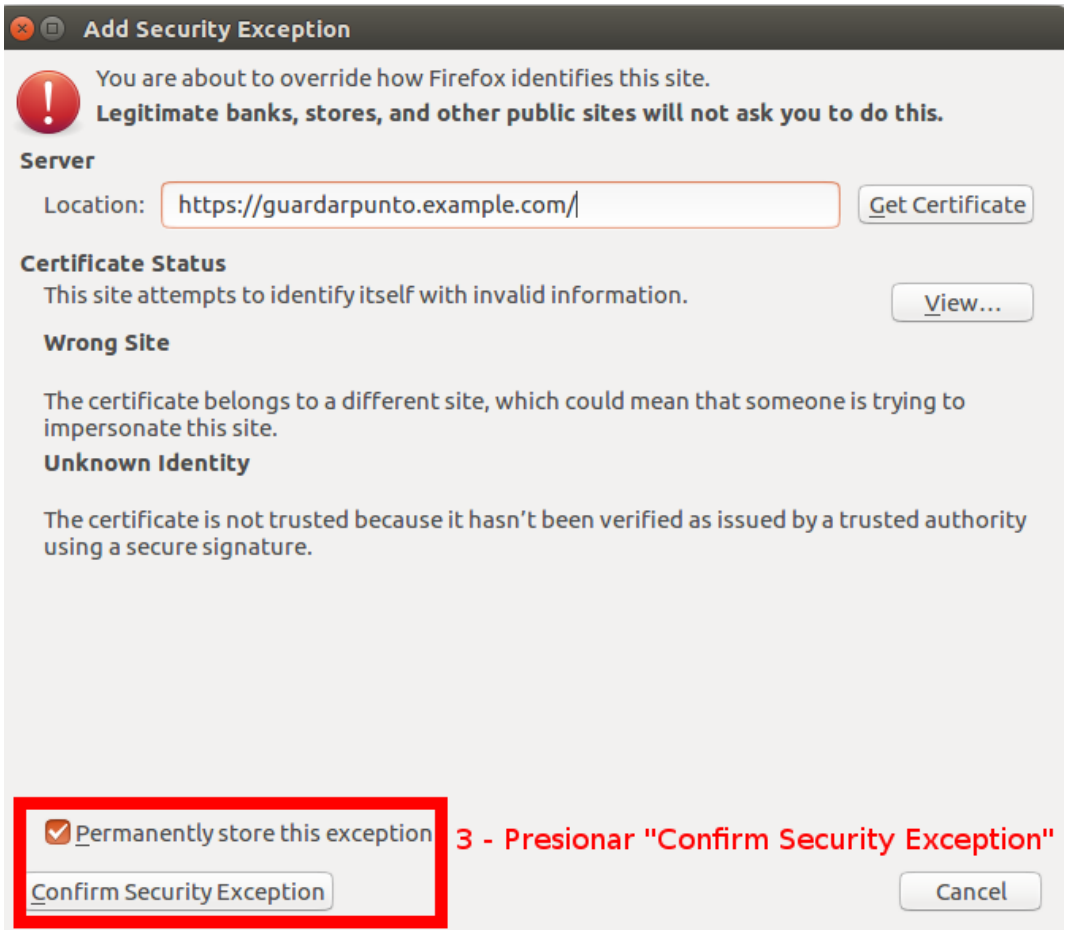

Figura 8: Confirmar Excepción de Seguridad

Una vez confirmada la excepción de seguridad, deberá observarse la página principal de la aplicación Web distribuída ”Guardarpunto”, servida por el frontal web, una imagen similar a la Figura 9, que se muestra a continuación:

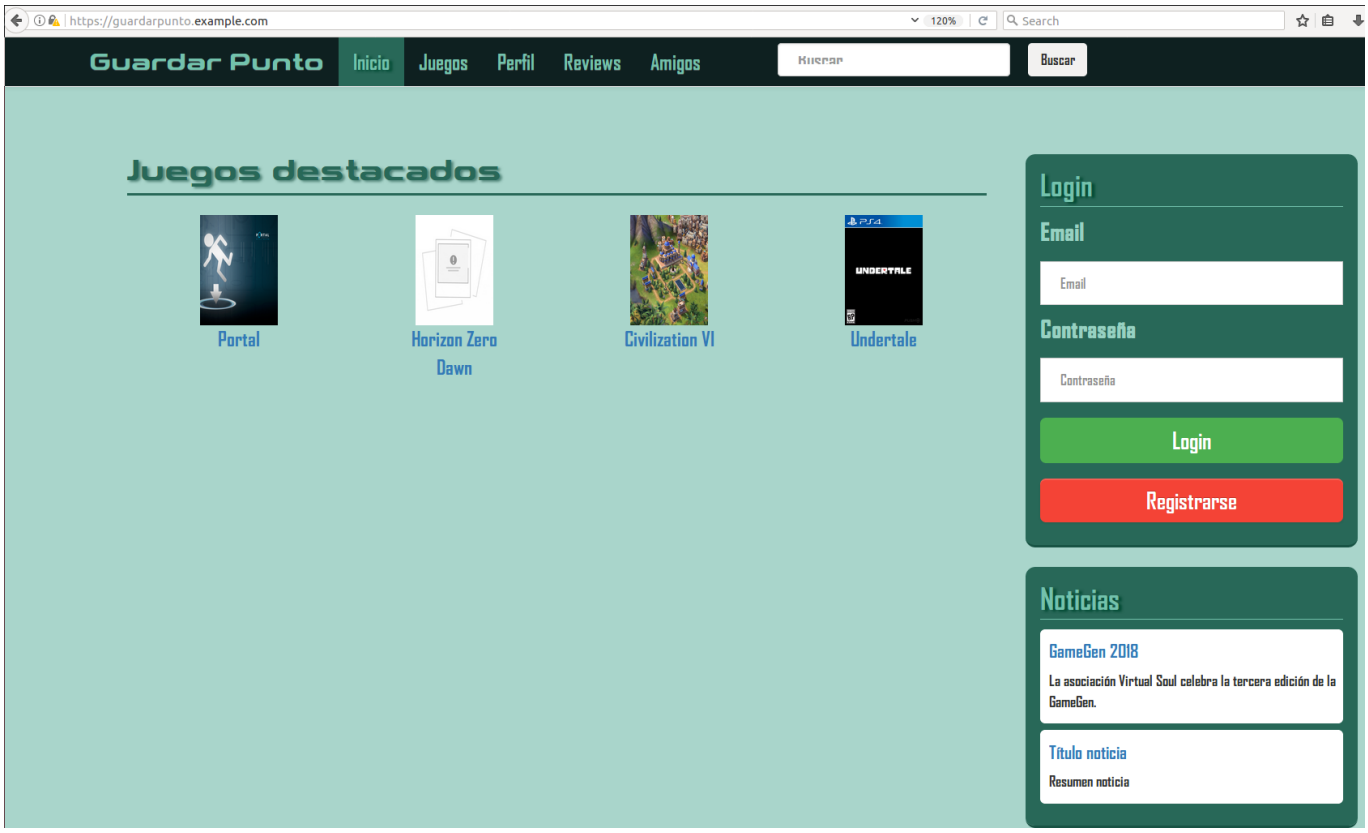

Figura 9: Página de inicio

3.2. Descripción de herramientas software utilizadas

Con motivo de depurar posibles problemas en el despliegue, se enumeran las diferentes herramientas software utilizadas para posibilitar el despliegue en Kubernetes, así como las versiones de cada una de ellas:

1. **minikube**: Como herramienta de despliegue de un cluster mínimo de Kubernetes. La versión utilizada ha sido la siguiente: **v0.35.0**. En cuanto a los "addons" de aplicación habilitados, se encuentran los siguientes:

```
$ minikube addons list
- addon-manager: enabled
- dashboard: disabled
- default-storageclass: disabled
- efk: disabled
- freshpod: disabled
- gvisor: disabled
- heapster: disabled
- ingress: enabled
- logviewer: disabled
- metrics-server: disabled
- nvidia-driver-installer: disabled
- nvidia-gpu-device-plugin: disabled
```

```
- registry: disabled
- registry-creds: disabled
- storage-provisioner: enabled
- storage-provisioner-gluster: disabled
```

2. **helm**: Como herramienta de gestión de paquetes en Kubernetes. La versión utilizada ha sido la siguiente, tanto para cliente como para servidor: **v2.11.0**.

```
$ helm version
Client: &version.Version{SemVer:''v2.11.0'', GitCommit:''2e55dbe''}
Server: &version.Version{SemVer:''v2.11.0'', GitCommit:''2e55dbe''}
```

3. **docker**: Como herramienta de gestión de contenedores. La versión utilizada ha sido la siguiente, tanto para cliente como para servidor: **18.06.1-ce**.

```
Client:
 Version:           18.06.1-ce
 API version:       1.38
 Go version:        go1.10.3
 Git commit:        e68fc7a
 Built:             Tue Aug 21 17:24:56 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
 Version:          18.06.1-ce
 API version:      1.38 (minimum version 1.12)
 Go version:       go1.10.3
 Git commit:       e68fc7a
 Built:            Tue Aug 21 17:23:21 2018
 OS/Arch:          linux/amd64
 Experimental:     false
```