

# Despliegue Base del Cloud Inter Universitarios para Emergencias

Mediante el uso de Tecnologías de Container y  
Orquestación de Container sobre Ubuntu Server  
18.04

Proyecto FONDEF ID 15 | 20560

Plataforma de Apoyo a la Gestión de Emergencia y Aplicaciones



## **Despliegue Base para el Cloud Inter Universitarios para Emergencias**

Mediante uso de Tecnología de Container  
y Orquestador de Container sobre  
Ubuntu Server 18.04

Proyecto FONDEF ID 15 | 20560

Plataforma de Apoyo a la Gestión de  
Emergencia y Aplicaciones

© Citiaps

Derechos reservados.

Primera edición: marzo 2021.

### **Contacto**

<https://citiaps.usach.cl>

citiaps@usach.cl

# Tabla de contenidos

|   |           |
|---|-----------|
| <b>Tabla de contenidos</b>                            | <b>4</b>  |
| <b>Presentación</b>                                   | <b>5</b>  |
| <b>Despliegue del Cloud y Configuración del Cloud</b> | <b>6</b>  |
| 2.1. Crear Máquinas Virtuales                         | 8         |
| 2.2. Configurar Subred de VMs                         | 10        |
| 2.3. Crear Repositorio de Imágenes de Containers      | 11        |
| 2.4. Crear Repositorio de Datos                       | 17        |
| 2.4.1. Instalación de MongoDB                         | 17        |
| 2.4.2. Instalación de MySql                           | 18        |
| 2.5. Crear Cache                                      | 20        |
| 2.6. Instalar tecnología de Container y Orquestador   | 21        |
| 2.7. Crear Servicios Básicos de DB y Cache            | 25        |
| 2.7.1. Creación del servicio mongodb                  | 25        |
| 2.7.2. Creación del servicio mysql                    | 26        |
| 2.7.3. Creación del servicio cache                    | 27        |
| <b>3. Despliegue de Aplicaciones sobre el Cloud</b>   | <b>28</b> |
| 3.1. Crear y Subir Container de Aplicación            | 29        |
| 3.2. Crear YAML de Configuración de la Aplicación     | 32        |
| 3.2.1. Instrucciones para crear los archivos          | 35        |
| 3.3. Lanzar Instancia de la Aplicación en el Cloud    | 39        |
| <b>4. Conclusiones</b>                                | <b>39</b> |

# 1. Presentación

El presente documento contiene la secuencia de paso para desplegar y configurar una versión base del cloud interuniversitario para emergencias, esto basado en tecnologías de container y orquestación de container sobre el sistema operativo Linux, distribución Ubuntu Server 18.04.

El despliegue se basa principalmente en dos aristas, que son: (1) los pasos para desplegar el cloud y dejarlo operativo, y (2) los pasos para desplegar aplicaciones de emergencias sobre ese cloud, ver Figura 1.1., que resume el contenido de este documento.

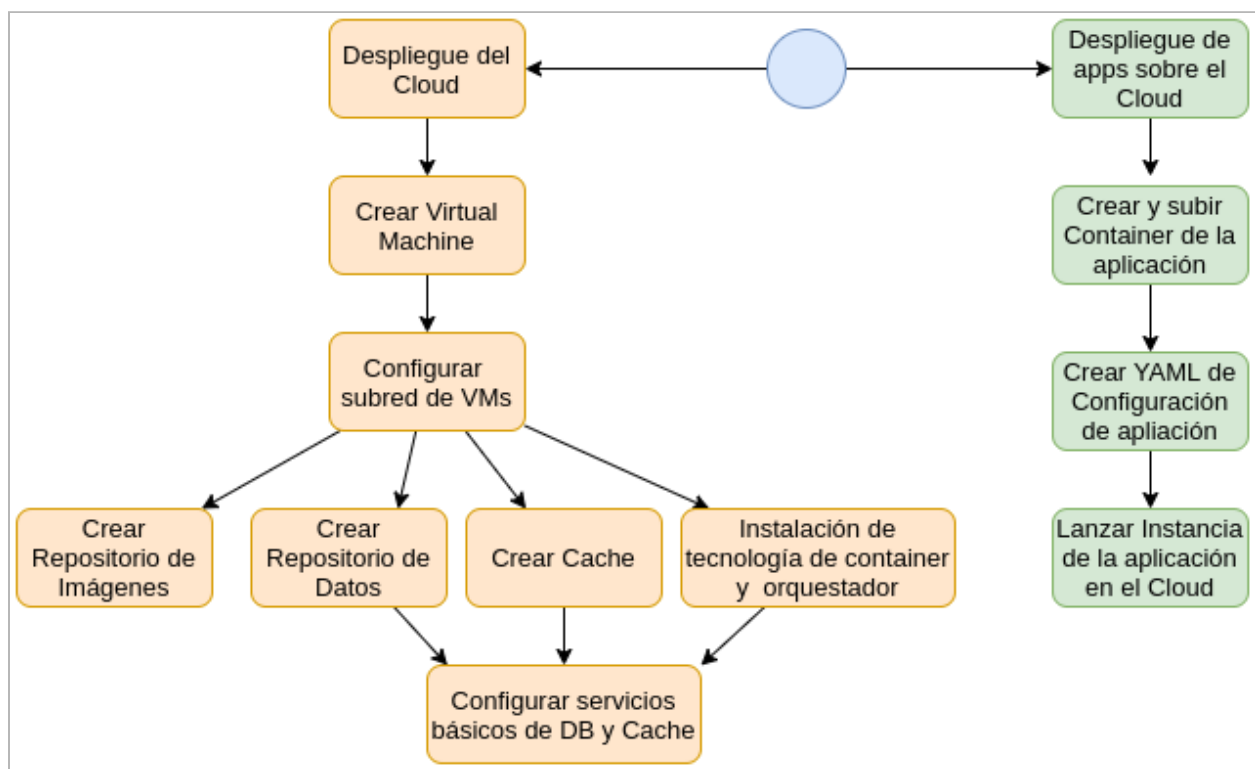


Figura 1.1: pasos para dejar operativo versión base de cloud para emergencias

Este documento fue creado, pensando en que los lectores ya conocen del tema en cuestión, por ende no se profundiza cada tópico, pero si se establece claramente la secuencia de pasos a seguir para el despliegue del cloud base.

## 2.Despliegue del Cloud y Configuración del Cloud

La arquitectura definida para el despliegue de las aplicaciones sobre la plataforma computacional o cloud universitarios para situaciones de desastre de origen natural, se basa en la tecnología de containers y orquestación de containers. Containers que surgen con el objetivo de realizar el despliegue de aplicaciones independiente del sistema operativo, ya que encapsula las aplicaciones y sus dependencias en módulos portables que corren bajo el mismo núcleo de sistema operativo, evitando la necesidad de crear máquinas virtuales para aislar las aplicaciones. Además, la orquestación de containers permite coordinar de buena forma la escalabilidad horizontal, la replicación y la tolerancia a fallas de las aplicaciones desplegadas mediante containers, ver la Figura 2.1, en esta se puede apreciar el diagrama de arquitectura para el despliegue de las aplicaciones sobre la plataforma computacional.

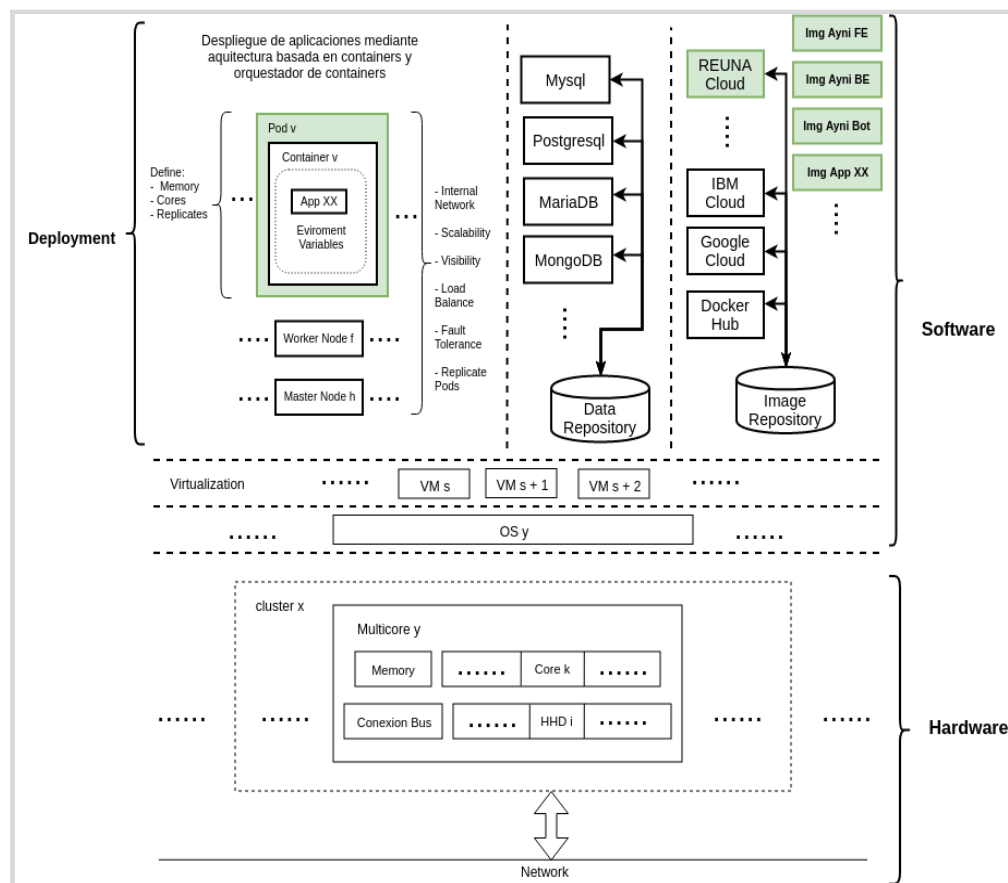


Figura 2.1: Arquitectura de despliegue de la plataforma computacional

Cabe destacar que los containers y el orquestador son tecnologías genéricas, de las cuales existen varias opciones para su implantación, en este caso puntual se ha seleccionado Docker Container para implantar la tecnología de container y Kubernetes para la orquestación de los containers, las principales razones por las cuales fueron escogidos son: que tienen un mayor respaldo de la comunidad de software libre, mejor documentación y de fácil integración.

Respecto del despliegue base que se plantea en este documento, este busca disponibilizar el cloud con todas las piezas claves que conllevan las características deseadas para que una aplicación sea de utilidad en una situación de desastre de origen natural, que son: alta disponibilidad, tolerancia a fallas, escalable y portabilidad. La Figura 2.2 contiene los componentes de este despliegue base, a continuación se describen cada uno de ellos:

- Master, es el componente principal disponibilizado por Kubernetes para realizar las veces de orquestador de la plataforma, permite establecer el contacto con los clientes de los sistemas y entre los servicios que componen el sistema.
- Image Repository o Repositorio de Imágenes (docker-registry, ya que usamos la tecnología de Docker), este componente es clave en la tolerancia a fallas y portabilidad de la plataforma, ya que contiene las imágenes de los containers, es decir, la aplicación con todas sus dependencias sin ser desplegada. Al momento de querer desplegar la aplicación esta es encapsulada como un servicio, el cual, puede interactuar con toda la plataforma.
- Data Repository o Repositorio de Datos, este componente hace referencia a las bases de datos que serán utilizadas por los sistemas, donde se espera que estas sean tolerantes a fallas, replicables y tengan alta disponibilidad, un ejemplo claro de este tipo de base de datos es mongodb, utilizada en algunos de los sistemas desarrollados por CITIAPS.
- Cache, este componente es clave para el manejo de estados internos en los sistemas (cuando es requerido), ejemplo: voluntario acepta tarea o voluntario finaliza tarea. La tecnología utilizada en esta plataforma es REDIS.
- Worker X, es el componente que encapsula las aplicaciones cuando estas son desplegadas en la plataforma, es decir, la aplicación se encuentra sin desplegar en

el docker-registry, cuando esta quiere ser desplegada, se deben entregar todas las variables de entorno en un archivo YAML de configuración y se lanza el despliegue mediante instrucciones de Kubernetes, finalmente la aplicación es encapsulada en un Worker, el cual mediante un servicio permite la interacción de la aplicación con el resto de la plataforma.

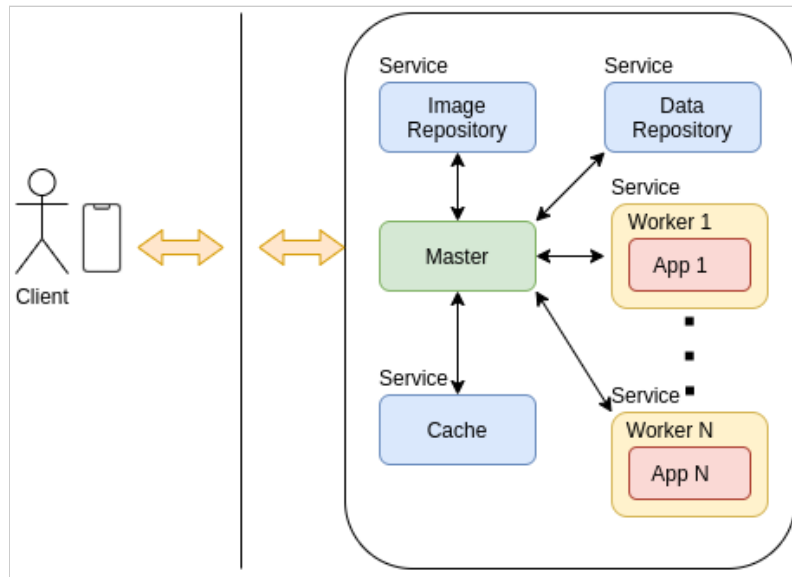


Figura 2.2: Despliegue Base del Cloud

## 2.1. Crear Máquinas Virtuales

Al trabajar en la distribución Ubuntu Server 18.04, se utiliza KVM para el despliegue y creación de Máquinas Virtuales (Virtual Machines), desde ahora VMs. Por ende si no lo tienen instalado, los siguientes son los comandos para realizar la instalación:

```
sudo apt update
```

```
sudo apt install cpu-checker
```

```
kvm-ok
```

```
sudo apt install qemu-kvm libvirt-bin bridge-utils virtinst virt-manager
```



Una vez instalado, se disponibiliza una interfaz gráfica sencilla de utilizar para la creación de las VMs, con la cual debe crear la distribución de VMs planteadas en la Tabla 2.1, que especifica los parámetros que debe tener cada VM. Cabe destacar que el “user” y “pass” aquí especificados son genéricos, usted debe establecer los propios.

Tabla 2.1: Especificación de VMs

| VM          | User | Pass  | CORES | RAM GB | HDD GB | S.O.                |
|-------------|------|-------|-------|--------|--------|---------------------|
| master01    | lvc  | 12345 | 3     | 4      | 40     | Ubuntu server 18.04 |
| repoimage01 | lvc  | 12345 | 2     | 2      | 300    | Ubuntu server 18.04 |
| repodata01  | lvc  | 12345 | 4     | 8      | 300    | Ubuntu server 18.04 |
| cache01     | lvc  | 12345 | 2     | 2      | 40     | Ubuntu server 18.04 |
| worker03    | lvc  | 12345 | 3     | 4      | 100    | Ubuntu server 18.04 |
| worker04    | lvc  | 12345 | 3     | 4      | 100    | Ubuntu server 18.04 |

Las apariencia final, luego de la creación de las VMs puede ser apreciada en la Figura 2.3, donde se aprecia un número antes del nombre de la VM, este número representa el último número de la IP de casa VM en la subred creada para la plataforma, esta se ve en detalle en la siguiente sección.

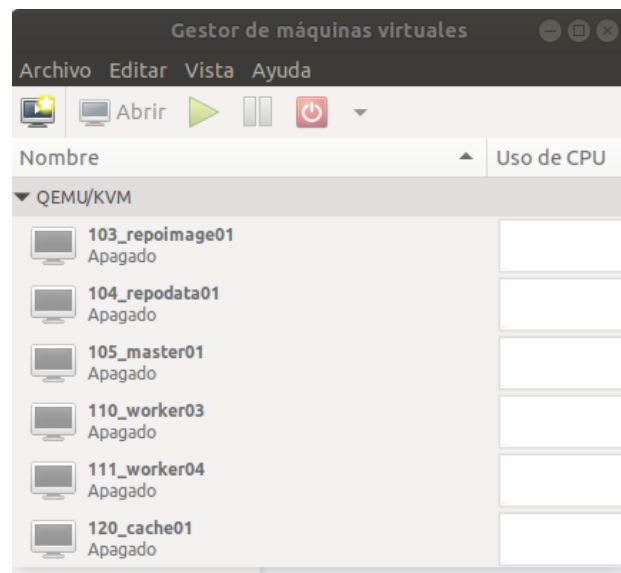


Figura 2.3. VMs Creadas

## 2.2. Configurar Subred de VMs

En esta sección se especifica cómo establecer la subred de las VMs que formarán parte de del cloud, es decir, se establece una ip estática por VM dentro de la red, con lo cual será posible la comunicación entre los distintos componentes de cada sistema. La secuencia de pasos a seguir es la siguiente:

- 1) Ingresar a la VM
- 2) Ingresar al archivo de configuración 50-cloud-init.yaml de la carpeta /etc/netplan/

```
sudo nano /etc/netplan/50-cloud-init.yaml
```

- 3) Establecer la ip de cada VM, las ip establecidas son:

```
192.168.1.103 ⇒ 103_repoimage01
192.168.1.104 ⇒ 104_repodata01
192.168.1.105 ⇒ 105_master01
192.168.1.110 ⇒ 110_worker03
192.168.1.111 ⇒ 111_worker04
192.168.1.120 ⇒ 120_cache01
```

Estando dentro del archivo de configuración borrar o comentar todo lo existente y fijar:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    ens3:
      dhcp4: no
      dhcp6: no
      addresses: [192.168.1.103/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [192.168.1.1, 8.8.8.8]
```

Con ctrl + x para salir y guardar los cambios (debe hacer lo mismo en cada VM cambiando la ip en “addresses”)

- 4) Reiniciar la red en la VM

```
sudo netplan apply
```

- 5) Configurar los hostname, correr el comando

```
sudo hostnamectl set-hostname repoimage01
```

- 6) Luego al archivo “hosts” de la carpeta /etc/, estando en este archivo configurar las IP, debe hacer dependiendo de la VM que está configurando, el contenido debe quedar así (en este caso se está configurando el repoimage01, debe hacer lo mismo para el resto de las VMs)

```
sudo nano /etc/hosts
```

```
127.0.0.1 repoimage01
```

```
127.0.1.1 repoimage01
```

```
192.168.1.103 repoimage01
```

```
192.168.1.104 repodata01
```

```
192.168.1.105 master01
```

```
192.168.1.110 worker03
```

```
192.168.1.111 worker04
```

```
192.168.1.120 cache01
```

Con ctrl + x para salir y guardar los cambios

- 7) Reiniciar la VM para confirmar los cambios

## 2.3. Crear Repositorio de Imágenes de Containers

Cómo ya se mencionó con anterioridad, el repositorio de imágenes de container es un docker-registry, el cual se utiliza principalmente para almacenar los container de las aplicaciones antes de ser desplegadas, es decir, con todas sus dependencias, pero sin las

variables de entorno que parametrizan el despliegue de sus instancias. La secuencia de pasos para levantar este docker-registry es la siguiente:

- 1) Luego de haber configurado la red, ingresar a la VM repoimage01 (192.168.1.103)  
`ssh lvc@192.168.1.103 (pass 12345)`

## 2) Instalar Docker

```
sudo apt update
sudo apt upgrade
sudo apt-get install curl apt-transport-https ca-certificates
software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
sudo apt update
```

```
apt-cache policy docker-ce
```

```
sudo apt install docker-ce
```

## 3) Verificar intslación

```
sudo systemctl status docker
```

## 4) Instalar Docker-Compose

```
sudo curl -L
https://github.com/docker/compose/releases/download/1.21.2/docker-compose-`una
me -s`-`uname -m` -o /usr/local/bin/docker-compose
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
docker-compose --version
```

## 5) Configurar carpetas de docker-registry

```
mkdir ~/docker-registry && cd $_  
mkdir data  
mkdir certs  
mkdir auth
```

## 6) Creamos un certificado de seguridad openssl (partes cert+key), para dar una capa de seguridad al docker-registry, debe establecer su email. Debe ingresar a la carpeta ~/docker-registry.

```
nano openssl.conf
```

```
[ req ]  
distinguished_name = req_distinguished_name  
x509_extensions    = req_ext  
default_md         = sha256  
prompt            = no  
encrypt_key       = no
```

```
[ req_distinguished_name ]  
countryName        = "CH"  
localityName       = "Santiago"  
organizationName    = "lvc"  
organizationalUnitName = "lvc"  
commonName         = "192.168.1.103"  
emailAddress       = "lvc@usach.cl"
```

```
[ req_ext ]  
subjectAltName = @alt_names
```

```
[alt_names]  
DNS = "192.168.1.103"
```

Con ctrl + x para salir y guardar los cambios

## 7) Se genera el certificado basado en el archivo, para esto se corre el siguiente comando

```
openssl req \  
-x509 -newkey rsa:4096 -days 365 -config openssl.conf \  
-keyout certs/domain.key -out certs/domain.crt
```

8) Para verificar la existencia del certificado se puede hacer

```
openssl x509 -text -noout -in certs/domain.crt
```

9) Creamos el archivo docker-compose, que permitirá lanzar la instancia del docker-registry, éste se basa en la documentación estándar de Docker

```
nano docker-compose.yml
```

```
#-----  
version: '3.0'  
  
services:  
  
  registry:  
    container_name: docker-registry  
    restart: always  
    image: registry:2  
    environment:  
      REGISTRY_HTTP_TLS_CERTIFICATE: /certs/domain.crt  
      REGISTRY_HTTP_TLS_KEY: /certs/domain.key  
      REGISTRY_AUTH: htpasswd  
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd  
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm  
    ports:  
      - 5000:5000  
    volumes:  
      - docker-registry-data:/var/lib/registry  
      - ./certs:/certs  
      - ./auth:/auth  
  
volumes:  
  docker-registry-data: {}  
  
#-----
```

Con ctrl + x para salir y guardar los cambios

10) Instalar, configurar htpasswd y crea archivo con credenciales de autenticación

```
sudo apt-get update  
sudo apt-get install apache2-utils
```

11) Para verificar la instalación

```
which htpasswd
```

12) Creamos las credenciales y las dejamos en "auth/htpasswd", considerar que estas pueden ser cambiadas por que usted estime (nombre de usuario: sysuser - contraseña: syspassword).

```
htpasswd -Bbn sysuser syspassword > auth/htpasswd
```

13) Ahora el sistema está listo para correr el docker-registry, el cual queda a la escucha de peticiones en el puerto 5000, correr el siguiente comando

```
sudo docker-compose up -d
```

14) Verificamos que está corriendo correctamente

```
sudo docker-compose ps
```

15) En cada VM que se instale docker (repoimage01, master01, worker03,worker04), debe correr los siguientes comandos, para informar donde queda docker-registry, y así permitir su comunicación.

```
sudo nano /etc/docker/daemon.json
```

```
{  
  "insecure-registries": ["192.168.1.103:5000"]  
}
```

Con ctrl + x para salir y guardar los cambios. Además reiniciar docker

```
sudo systemctl docker restart
```

ó

```
sudo service docker restart
```

Ya que cuenta con un docker-registry con autenticación, los siguientes pasos son para verificar que todo quedó correctamente y podemos subir un container al repositorio de imágenes.

- A) Creamos el archivo para generar la imagen de prueba (el cual permitirá crear un container de prueba con la última versión de Ubuntu)

```
nano Dockerfile
```

```
FROM ubuntu:latest  
CMD tail -t /dev/null
```

Con ctrl + x para salir y guardar los cambios

- B) Luego ingresar al docker-registry con las credenciales (nombre de usuario y contraseña señaladas anteriormente)

```
sudo docker login 192.168.1.103:5000
```

- C) Creamos la imagen basada en el Dockerfile

```
sudo docker build -t 192.168.1.103:5000/myubuntu:v1 .
```

- D) Subimos la imagen a docker-registry

```
sudo docker push 192.168.1.103:5000/myubuntu:v1
```

- E) Borramos la Imagen del docker-registry

```
sudo docker rmi 192.168.1.103:5000/myubuntu:v1
```



F) Descargamos la imagen del docker-registry

```
sudo docker pull 192.168.1.103:5000/myubuntu:v1
```

G) Dar permisos en el cortafuegos

```
sudo ufw allow from 192.168.1.0/24 to any port 5000
sudo ufw allow from 192.168.0.0/24 to any port 5000
sudo ufw allow from 10.0.0.0/24 to any port 5000
sudo ufw allow ssh
```

## 2.4. Crear Repositorio de Datos

En esta VM (reodata01 - 192.168.1.104) se instalarán mongodb y mysql, que son las bases de datos utilizadas por las aplicaciones manejadas en el cloud para emergencias.

### 2.4.1. Instalación de MongoDB

Para instalar la versión 4.0 de mongodb debe seguir la siguiente secuencia de instrucciones:

1) Instalar mongodb

```
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key  
add -
```

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu  
bionic/mongodb-org/4.4 multiverse" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.4.list
```

```
sudo apt update
sudo apt-get install -y mongodb-org=4.4.2 mongodb-org-server=4.4.2  
mongodb-org-shell=4.4.2 mongodb-org-mongos=4.4.2 mongodb-org-tools=4.4.2
```

```
sudo systemctl status mongod
```

- 2) Configurar mongodb, modificar el archivo mongod.conf de /etc/

```
sudo nano /etc/mongod.conf
```

```
bind_ip = 127.0.0.1,192.168.1.104
```

Con ctrl + x para salir y guardar los cambios

- 3) Reiniciar mongodb

```
sudo systemctl restart mongod
```

## 2.4.2. Instalación de MySQL

Para instalar la versión 5.7.31 de MySQL debe seguir la siguiente secuencia de instrucciones:

- 1) Instalar mysql

```
sudo apt update
```

```
sudo apt install mysql-server
```

```
sudo mysql_secure_installation
```

Durante la instalación el sistema solicita datos de seguridad, debe ingresar los siguientes datos a cada pregunta, donde "syspassword" es la contraseña del usuario root.

```
n
syspassword
syspassword
y
y
y
y
```

- 2) Modificar configuración interna de mysql para aceptar la password

```
sudo mysql
```

```
SELECT user,authentication_string,plugin,host FROM mysql.user;
```

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY  
'syspassword';
```

```
FLUSH PRIVILEGES;
```

```
exit;
```

```
mysql -u root -p  
syspassword
```

```
grant all privileges on *.* to 'root'@'%' identified by 'syspassword';
```

```
FLUSH PRIVILEGES;
```

```
exit;
```

### 3) Cambiar archivo de configuración de mysql, comentar las líneas

```
sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

```
#skip-external-locking  
#bind-address 127.0.0.1
```

Con ctrl + x para salir y guardar los cambios

### 4) Reiniciar mysql

```
sudo service mysql status  
sudo service mysql restart  
sudo service mysql status
```

### 5) Dar permisos en el cortafuegos

```
sudo ufw allow from 192.168.1.0/24 to any port 27017  
sudo ufw allow from 192.168.0.0/24 to any port 27017  
sudo ufw allow from 10.0.0.0/24 to any port 27017  
sudo ufw allow from 192.168.1.0/24 to any port 3306  
sudo ufw allow from 192.168.0.0/24 to any port 3306  
sudo ufw allow from 10.0.0.0/24 to any port 3306  
sudo ufw allow ssh
```

## 2.5. Crear Cache

En este apartado se instala y configura el Cache, que cómo se mencionó anteriormente es implementado mediante REDIS. La secuencia de pasos es la siguiente:

1) Ingresar al cache01 (192.168.1.120)

2) Instalar Redis

```
sudo apt install redis-server
```

3) Modificar el archivo redis.conf en /etc/redis, se debe quitar o comentar **supervised no** y luego agregar **supervised systemd**. Además, establecer el **bind**

```
sudo nano /etc/redis/redis.conf
```

```
#supervised no  
supervised systemd
```

```
#bind 127.0.0.1 ::1  
bind 192.168.1.120
```

Con ctrl + x para salir y guardar los cambios

4) Resetear Redis

```
sudo systemctl restart redis
```

5) Verificar lo anterior

```
sudo netstat -lnp | grep redis
```

6) Dar permisos en el cortafuegos

```
sudo ufw allow from 192.168.1.0/24 to any port 6379  
sudo ufw allow from 192.168.0.0/24 to any port 6379  
sudo ufw allow from 10.0.0.0/24 to any port 6379  
sudo ufw allow ssh
```

## 2.6 Instalar tecnología de Container y Orquestador

Se debe realizar la instalación de Docker (container) y Kubernetes (orquestador) sobre todos las VMs que forman parte directa del cloud que son los componentes Masters y Workers, al ser una versión base del cloud solo se instalará sobre las VMs: master01 (ip: 192.168.1.105), worker03 (ip: 192.168.1.110) y worker04 (ip: 192.168.1.111), pero antes de realizar la instalación se debe autorizar la conexión ssh cómo root de las VMs, la secuencia completa es la siguiente:

- 1) Autorizar ssh cómo root (realizar los paso de 1 al 9 en cada VM mencionada anteriormente)

```
sudo su
```

Pedirá las pass se debe ingresar 12345, con eso se conectará a la cuenta root, estando en la cuenta root debe asignar las password a esa cuenta, para eso ejecute el comando

```
passwd
```

Luego ingresar 12345 y repetirla 12345 (recuerde que puede establecer la password se su elección)

- 2) Ahora debe modificar el archivo de configuración para conexiones ssh, para autorizar la conexión cómo root (agregar o descomentar "PermitRootLogin yes")

```
nano /etc/ssh/sshd_config
```

```
PermitRootLogin yes
```

Con ctrl + x para salir y guardar los cambios

- 3) Restaurar el servicio de ssh

```
sudo service sshd restart
```

- 4) Luego salir de ssh completamente, para esto debe ejecutar dos veces el comando  
exit

```
exit  
exit
```

- 5) Volver a conectarse vía ssh, esta vez cómo root (192.168.1.105, 192.168.1.110 y 192.168.1.111)

```
ssh root@192.168.1.105
```

- 6) Deshabilitar el cortafuego de estas VMs y el swap

```
ufw disable  
swapoff -a; sed -i '/swap/d' /etc/fstab
```

- 7) Actualizar el sysctl, estableciendo la configuración base de la red que montará kubernetes

```
cat >>/etc/sysctl.d/kubernetes.conf<<EOF  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
EOF  
sysctl --system
```

- 8) Instalar docker

```
sudo apt update  
sudo apt upgrade  
sudo apt-get install curl apt-transport-https ca-certificates  
software-properties-common
```

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
sudo add-apt-repository "deb [arch=amd64]  
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
```

```
sudo apt update
```

```
apt-cache policy docker-ce
```

```
sudo apt install docker-ce
```

```
sudo systemctl status docker
```

## 9) Instalar componentes de kubernetes

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
```

```
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" >  
/etc/apt/sources.list.d/kubernetes.list
```

```
apt update && apt install -y kubeadm kubelet kubectl
```

10) Estos comandos solo se ejecutan en el master01 (192.168.1.105), instrucciones del 10 al 12

```
systemctl enable docker.service
```

```
kubeadm init --apiserver-advertise-address=192.168.1.105  
--pod-network-cidr=10.233.0.0/16
```

## 11) Desplegar red Calico en kubernetes

```
kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f  
https://docs.projectcalico.org/v3.14/manifests/calico.yaml
```

- 12) Crear el token para que los Workers se conecten al cloud administrado por el Master, lo que se genera es un comando que se debe correr sobre cada Worker y para correrlo debes estar conectado vía ssh cómo root

```
kubeadm token create --print-join-command
```

- 13) Conectarse vía ssh a los Workers (192.168.1.110 y 192.168.1.111) Debe ejecutar el comando genera en (12) y con esto los Workers quedan enlazados al cloud

- 14) Salir de todas las conexiones ssh y volver a entrar al master01 con el usuario regular

```
ssh lvc@192.168.1.105
```

- 15) Correr los siguientes comando para permitir que este VM pueda correr los comando de kubernetes

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

- 16) En este punto el cluster ya está desplegado, solo debe validar su funcionamiento, algunos comando utilices son

```
kubectl get nodes
```

```
kubectl get cs
```

```
kubectl cluster-info
```

```
kubectl version --short
```

```
kubectl get all
```

```
kubectl get all -o wide
```



```
watch kubectl get all -o wide
```

- 17) Establecer conexión con el docker-registry, para que el cloud sea capaz de acceder a los container cada vez que sea requerido (cuando se despliegan nuevas instancias o se levantan nuevas instancias por tolerancia a fallas). Recordar que el docker-registry quedó con cuenta de usuarios. Corra el siguiente comando para establecer la conexión (solo se ejecuta una vez y queda registrado).

```
kubectl create secret docker-registry mydockercredentials --docker-server  
192.168.1.103:5000 --docker-username sysuser --docker-password syspassword
```

## 2.7 Crear Servicios Básicos de DB y Cache

El cloud orquestado por kubernetes utiliza a todos los componentes que se integran a el cómo servicios, diciendo esto las bases de datos y el cache no son la excepción, en esta sección se describe la forma en la que se crean dichos servicios.

### 2.7.1. Creación del servicio mongodb

Debe seguir la siguiente secuencia de pasos:

- 1) Crear el archivo YAML de configuración

```
nano mongo-service.yaml  
  
---  
kind: "Service"  
apiVersion: "v1"  
metadata:  
  name: "external-mongodb-service"  
spec:  
  ports:  
  -  
    name: "mongo-db"  
    protocol: "TCP"
```

```

    port: 27017
    targetPort: 27017

---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "external-mongodb-service"
subsets:
-
  addresses:
  -
    ip: "192.168.1.104"
  ports:
  -
    port: 27017
    name: "mongo-db"

```

Con ctrl + x para salir y guardar los cambios

- 2) Lanzar la ejecución, con esto queda corriendo el servicio y puede ser utilizado por las aplicaciones

```
kubectl apply -f mongo-service.yaml
```

## 2.7.2. Creación del servicio mysql

Debe seguir la siguiente secuencia de pasos:

- 1) Crear el archivo YAML de configuración

```

nano mysql-service.yaml

---
kind: "Service"
apiVersion: "v1"
metadata:
  name: "external-mysqldb-service"
spec:
  ports:

```

```

-
  name: "mysql-db"
  protocol: "TCP"
  port: 3306
  targetPort: 3306

---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "external-mysqldb-service"
subsets:
-
  addresses:
  -
    ip: "192.168.1.104"
  ports:
  -
    port: 3306
    name: "mysql-db"

```

Con ctrl + x para salir y guardar los cambios

- 2) Lanzar la ejecución, con esto queda corriendo el servicio y puede ser utilizado por las aplicaciones

```
kubectl apply -f mysql-service.yaml
```

### 2.7.3. Creación del servicio cache

Debe seguir la siguiente secuencia de pasos:

- 3) Crear el archivo YAML de configuración

```
nano redis-service.yaml
```

```

---
kind: "Service"
apiVersion: "v1"

```

```

metadata:
  name: "external-redis-service"
spec:
  ports:
  -
    name: "redis-cache"
    protocol: "TCP"
    port: 6379
    targetPort: 6379

---
kind: "Endpoints"
apiVersion: "v1"
metadata:
  name: "external-redis-service"
subsets:
-
  addresses:
  -
    ip: "192.168.1.120"
  ports:
  -
    port: 6379
    name: "redis-cache"

```

Con ctrl + x para salir y guardar los cambios

- 4) Lanzar la ejecución, con esto queda corriendo el servicio y puede ser utilizado por las aplicaciones

```
kubectl apply -f redis-service.yaml
```

### 3. Despliegue de Aplicaciones sobre el Cloud

En esta sección se describe la instalación de Ayni, sistema para reclutamiento de voluntarios. Deben solicitar la versión Cloud de Ayni vía formulario (contempla 2 componentes lvc-ayni-fondef-bot-webhook y lvc-ayni-fondef-backend), que es una versión de pruebas, pero que servirá para que experimente con el despliegue en el cloud. Cabe

destacar que el bot de Ayni es de Telegram, y para correr tus instancias debes crear un nuevo bot y especificar las credenciales del archivo YAML que se verá en el punto 3.2.

Ayni es una aplicación creada por CITIAPS, cuyo objetivo es encargarse de gestionar las acciones que deben realizar grupos de voluntarios, estos voluntarios son congregados mediante un "BOT", implementado con el API disponibilizado por Telegram. Estos voluntarios se inscriben en la emergencia en curso e informan los requisitos con los que cumplen (físicos, habilidades, conocimiento, etc.), luego pasan por un proceso de selección automático. Finalmente, el voluntario puede participar en las tareas creadas producto de la emergencia. La arquitectura de este sistema está descrita en la Figura 3.1. y se distingue, un módulo de cache, dos Backend y un repositorio de datos.

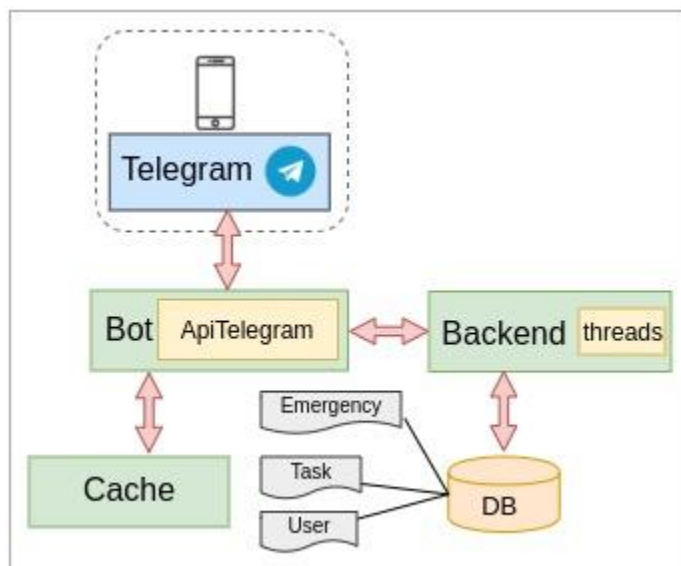
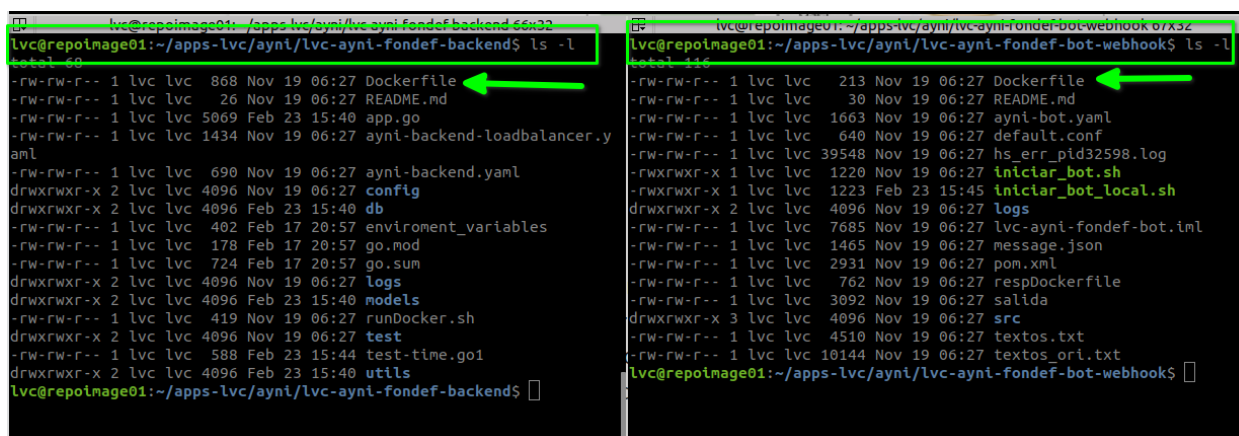


Figura 3.1: Arquitectura del sistema Ayni

### 3.1. Crear y Subir Container de Aplicación

Debe seguir la siguiente secuencia de pasos:

- 1) Descargar las versiones de Ayni disponibilizadas por CITIAPS antes mencionadas, estas deben ser descargadas en repoimage01 (192.168.1.103). Ambas carpetas contendrán el archivo "Dockerfile" para la creación del container, ver Figura 3.2, que nos da una idea de cómo se vería el ambiente descrito.

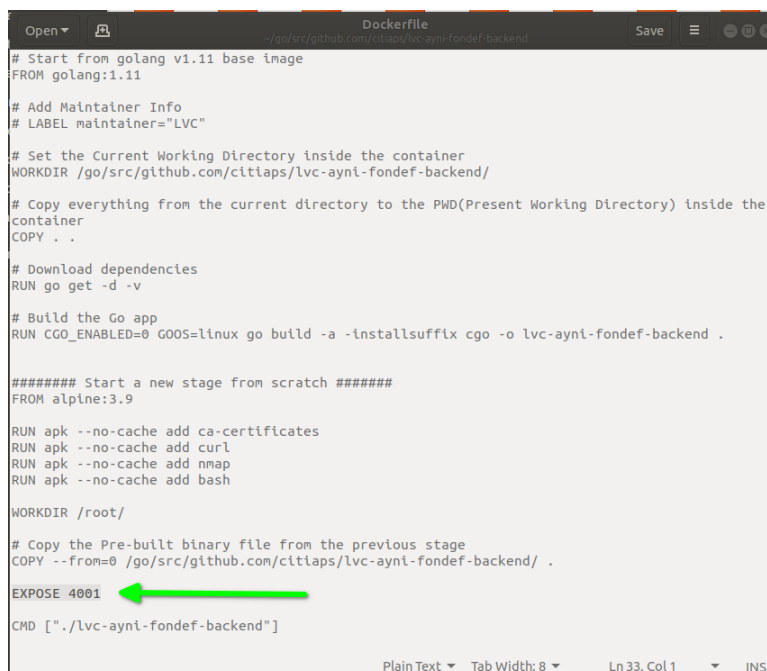


```
lvc@repoimage01:~/apps-lvc/ayni/lvc-ayni-fondef-backend$ ls -l
total 60
-rw-rw-r-- 1 lvc lvc 868 Nov 19 06:27 Dockerfile
-rw-rw-r-- 1 lvc lvc 26 Nov 19 06:27 README.md
-rw-rw-r-- 1 lvc lvc 5069 Feb 23 15:40 app.go
-rw-rw-r-- 1 lvc lvc 1434 Nov 19 06:27 ayni-backend-loadbalancer.yaml
-rw-rw-r-- 1 lvc lvc 690 Nov 19 06:27 ayni-backend.yaml
drwxrwxr-x 2 lvc lvc 4096 Nov 19 06:27 config
drwxrwxr-x 2 lvc lvc 4096 Feb 23 15:40 db
-rw-rw-r-- 1 lvc lvc 402 Feb 17 20:57 enviroment_variables
-rw-rw-r-- 1 lvc lvc 178 Feb 17 20:57 go.mod
-rw-rw-r-- 1 lvc lvc 724 Feb 17 20:57 go.sum
drwxrwxr-x 2 lvc lvc 4096 Nov 19 06:27 logs
drwxrwxr-x 2 lvc lvc 4096 Feb 23 15:40 models
-rw-rw-r-- 1 lvc lvc 419 Nov 19 06:27 runDocker.sh
drwxrwxr-x 2 lvc lvc 4096 Nov 19 06:27 test
-rw-rw-r-- 1 lvc lvc 588 Feb 23 15:44 test-time.go1
drwxrwxr-x 2 lvc lvc 4096 Feb 23 15:40 utils
lvc@repoimage01:~/apps-lvc/ayni/lvc-ayni-fondef-backend$

lvc@repoimage01:~/apps-lvc/ayni/lvc-ayni-fondef-bot-webhook$ ls -l
total 116
-rw-rw-r-- 1 lvc lvc 213 Nov 19 06:27 Dockerfile
-rw-rw-r-- 1 lvc lvc 30 Nov 19 06:27 README.md
-rw-rw-r-- 1 lvc lvc 1663 Nov 19 06:27 ayni-bot.yaml
-rw-rw-r-- 1 lvc lvc 640 Nov 19 06:27 default.conf
-rw-rw-r-- 1 lvc lvc 39548 Nov 19 06:27 hs_err_pid32598.log
-rw-rw-r-- 1 lvc lvc 1220 Nov 19 06:27 iniciar_bot.sh
-rw-rw-r-- 1 lvc lvc 1223 Feb 23 15:45 iniciar_bot_local.sh
drwxrwxr-x 2 lvc lvc 4096 Nov 19 06:27 logs
-rw-rw-r-- 1 lvc lvc 7685 Nov 19 06:27 lvc-ayni-fondef-bot.inl
-rw-rw-r-- 1 lvc lvc 1465 Nov 19 06:27 message.json
-rw-rw-r-- 1 lvc lvc 2931 Nov 19 06:27 pom.xml
-rw-rw-r-- 1 lvc lvc 762 Nov 19 06:27 respDockerfile
-rw-rw-r-- 1 lvc lvc 3092 Nov 19 06:27 salida
drwxrwxr-x 3 lvc lvc 4096 Nov 19 06:27 src
-rw-rw-r-- 1 lvc lvc 4510 Nov 19 06:27 textos.txt
-rw-rw-r-- 1 lvc lvc 10144 Nov 19 06:27 textos_ori.txt
lvc@repoimage01:~/apps-lvc/ayni/lvc-ayni-fondef-bot-webhook$
```

Figura 3.2: ambiente que debe crear de referencia en la VM repoimage01

- 2) Entrar a la carpeta lvc-ayni-fondef-backend (aplicación desarrollada en lenguaje Golang) y estando dentro de ella correr los siguientes comandos, con los cuales se crea el container del backend del sistema, que se conectará la base de datos mongodb, gracias al servicio creado anteriormente. Notar que la aplicación queda desplegada en el puerto 4001, según la especificación del Dockerfile, ver Figura 3.3. con el contenido del documento.



```
# Start from golang v1.11 base image
FROM golang:1.11

# Add Maintainer Info
# LABEL maintainer="LVC"

# Set the Current Working Directory inside the container
WORKDIR /go/src/github.com/citiaps/lvc-ayni-fondef-backend/

# Copy everything from the current directory to the PWD(Present Working Directory) inside the container
COPY . .

# Download dependencies
RUN go get -d -v

# Build the Go app
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o lvc-ayni-fondef-backend .

##### Start a new stage from scratch #####
FROM alpine:3.9

RUN apk --no-cache add ca-certificates
RUN apk --no-cache add curl
RUN apk --no-cache add nmap
RUN apk --no-cache add bash

WORKDIR /root/

# Copy the Pre-built binary file from the previous stage
COPY --from=0 /go/src/github.com/citiaps/lvc-ayni-fondef-backend/ .

EXPOSE 4001

CMD [\"./lvc-ayni-fondef-backend\"]
```

Figura 3.3 contenido del Dockerfile del backend de Ayni

```
#por si existe corra la versión existente
```

```
sudo docker rmi 192.168.1.103:5000/img_lvc_ayni_backend:v1
```

```
#crea el container
```

```
sudo docker build -t 192.168.1.103:5000/img_lvc_ayni_backend:v1 .
```

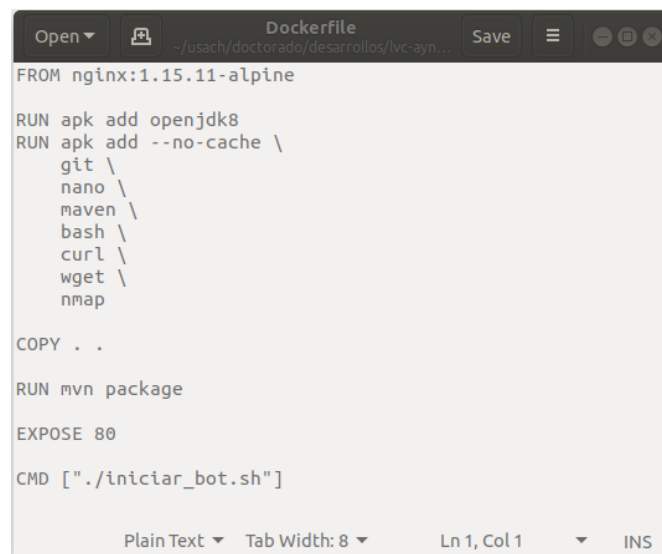
```
#lo sube al docker registry
```

```
sudo docker push 192.168.1.103:5000/img_lvc_ayni_backend:v1
```

Si tiene problemas por autenticación recuerde correr el comando, luego ingresar user y password establecidos en el punto 2.3. sección B de este documento.

```
sudo docker login 192.168.1.103:5000
```

- 3) Entrar a la carpeta lvc-ayni-fondef-bot-webhook (aplicación desarrollada en lenguaje Java) y estando dentro de ella correr los siguientes comandos, con los cuales se crea el container del bot del sistema, que se conectará con: al cache, al backend y a Telegram gracias a su API. Notar que la aplicación queda desplegada en el puerto 80, según la especificación del Dockerfile, y que luego vuelve a mapearse al puerto 4550 en el programa iniciar\_bot.sh (utilizado como lanzador del Dockerfile), ver Figura 3.4. Con el contenido del documento.



```
FROM nginx:1.15.11-alpine
RUN apk add openjdk8
RUN apk add --no-cache \
    git \
    nano \
    maven \
    bash \
    curl \
    wget \
    nmap
COPY . .
RUN mvn package
EXPOSE 80
CMD ["/iniciar_bot.sh"]
```

Figura 3.4: Dockerfile del bot e Ayni.

```
#por si existe corra la versión existente
```

```
sudo docker rmi 192.168.1.103:5000/img_lvc-ayni-fondef-bot:v1
```

```
#crea el container
```

```
sudo docker build -t 192.168.1.103:5000/img_lvc-ayni-fondef-bot:v1 .
```

```
#lo sube al docker registry
```

```
sudo docker push 192.168.1.103:5000/img_lvc-ayni-fondef-bot:v1
```

## 3.2. Crear YAML de Configuración de la Aplicación

La creación de los archivos YAML permite establecer el ambiente de configuración de las aplicaciones, y qué servicios utilizan cada una. Es importante destacar que luego de crear el YAML de configuración del backend de Ayni, este debe ser lanzado sobre la plataforma (ver sección 3.3), ya que este backend asume un puerto dinámico, el cual debe ser mapeado en los parámetros de configuración del bot de Ayni, en esta ocasión asumiremos que el puerto asignado al backend de ayni es 31757, con lo cual aislamos la sección de configuración.

Algo destacable de mencionar de los YAML de configuración es que se puede definir el número de réplicas que tendrá cada instancia. A continuación se presentan las Figuras 3.5., 3.6., 3.7. y 3.8. que muestran el contenido de los YAML del backend y del bot de Ayni, usuarios y contraseñas reales están borrados (recuerde que usted debe definir las propias, incluyendo las de Telegram). Posteriormente se deja una copia del YAML de backend y del bot de Ayni, para que le sea más fácil la creación de estos documentos. Cabe destacar que estos deben ser alojados en la VM master01, para poder posteriormente usar los archivos para lanzar las instancias requeridas.



```

---
apiVersion: v1
kind: Service
metadata:
  name: ayni-be-service
  labels:
    name: ayni-be
spec:
  type: LoadBalancer
  externalIPs:
  - 192.168.1.105
  ports:
  - port: 4001
    name: http
  selector:
    name: ayni-be

```

Figura 3.5, sección 2 del YAML del backend de Ayni (define al backend cómo un servicio LoadBalancer, visible por la subred)

```

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ayni-be-deployment
  labels:
    name: ayni-be
spec:
  replicas: 1
  selector:
    matchLabels:
      name: ayni-be
  template:
    metadata:
      labels:
        name: ayni-be
    spec:
      imagePullSecrets:
      - name: mydockercredentials
      containers:
      - name: ayni-be
        image: 192.168.1.103:5000/img_lvc_ayni_backend:v1
        imagePullPolicy: Always
        ports:
        - containerPort: 4001
        volumeMounts:
        - mountPath: /root/logs
          name: ayni-backend
      env:
      - name: AYNIDB_DB_HOST
        value: "external-mongodb-service"
      - name: AYNIDB_DB_NAME
        value: "aynidb"
      - name: AYNIDB_DB_USER
        value: "aynidbuser"
      - name: AYNIDB_DB_PASS
        value: "XXXXXXXXXX"
      - name: AYNIDB_ADDR
        value: "0.0.0.0:4001"
      - name: AYNIDB_TELEGRAM_BOT_NAME
        value: "LvcAyniBot"
      - name: AYNIDB_TELEGRAM_BOT_TOKEN
        value: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
      - name: AYNIDB_NAME_TEST
        value: "3coreWorkers-2coreOtros-4gbRamTodos"
      - name: AYNIDB_ID_TEST
        value: "0"
      volumes:
      - name: ayni-backend
        hostPath:
          path: /home/lvc/volumes

```

Figura 3.6, sección 1 del YAML del backend de Ayni (define variables de entorno)



Figura 3.7, sección 1 del YAML del bot de Ayni (define variables de entorno)

```

57 ---
58 apiVersion: v1
59 kind: Service
60 metadata:
61   name: ayni-bot-service
62   labels:
63     name: ayni-bot
64 spec:
65   type: NodePort
66   ports:
67     - port: 80
68       nodePort: 30081
69       name: http
70   selector:
71     name: ayni-bot
72

```

Figura 3.8, sección 2 del YAML del bot de Ayni (define al bot cómo un servicio con puerto estático, solo visible por los servicios de cloud)

### 3.2.1. Instrucciones para crear los archivos

- 1) Crear YAML para el backend de Ayni

```
nano ayni-backend-loadbalancer.yaml
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ayni-be-deployment
  labels:
    name: ayni-be
spec:
  replicas: 1
  selector:
    matchLabels:
      name: ayni-be
  template:
    metadata:
      labels:
        name: ayni-be
    spec:
      imagePullSecrets:
        - name: mydockercredenciales
      containers:

```

```

    - name: ayni-be
      image: 192.168.1.103:5000/img_lvc_ayni_backend:v1
      imagePullPolicy: Always
      ports:
        - containerPort: 4001
      volumeMounts:
        - mountPath: /root/logs
          name: ayni-backend
      env:
        - name: AYNIDB_DB_HOST
          value: "external-mongodb-service"
        - name: AYNIDB_DB_NAME
          value: "aynidb"
        - name: AYNIDB_DB_USER
          value: "aynidbUser"
        - name: AYNIDB_DB_PASS
          value: "xxxxxxxxxxxxxxxxxxxx"
        - name: AYNIDB_ADDR
          value: "0.0.0.0:4001"
        - name: AYNIDB_TELEGRAM_BOT_NAME
          value: "LvcAyniBot"
        - name: AYNIDB_TELEGRAM_BOT_TOKEN
          value: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
        - name: AYNIDB_TEST
          value: "3coreWorkers-2core0tros-4gbRamTodos"
        - name: AYNIDB_ID_TEST
          value: "0"
      volumes:
        - name: ayni-backend
          hostPath:
            path: /home/lvc/volumes

```

```
---
```

```

apiVersion: v1
kind: Service
metadata:
  name: ayni-be-service
  labels:
    name: ayni-be
spec:
  type: LoadBalancer

```

```
externalIPs:
- 192.168.1.105
ports:
- port: 4001
  name: http
selector:
  name: ayni-be
```

Con ctrl + x para salir y guardar los cambios

## 2) Crear YAML para el bot de Ayni

```
nano ayni-bot.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ayni-bot-deployment
  labels:
    name: ayni-bot
spec:
  replicas: 1
  selector:
    matchLabels:
      name: ayni-bot
  template:
    metadata:
      labels:
        name: ayni-bot
    spec:
      imagePullSecrets:
        - name: mydockercredenciales
      containers:
        - name: ayni-bot
          image: 192.168.1.103:5000/img_lvc-ayni-fondef-bot:v1
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - mountPath: /logs
              name: ayni-bot
```

```

    env:
      - name: bot_nombre
        value: "LvcAyniBot"
      - name: bot_token
        value: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
      - name: bot_url
        value: "https://9bd8fca87126.ngrok.io"
      - name: path_bot_basico
        value: "telegram_ayni_bot_basico"
      - name: port_to_intenalwebhook
        value: "4550"
      - name: external_call_to_webhook
        value: "http://localhost:4550/callback/telegram_ayni_bot_basico/"
      - name: url_base
        value: "192.168.1.105"
      - name: url_api
        value: "http://192.168.1.105:31757"
      - name: config_textos
        value: "textos.txt"
      - name: redis_host
        value: "external-redis-service:6379"
      - name: AYNI_NAME_TEST
        value: "3coreWorkers-2core0tros-4gbRamTodos"
      - name: AYNI_ID_TEST
        value: "0"
    volumes:
      - name: ayni-bot
        hostPath:
          path: /home/lvc/volumes
---
apiVersion: v1
kind: Service
metadata:
  name: ayni-bot-service
  labels:
    name: ayni-bot
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30081
      name: http

```

```
selector:  
  name: ayni-bot
```

Con ctrl + x para salir y guardar los cambios

### 3.3. Lanzar Instancia de la Aplicación en el Cloud

Una vez creado los archivos de configuración (recordar que luego de lanzar el backend de Ayni debe modificar el YAML de configuración del bot, para especificar el puerto correspondiente). Correr las instrucciones:

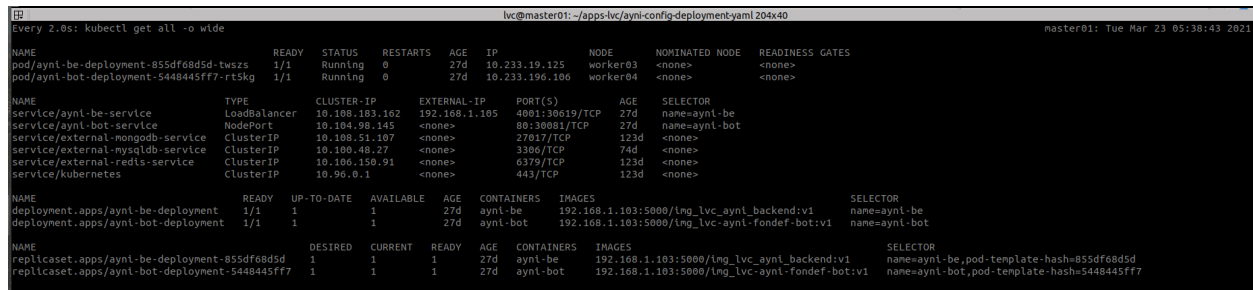
- 1) Lanzar backend de Ayni

```
kubectl apply -f ayni-backend-loadbalancer.yaml
```

- 2) Lanzar bot de Ayni

```
kubectl apply -f ayni-bot.yaml
```

Finalmente la apariencia del cloud desplegado puede ser vista en la Figura 3.9. En la cual se contemplan todos los servicios y las instancias desplegadas



```
lv@master01: ~/apps/lvc/ayni-config-deployment.yaml 204x40  
Every 2.0s: kubectl get all -o wide  
master01: Tue Mar 23 05:38:43 2021
```

| NAME                                     | READY | STATUS  | RESTARTS | AGE | IP             | NODE     | NOMINATED | NODE | READINESS | GATES |
|--|-------|---------|----------|-----|----------------|----------|-----------|------|-----------|-------|
| pod/aynl-be-deployment-855df68d5d-twszs  | 1/1   | Running | 0        | 27d | 10.233.19.125  | worker03 | <none>    |      | <none>    |       |
| pod/aynl-bot-deployment-5448445ff7-rt5kg | 1/1   | Running | 0        | 27d | 10.233.196.106 | worker04 | <none>    |      | <none>    |       |

| NAME                             | TYPE         | CLUSTER-IP     | EXTERNAL-IP   | PORT(S)        | AGE  | SELECTOR      |
|----------------------------------|--------------|----------------|---------------|----------------|------|---------------|
| service/aynl-be-service          | LoadBalancer | 10.108.183.162 | 192.168.1.105 | 4081:30619/TCP | 27d  | name=aynl-be  |
| service/aynl-bot-service         | NodePort     | 10.104.98.145  | <none>        | 80:30081/TCP   | 27d  | name=aynl-bot |
| service/external-mongodb-service | ClusterIP    | 10.108.51.187  | <none>        | 27817/TCP      | 123d | <none>        |
| service/external-mysqldb-service | ClusterIP    | 10.108.48.27   | <none>        | 3386/TCP       | 74d  | <none>        |
| service/external-redis-service   | ClusterIP    | 10.106.150.91  | <none>        | 6379/TCP       | 123d | <none>        |
| service/kubernetes               | ClusterIP    | 10.96.0.1      | <none>        | 443/TCP        | 123d | <none>        |

| NAME                                | READY | UP-TO-DATE | AVAILABLE | AGE | CONTAINERS | IMAGES  | SELECTOR      |
|-------------------------------------|-------|------------|-----------|-----|------------|---|---------------|
| deployment.apps/aynl-be-deployment  | 1/1   | 1          | 1         | 27d | aynl-be    | 192.168.1.103:5000/lmg_lvc_aynl_backend:v1    | name=aynl-be  |
| deployment.apps/aynl-bot-deployment | 1/1   | 1          | 1         | 27d | aynl-bot   | 192.168.1.103:5000/lmg_lvc_aynl-fondef-bot:v1 | name=aynl-bot |

| NAME   | DESIRED | CURRENT | READY | AGE | CONTAINERS | IMAGES  | SELECTOR                                   |
|--|---------|---------|-------|-----|------------|---|--|
| replicaset.apps/aynl-be-deployment-855df68d5d  | 1       | 1       | 1     | 27d | aynl-be    | 192.168.1.103:5000/lmg_lvc_aynl_backend:v1    | name=aynl-be,pod-template-hash=855df68d5d  |
| replicaset.apps/aynl-bot-deployment-5448445ff7 | 1       | 1       | 1     | 27d | aynl-bot   | 192.168.1.103:5000/lmg_lvc_aynl-fondef-bot:v1 | name=aynl-bot,pod-template-hash=5448445ff7 |

Figura 3.9: Apariencia final del cloud desplegado

## 4. Conclusiones

En este documento se especificó la secuencia de pasos que permite desplegar la versión base del cloud interuniversitarios para emergencias.