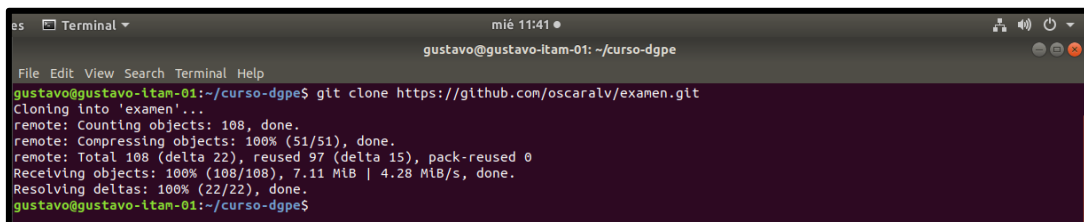

CREACIÓN Y DESPLIEGUE DE UNA VERTICAL EN UN SERVIDOR RANCHER

CREACIÓN DE LA VERTICAL CON VERTX

Para crear la vertical con el servicio REST que queremos levantar en el servidor Rancher primero debemos clonar de GitHub el proyecto “examen” en la URL

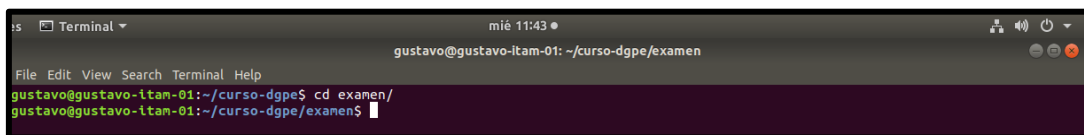
<https://github.com/oscaralv/examen.git>



```
es Terminal mié 11:41
gustavo@gustavo-itam-01: ~/curso-dgpe

File Edit View Search Terminal Help
gustavo@gustavo-itam-01:~/curso-dgpe$ git clone https://github.com/oscaralv/examen.git
Cloning into 'examen'...
remote: Counting objects: 108, done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 108 (delta 22), reused 97 (delta 15), pack-reused 0
Receiving objects: 100% (108/108), 7.11 MiB | 4.28 MiB/s, done.
Resolving deltas: 100% (22/22), done.
gustavo@gustavo-itam-01:~/curso-dgpe$
```

Una vez clonado el proyecto debemos posicionarnos en el directorio creado con el comando `cd examen/`

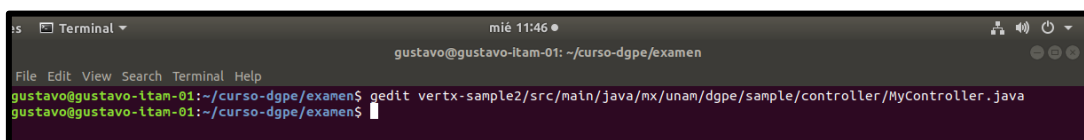


```
es Terminal mié 11:43
gustavo@gustavo-itam-01: ~/curso-dgpe/examen

File Edit View Search Terminal Help
gustavo@gustavo-itam-01:~/curso-dgpe$ cd examen/
gustavo@gustavo-itam-01:~/curso-dgpe/examen$
```

Después de posicionarnos en el directorio debemos modificar el archivo `MyController.java` ubicado en la ruta:

`vertex-sample2/src/main/java/mx/unam/dgpe/sample/controller/MyController.java`



```
es Terminal mié 11:46
gustavo@gustavo-itam-01: ~/curso-dgpe/examen

File Edit View Search Terminal Help
gustavo@gustavo-itam-01:~/curso-dgpe/examen$ gedit vertex-sample2/src/main/java/mx/unam/dgpe/sample/controller/MyController.java
gustavo@gustavo-itam-01:~/curso-dgpe/examen$
```

Una vez abierto el archivo, primero hay que modificar el método “**calculadora**” en la invocación al método “**calcula**”, agregándole como parámetro extra el **request**.

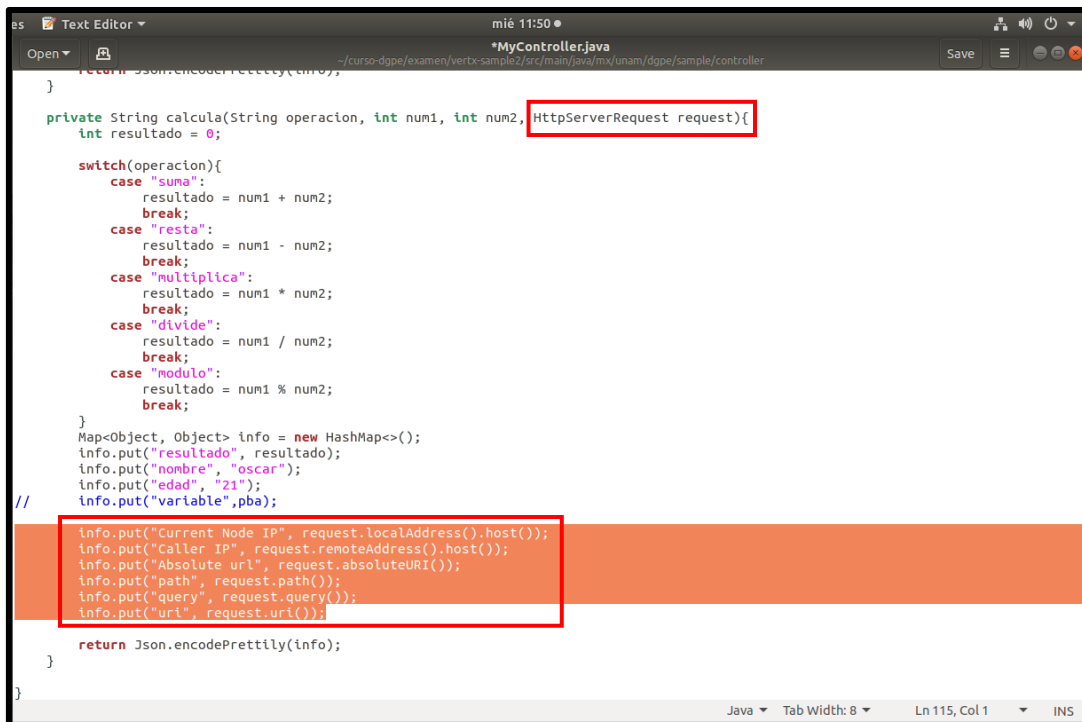


```
es Text Editor mié 11:49
MyController.java
~/curso-dgpe/examen/vertex-sample2/src/main/java/mx/unam/dgpe/sample/controller

HttpServerResponse response = routingContext.response();
String decoded = routingContext.getBodyAsString();
String jsonResponse = procesa(decoded);
response.setStatus(200);
putHeader("content-type", "application/json; charset=utf-8");
end(jsonResponse);
}

private void calculadora(RoutingContext routingContext) {
    HttpServerResponse response = routingContext.response();
    HttpRequest request = routingContext.request();
    String operacion = request.getParam("operacion");
    int valor1 = Integer.parseInt(request.getParam("valor1"));
    int valor2 = Integer.parseInt(request.getParam("valor2"));
    String jsonResponse = calcula(operacion, valor1, valor2, request);
    response.setStatus(200);
    putHeader("content-type", "application/json; charset=utf-8");
    end(jsonResponse);
}
```

Modificamos el método “calcula” para que reciba el mismo parámetro y podamos usarlo para mostrar la IP donde se encuentra corriendo el servicio. Esta información nos servirá más adelante para verificar que el balanceador de cargas en el servidor Rancher está funcionando correctamente.



```

}

return Json.encodePretty(info);
}

private String calcula(String operacion, int num1, int num2, HttpServletRequest request){
    int resultado = 0;

    switch(operacion){
        case "suma":
            resultado = num1 + num2;
            break;
        case "resta":
            resultado = num1 - num2;
            break;
        case "multiplica":
            resultado = num1 * num2;
            break;
        case "divide":
            resultado = num1 / num2;
            break;
        case "modulo":
            resultado = num1 % num2;
            break;
    }

    Map<Object, Object> info = new HashMap<>();
    info.put("resultado", resultado);
    info.put("nombre", "oscar");
    info.put("edad", "21");
    // info.put("variable", pba);

    info.put("Current Node IP", request.getLocalAddress().host());
    info.put("Caller IP", request.getRemoteAddress().host());
    info.put("Absolute url", request.getAbsoluteURI());
    info.put("path", request.getPath());
    info.put("query", request.getQuery());
    info.put("uri", request.getUri());

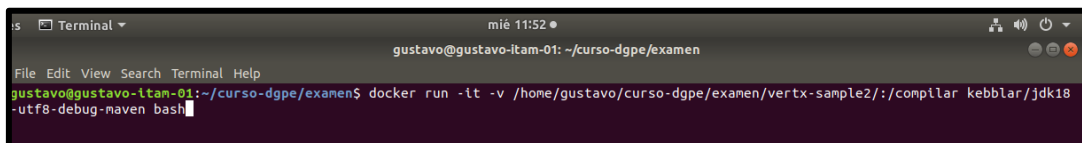
    return Json.encodePretty(info);
}
}

```

Guardamos los cambios y cerramos el archivo.

Para poder compilarlo necesitamos usar la imagen Docker **keblar/jdk18-utf8-debug-maven**. Agregamos un volumen y lo invocamos de la siguiente manera.

docker run -it -v /home/gustavo/curso-dgp/examen/vertex-sample2:/compilar keblar/jdk18-utf8-debug-maven bash



```

gustavo@gustavo-itam-01: ~/curso-dgpe/examen
gustavo@gustavo-itam-01:~/curso-dgpe/examen$ docker run -it -v /home/gustavo/curso-dgpe/examen/vertex-sample2:/compilar keblar/jdk18-utf8-debug-maven bash
root@c060be375e4a: /compilar#

```

Después de ejecutarlo nos muestra el prompt que indica que estamos dentro del Docker. Nos posicionamos en el directorio que creamos con el volumen en el comando anterior.

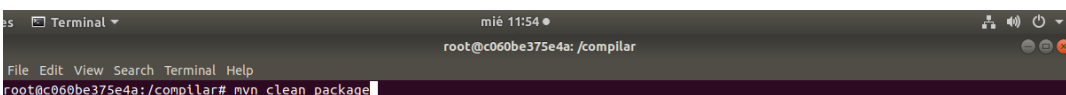


```

root@c060be375e4a: /compilar# cd compilar/
root@c060be375e4a: /compilar#

```

Dentro del directorio ejecutamos la instrucción **mvn clean package** para compilar nuestro proyecto de la vertical.



```

root@c060be375e4a: /compilar# mvn clean package

```

Oscar Alvarez Fernández

Podemos observar que la compilación se realizó correctamente.

```
Terminal
mié 11:55
root@c060be375e4a: /compilar

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ sample ---
[INFO] Building jar: /compilar/target/sample-1.0-SNAPSHOT.jar
[INFO] --- maven-shade-plugin:2.3:shade (default) @ sample ---
[WARNING] Map in class org.apache.maven.plugins.shade.resource.ManifestResourceTransformer declares value type as: class java.util.jar.Attributes but saw: class java.lang.String at runtime
[WARNING] Map in class org.apache.maven.plugins.shade.resource.ManifestResourceTransformer declares value type as: class java.util.jar.Attributes but saw: class java.lang.String at runtime
[INFO] Including log4j:log4j:jar:1.2.17 in the shaded jar.
[INFO] Including io.vertx:vertx-core:jar:3.3.3 in the shaded jar.
[INFO] Including io.netty:netty-common:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-buffer:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-transport:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-handler:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-codec:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-handler-proxy:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-codec-socks:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-codec-http:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-codec-http2:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-resolver:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-resolver-dns:jar:4.1.5.Final in the shaded jar.
[INFO] Including io.netty:netty-codec-dns:jar:4.1.5.Final in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-core:jar:2.7.4 in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-databind:jar:2.7.4 in the shaded jar.
[INFO] Including com.fasterxml.jackson.core:jackson-annotations:jar:2.7.0 in the shaded jar.
[INFO] Including io.vertx:vertx-web:jar:3.2.1 in the shaded jar.
[INFO] Including io.vertx:vertx-auth-common:jar:3.2.1 in the shaded jar.
[INFO] Including org.apache.httpcomponents:httpclient:jar:4.5.2 in the shaded jar.
[INFO] Including org.apache.httpcomponents:httpcore:jar:4.4.4 in the shaded jar.
[INFO] Including commons-logging:commons-logging:jar:1.2 in the shaded jar.
[INFO] Including commons-codec:commons-codec:jar:1.9 in the shaded jar.
[INFO] BUILD SUCCESS
[INFO] Total time: 17.989 s
[INFO] Finished at: 2018-06-13T16:55:05Z
[INFO] Final Memory: 43M/182M
[INFO]
```

Con el comando **exit** salimos del contenedor Docker.

```
Terminal
mié 11:57
gustavo@gustavo-itam-01: ~/curso-dgpe/examen

root@c060be375e4a: /compilar# exit
exit
gustavo@gustavo-itam-01:~/curso-dgpe/examen$
```

La compilación nos creó un directorio llamado **target** dentro del directorio **vertx-sample2**. Dentro de este directorio (**target**) fue generado un archivo **jar** llamado **sample-1.0-SNAPSHOT-fat.jar**. Copiamos ese archivo a la raíz de nuestro proyecto “examen” para poder usarlo más adelante con el archivo **Dockerfile**.

```
Terminal
mié 12:36
gustavo@gustavo-itam-01: ~/curso-dgpe/examen

gustavo@gustavo-itam-01:~/curso-dgpe/examen$ cp vertx-sample2/target/sample-1.0-SNAPSHOT-fat.jar .
gustavo@gustavo-itam-01:~/curso-dgpe/examen$ ll
total 7884
drwxr-xr-x 4 gustavo gustavo 4096 jun 13 12:33 ./
drwxr-xr-x 9 gustavo gustavo 4096 jun 13 11:41 ../
-rwxr-xr-x 1 gustavo gustavo 1638 jun 13 11:41 bombardeo.sh*
-rwxr-xr-x 1 gustavo gustavo 146 jun 13 12:32 Dockerfile*
-rwxr-xr-x 1 gustavo gustavo 55 jun 13 11:41 entry.sh*
drwxr-xr-x 8 gustavo gustavo 4096 jun 13 11:41 .git/
-rw-r--r-- 1 gustavo gustavo 11357 jun 13 11:41 LICENSE
-rw-r--r-- 1 gustavo gustavo 2019 jun 13 11:41 README.md
-rw-r--r-- 1 gustavo gustavo 8027279 jun 13 12:36 sample-1.0-SNAPSHOT-fat.jar
drwxr-xr-x 6 gustavo gustavo 4096 jun 13 11:55 vertx-sample2/
gustavo@gustavo-itam-01:~/curso-dgpe/examen$
```

Creamos un archivo llamado “**entry.sh**” que será usado mas adelante por el **Dockerfile**. Este archivo contiene instrucciones que se ejecutaran cuando la imagen creada sea inicializada. No antes.

Creamos un archivo llamado **Dockerfile** con las siguientes instrucciones:

INSTRUCCIÓN	ACCIÓN
FROM gustavoarellano/jdk18	Nos indica que imagen tomaremos como base para crear la nuestra.
RUN apt-get update	Ejecuta el comando de actualización de las dependencias del SO.
COPY sample-1.0-SNAPSHOT-fat.jar /home	Se copia el archivo jar que creamos al directorio /home de nuestra imagen que se va crear.
COPY entry.sh /home/entry.sh	Copiamos el archivo entry.sh creado anteriormente al directorio /home de nuestra imagen que se va crear.
ENTRYPOINT ["/home/entry.sh"]	Ejecuta las instrucciones en el archivo entry.sh copiado anteriormente.

Con los archivos listos ejecutamos el comando **docker build . -t oscaralv/examen02** para crear la imagen nueva llamada “**oscaralv/examen02**” con la información contenida en el archivo **Dockerfile**.

Oscar Alvarez Fernández

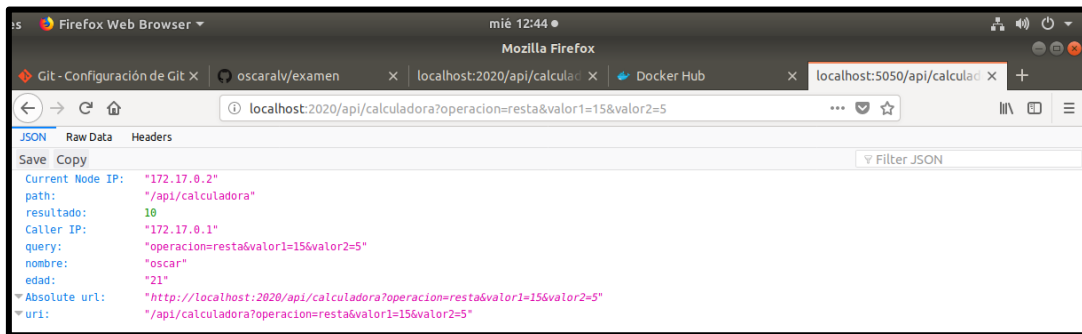
Levantamos un contenedor Docker con la imagen ya creada para poder verificar que funcione adecuadamente.

```
gustavo@gustavo-ltam-01: ~/curso-dgpe/examen
gustavo@gustavo-ltam-01:~/curso-dgpe/examen$ docker run -d -p 2020:8080 oscarlv/examen02
c3559d7270da564726414b54e4f1ccd34250757d13cebc4c8db97c006c1e067a
gustavo@gustavo-ltam-01:~/curso-dgpe/examen$
```

En una ventana del navegador llamamos a nuestra vertical con:

localhost:2020/api/calculadora?operacion=suma&valor1=15&valor2=5

Observamos que la información es correcta.



Procedemos a subir la imagen a nuestra cuenta de Docker Hub con el comando

docker push oscarlv/examen02

Recordemos que para poder subirla correctamente debe llevar el formato

nombre-usuario-docker/nombre-imagen

De no ser así marcará un error.

```
gustavo@gustavo-ltam-01: ~/curso-dgpe/examen
gustavo@gustavo-ltam-01:~/curso-dgpe/examen$ docker push oscarlv/examen02
The push refers to repository [docker.io/oscarlv/examen02]
b95ace2be01e: Pushed
03200c6c4807: Pushed
687a870715f2: Mounted from oscarlv/prueba_exam
0dd2d2815662: Mounted from oscarlv/prueba_exam
5f70bf18a086: Mounted from oscarlv/prueba_exam
918dbf1cf3de: Mounted from oscarlv/prueba_exam
9e1fe90ee292: Mounted from oscarlv/prueba_exam
d97fd2c5d8e1: Mounted from oscarlv/prueba_exam
c1cc34424286: Mounted from oscarlv/prueba_exam
latest: digest: sha256:b7edf64f62fbd60ab6c00cf3cdf1c92deafcc3af3374ff96036baf5b38cb446e size: 2198
gustavo@gustavo-ltam-01:~/curso-dgpe/examen$
```

INSTALACIÓN DE RANCHER

Ejecutamos en una maquina el siguiente comando

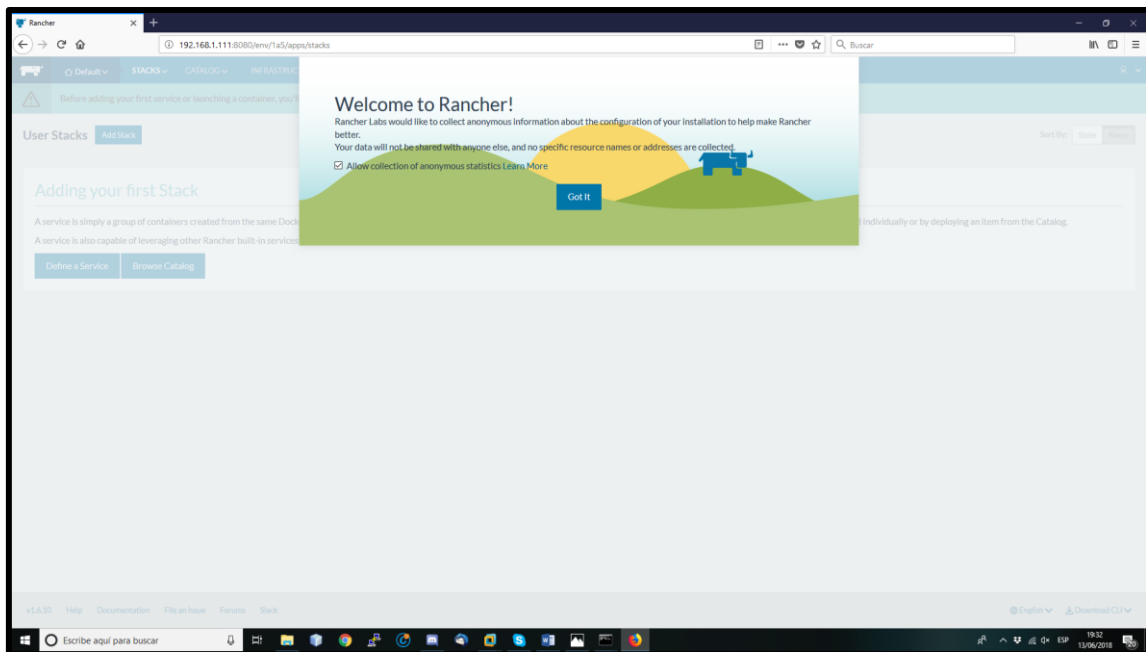
```
gustavo@ubuntu: ~
gustavo@ubuntu:~$ docker run -d -p 8080:8080 kebbiar/rancher
```

Verificamos que se haya levantado correctamente.

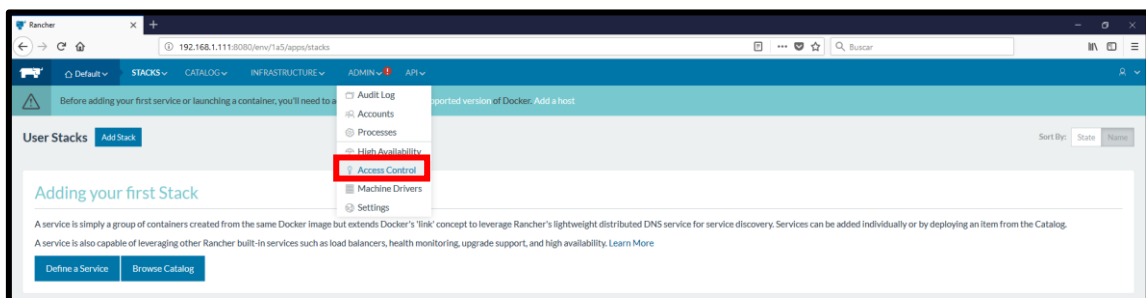
```
gustavo@ubuntu: ~
gustavo@ubuntu:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED            STATUS              PORTS               NAMES
9055f780909b      kebbiar/rancher    "/usr/bin/entry /u..."  8 seconds ago     Up 5 seconds       3306/tcp, 0.0.0.0:8080->8080/tcp   distracted_shockley
```

Oscar Alvarez Fernández

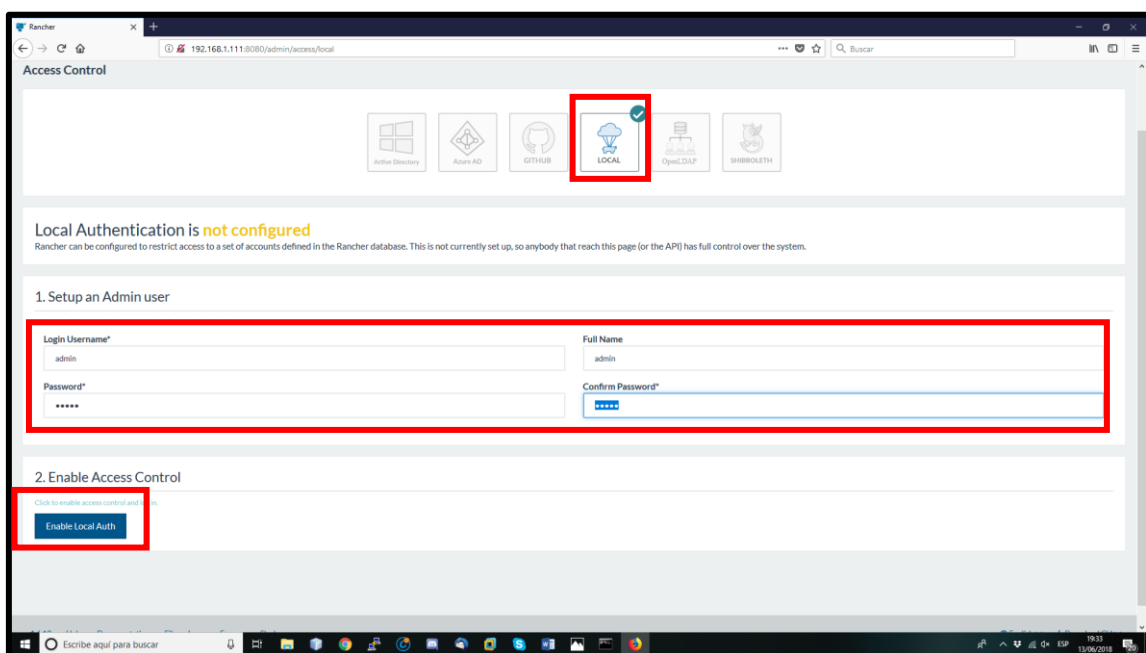
En el navegador entramos a la administración del servidor Rancher.



Nos dirigimos a la pestaña “ADMIN” y luego a “Access Control”.

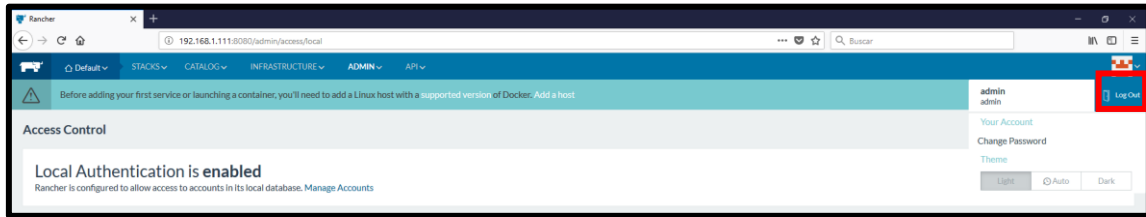


Seleccionamos “LOCAL” e ingresamos los datos para autenticarnos. “Enable Local Auth”.

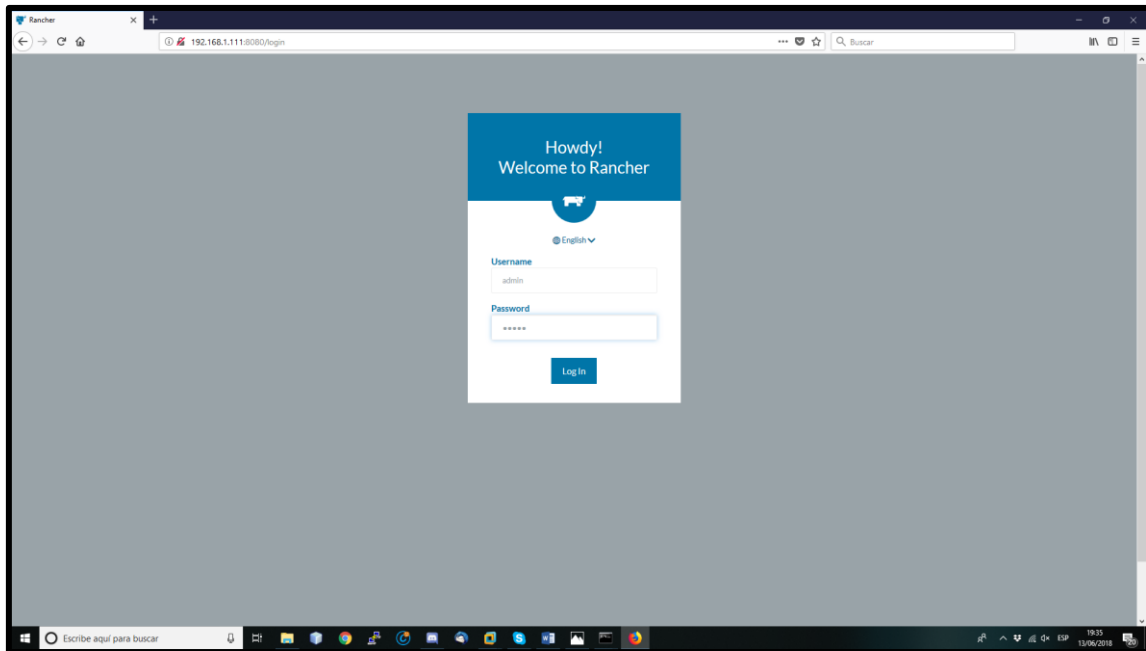


Oscar Alvarez Fernández

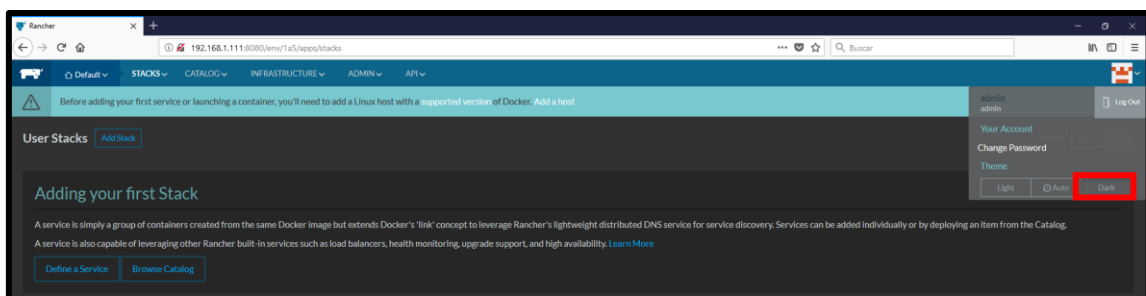
Hacemos click en “**Log Out**” en la parte superior derecha.



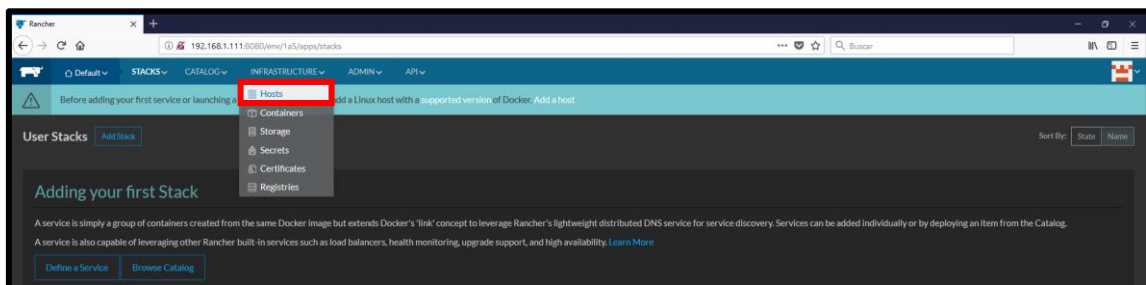
Ingresamos con los datos que registramos anteriormente.



En la pestaña superior derecha elegimos el tema “**Dark**”. Paso vital para el correcto funcionamiento del servidor Rancher.

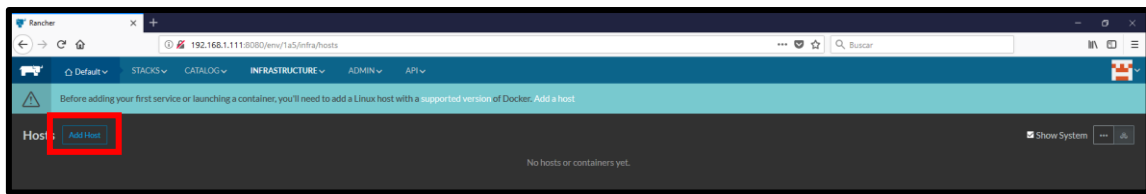


Nos dirigimos a la pestaña “**INFRASTRUCTURE**” – “**Hosts**”.

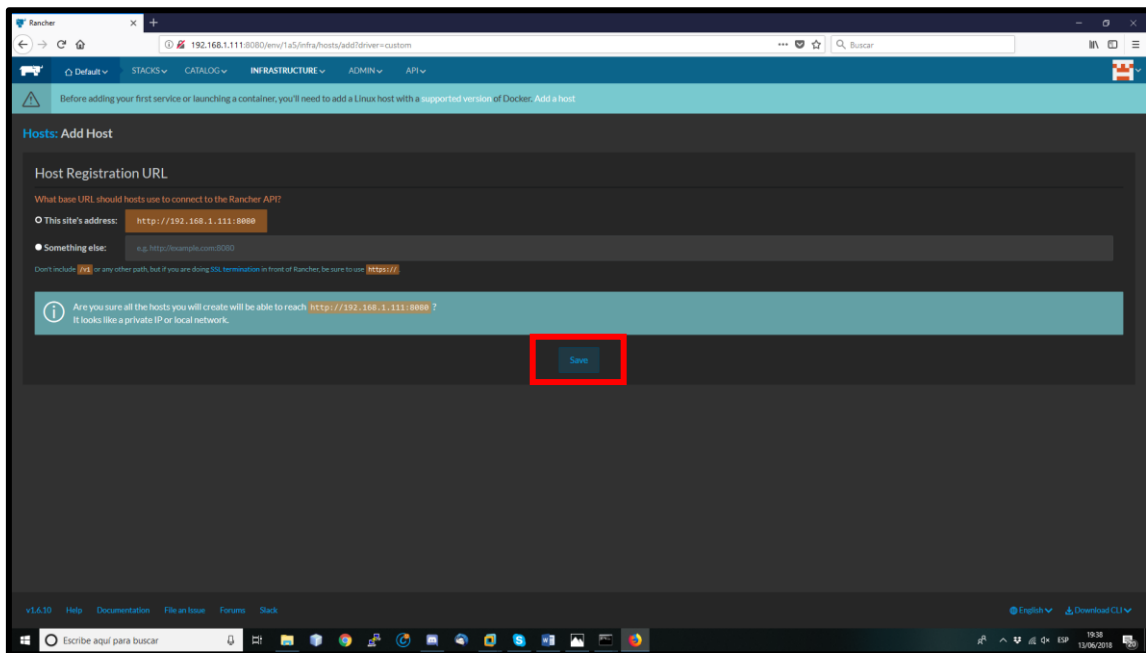


Oscar Alvarez Fernández

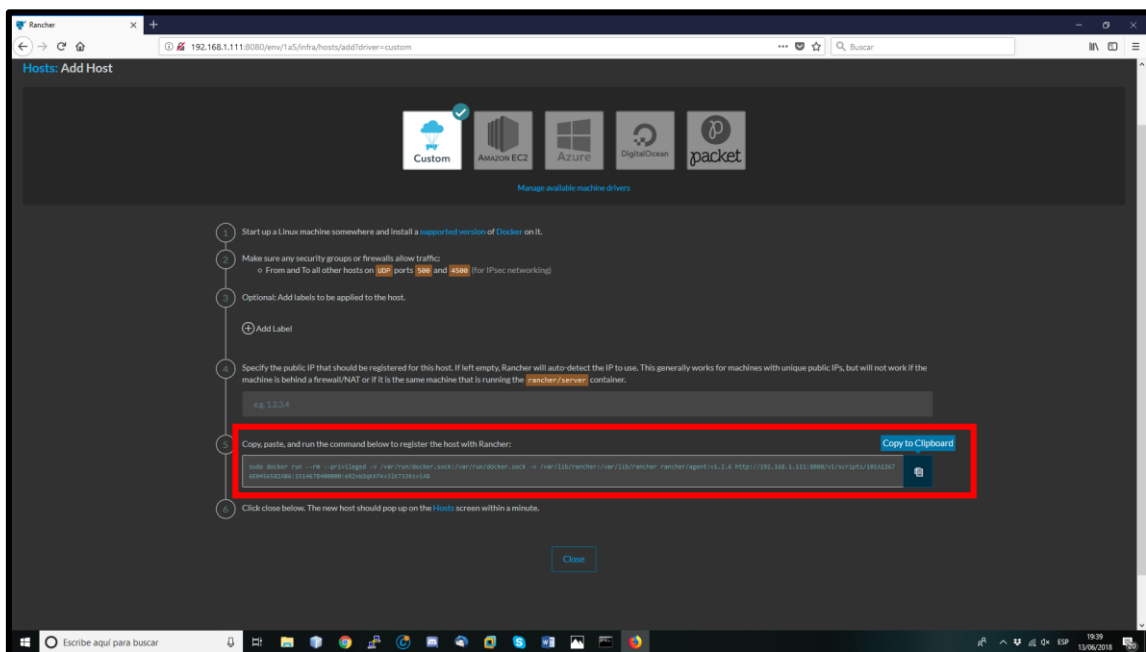
Seleccionamos **“Add Host”**.



Damos click en **“Save”**.



Damos click en **“Copy to Clipboard”** para copiar el comando que ejecutarán los servidores clientes para levantar el agente y conectarse al servidor Rancher.



Oscar Alvarez Fernández

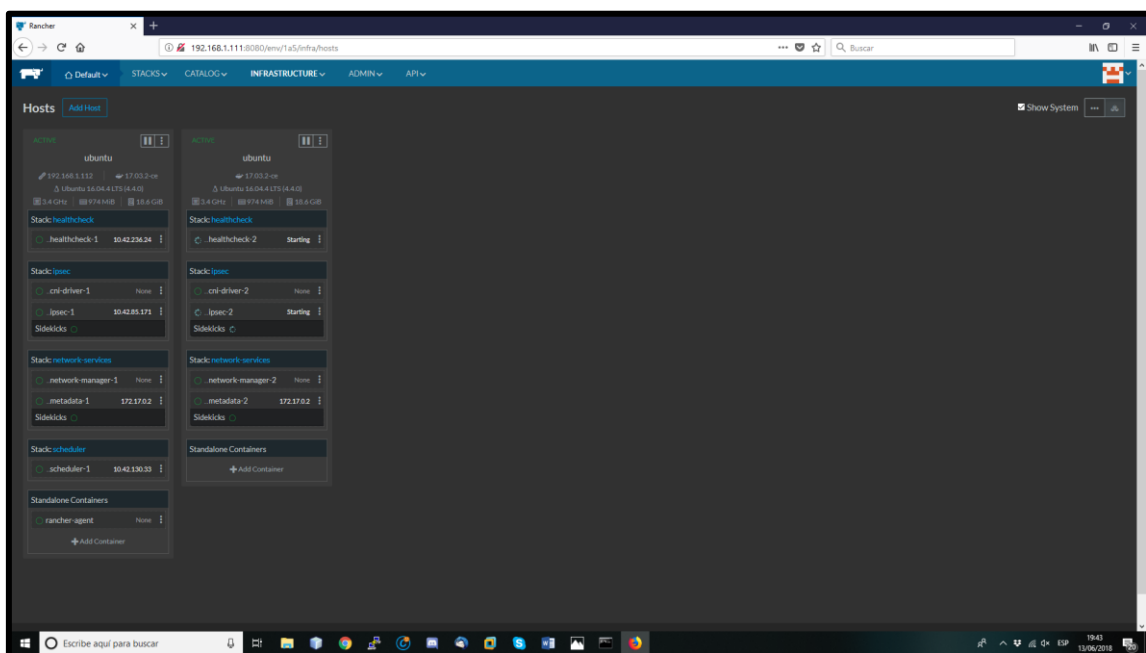
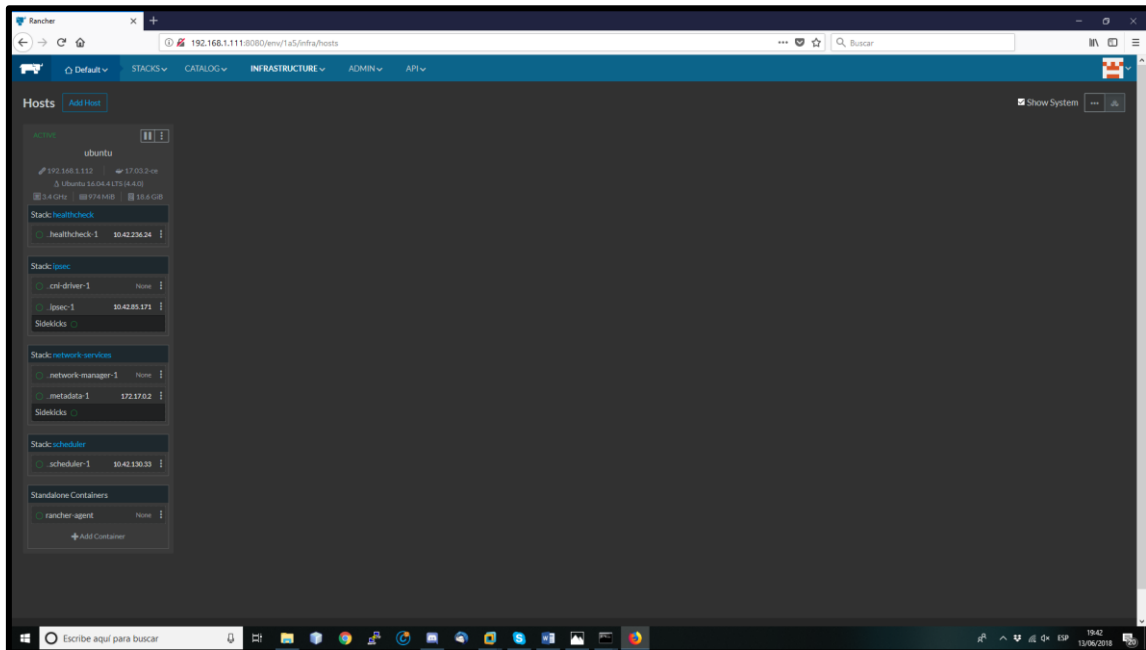
Ejecutamos el comando copiado en ambos clientes.

```

guitav@ubuntu:~$ sudo docker run --rm --privileged -v /var/run/docker.sock:/var/run/docker.sock -v /var/lib/rancher:/var/lib/rancher rancher/agent:v1.2.6 http://192.168.1.111:8080/v1/scripts/101A1267604056582A86:1514678400000:ef243d9c-ef391d73335d4d
INFO: Running Agent Registration Process, CATTLE_URL=http://192.168.1.111:8080/v1
INFO: Attempting to connect to: http://192.168.1.111:8080/v1
INFO: http://192.168.1.111:8080/v1 is accessible
INFO: Inspecting host capabilities
INFO: Boot2Docker: false
INFO: Host writable: true
INFO: Token: xxxxxxxx
INFO: Running registration
INFO: Printing Environment
INFO: ENV: CATTLE_ACCESS_KEY=7f8f398b3a7a3684f27
INFO: ENV: CATTLE_HOME=/var/lib/cattle
INFO: ENV: CATTLE_REGISTRATION_ACCESS_KEY=registrationToken
INFO: ENV: CATTLE_REGISTRATION_SECRET_KEY=xxxxxxxx
INFO: ENV: CATTLE_SECRET_KEY=xxxxxxxx
INFO: ENV: CATTLE_URL=http://192.168.1.111:8080/v1
INFO: ENV: DETECTED_CATTLE_AGENT_IP=192.168.1.112
INFO: ENV: RANCHER_AGENT_IMAGE=rancher/agent:v1.2.6
INFO: Launched Rancher agent: f4a0b1d728f1c36be4d6eeef8345a6f415d6f1a4fa42e17614812be28c51c
guitav@ubuntu:~$

```

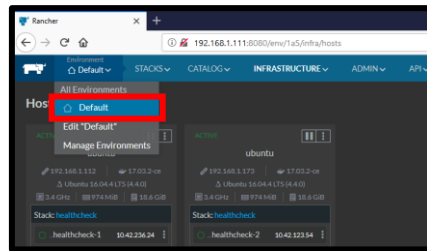
Verificamos que en el servidor aparecen los clientes que se van conectando



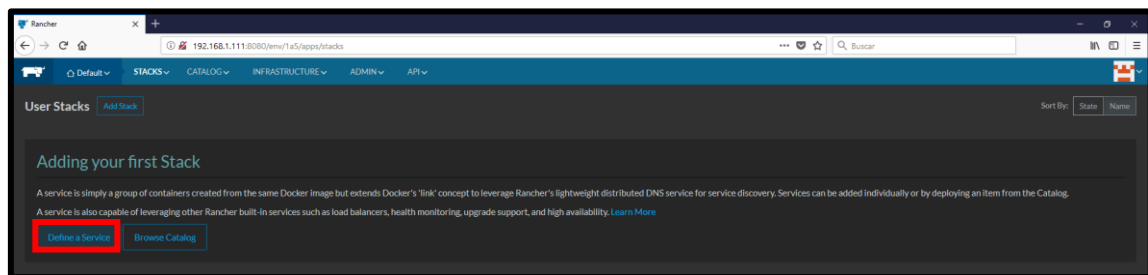
Oscar Alvarez Fernández

CREACIÓN DE UN SERVICIO EN RANCHER

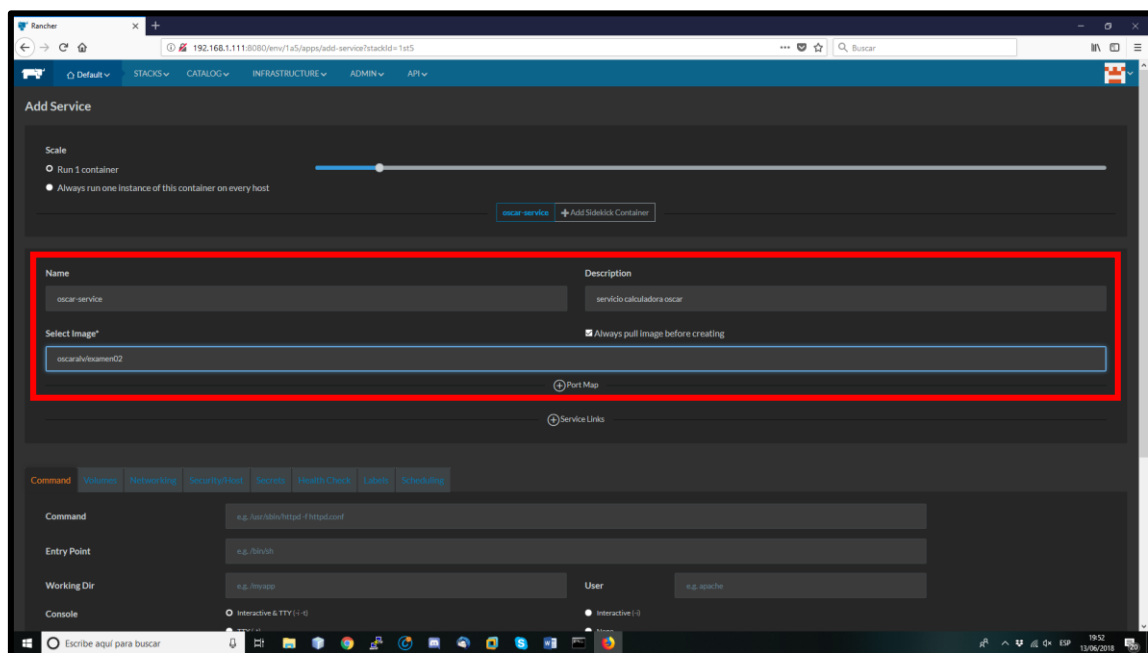
Damos click en la pestaña “Default – “Default”.



Seleccionamos “Define a Service”.

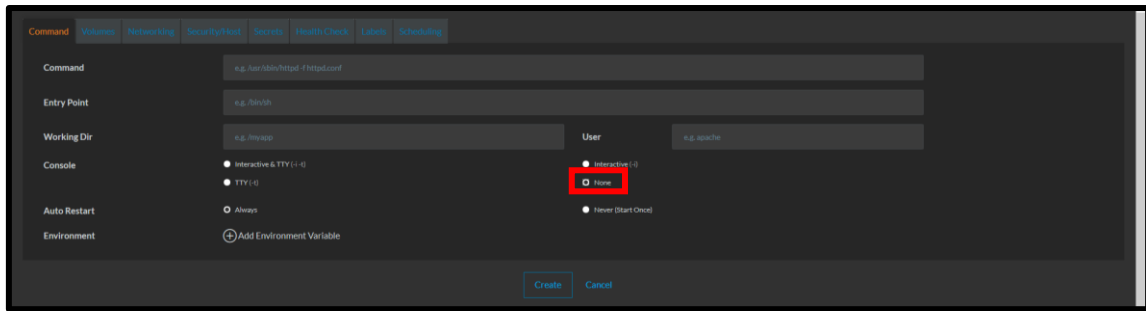


Ingresamos el nombre del servicio, una descripción y la imagen que vamos a utilizar para levantar el servicio. En este caso es la imagen creada con anterioridad, **oscaralv/examen02**.

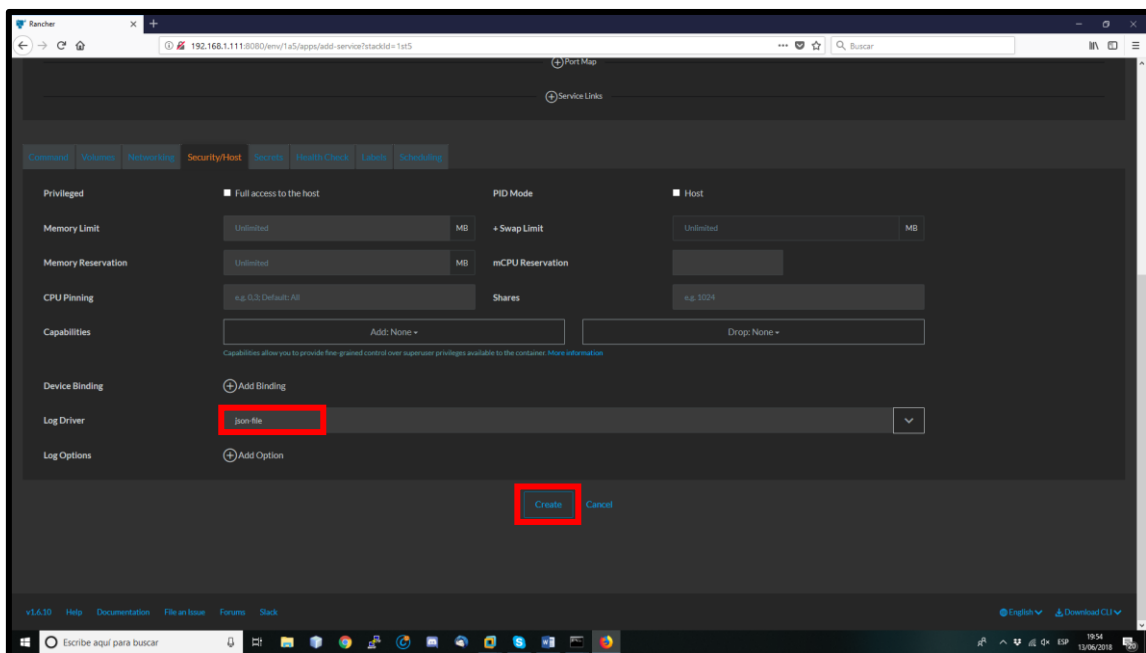


Oscar Alvarez Fernández

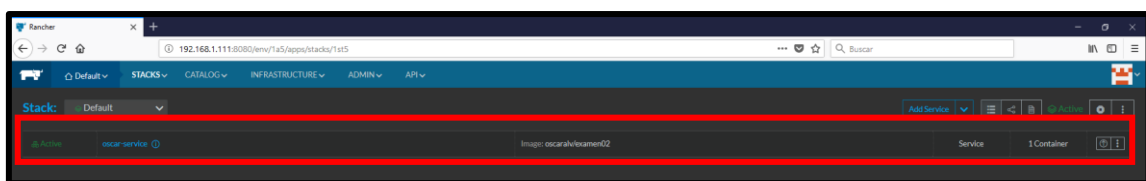
En la parte inferior seleccionamos en la sección **“Console”** la opción **“None”**.



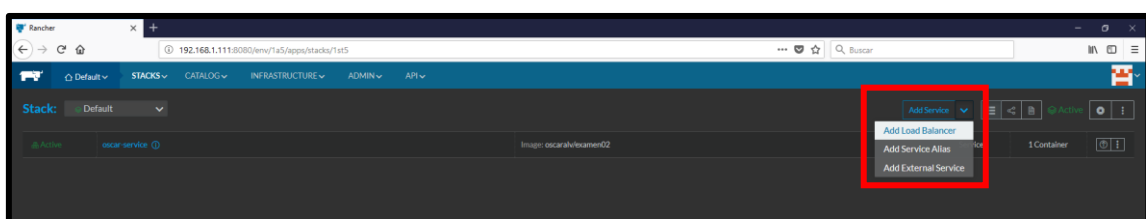
Cambiamos a la pestaña **“Security/Host”** y en la sección **“Log Driver”** elegimos la opción **“json-file”**. Damos click en **“Create”**.



Abrimos de nuevo la pestaña **“Default”** - **“Default”** y vemos que nuestro servicio ya aparece activo.

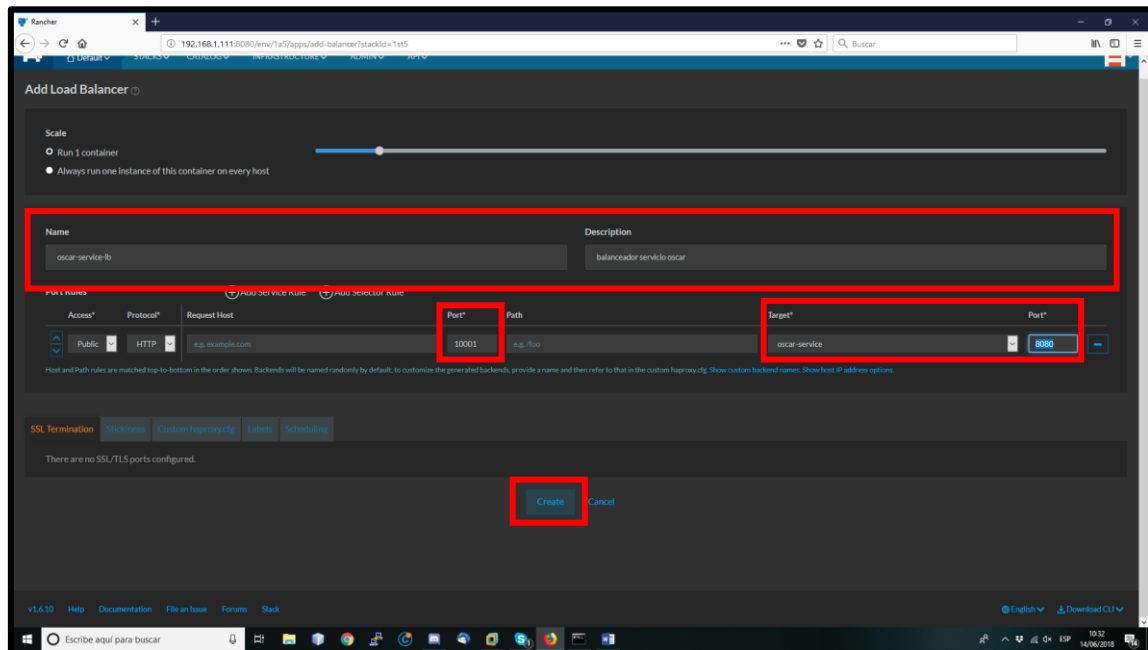


En la parte superior derecha desplegamos el menú **“Add Service”** y seleccionamos la opción **“Add Load Balancer”**.

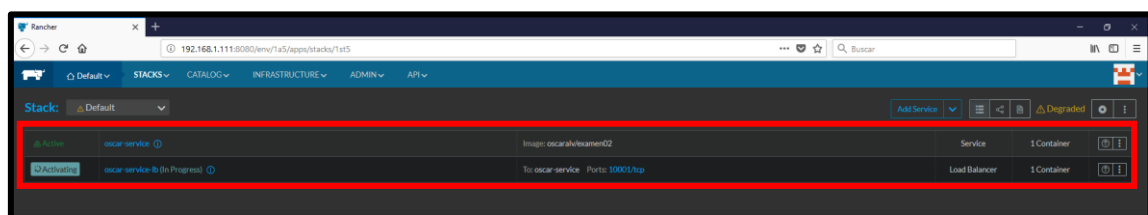


Oscar Alvarez Fernández

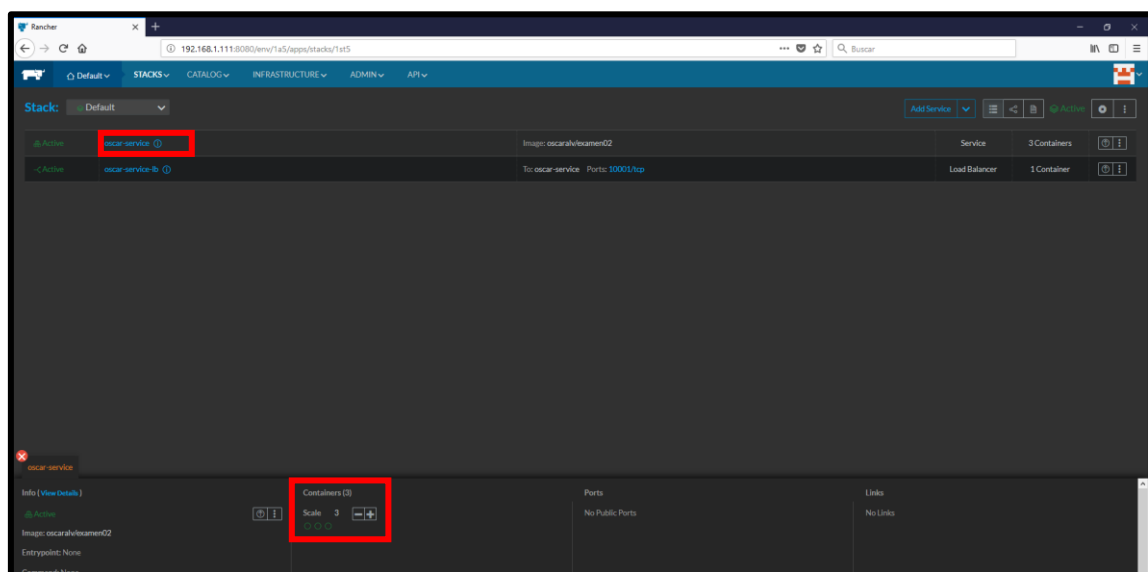
Definimos un nombre para el balanceador, una descripción, el puerto donde se va desplegar, el servicio que va balancear y el puerto que usa ese servicio internamente. Damos click en “Create”.



Vemos que en la pestaña default aparece el balanceador junto a nuestro servicio. Debemos esperar a que se despliegue y quede en estado activo.

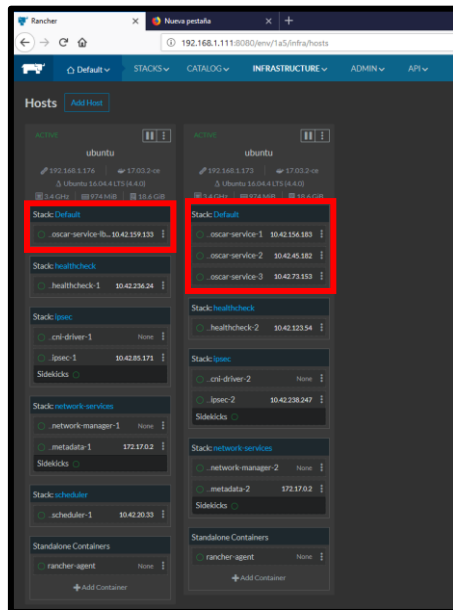


Seleccionamos la “i” junto al nombre del servicio y en la parte inferior en el segmento “Containers” aumentamos la escala a 3.

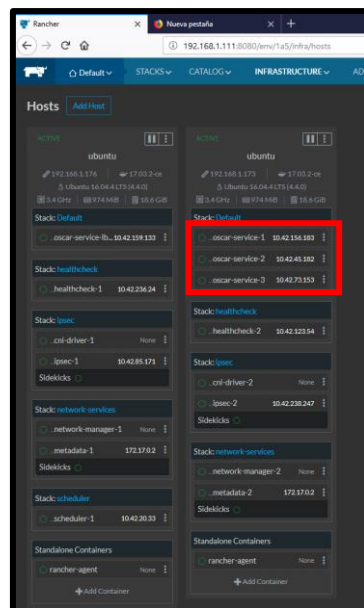


Oscar Alvarez Fernández

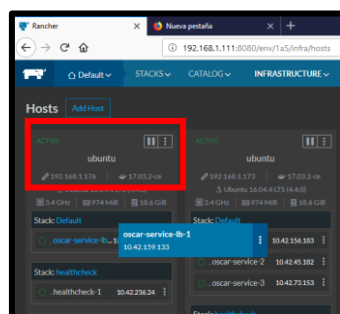
Nos dirigimos al menú “**INFRASTRUCTURE**”- “**Hosts**” y verificamos que nuestro servidor ha distribuido las instancias y el balanceador entre los 2 servidores agentes creados.



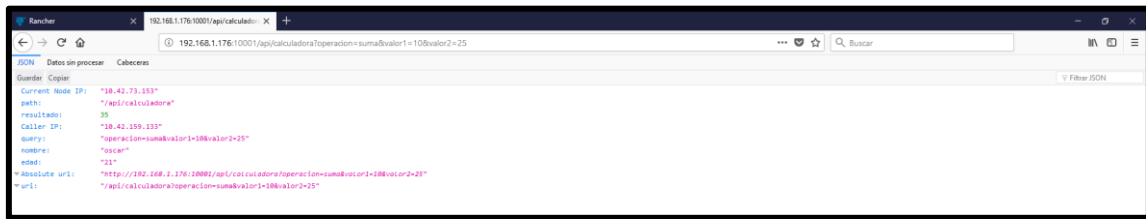
Observamos que nuestras 3 instancias del servicio tienen asignadas una IP distinta cada una.



Observamos que nuestro balanceador esta corriendo en la maquina **192.168.1.176**



En una ventana del navegador ejecutamos nuestra vertical llamando al balanceador con la URL <http://192.168.1.176:10001/api/calculadora?operacion=suma&valor1=10&valor2=25>



Al refrescar varias veces podemos observar que la IP donde se encuentra el servicio va cambiando. Esto nos indica que el balanceador está funcionando correctamente.

