



TRABAJO FINAL DE ASIGNATURA

Despliegue de un clúster de Kubernetes con Rancher en OpenStack, Google Cloud y Azure, implementando una topología de red de una empresa con distintos servidores.

Autores: D. Aarón Blanco Álvarez.
D. Ángel Luis Labraca Miranda.
D. Francisco José López Carrillo.

Profesores: D. José Antonio Martínez García.
D. Juan Francisco Sanjuan Estrada.

Grado en Ingeniería Informática.

Administración de Redes y Sistemas Operativos.

Escuela Superior de Ingeniería
UNIVERSIDAD DE ALMERÍA
Curso Académico: 2020/2021
Almería, 18 de enero de 2021

Índice

Tabla de Contenidos

Índice	1
1. Introducción.....	2
2. Preparación de cosas generales.	3
• ¿Qué es Rancher?.....	3
• ¿Qué es Kubernetes?.....	4
• ¿Qué es Fleet?.....	6
• Git en nuestro proyecto.....	8
• Preparación de las imágenes	8
• Despliegue de servicios	11
• Volúmenes	20
• LoadBalancers	24
• DNS	26
3. Creación de MV y despliegue de Clústeres.....	27
• OpenStack.....	27
• Despliegue en Google Cloud	32
• Despliegue en Azure	43
4. Conclusión.....	59
5. Bibliografía.....	60

1. Introducción

A la hora de realizar este trabajo, hemos estado planteando diversas opciones para elegir un tema qué pudiera agrupar todo el contenido qué hemos dado en la asignatura y, sin embargo, poder profundizar en algunos temas e incluso conocer nuevos.

De ahí qué hemos elegido una temática relacionada con Kubernetes, una plataforma de código abierto qué sabemos que es algo muy usado por las empresas de todo el mundo.

Conforme fuimos interesándonos por el tema, nos dimos cuenta de que para un caso como el nuestro en el qué hemos querido usar varios proveedores de nube con distintos usuarios, necesitábamos un orquestador, Rancher.

Así, a la hora de elegir nuestro trabajo, hemos optado por usar estas tecnologías, algunas punteras como Kubernetes qué tienen mucho que ofrecer.

Por otra parte, pensamos en darle un uso práctico, así que hemos querido juntar el despliegue con una implementación de topología de una empresa al uso. La idea parte de qué hoy en día todo funciona mediante servicios en la nube, así que no hay mejor ejemplo qué una implementación de una infraestructura de servicios de una empresa en la misma nube.

Esto por supuesto no deja de ser un ejemplo de implementación de distintos servicios, pero por supuesto, este trabajo pretende llegar a ser una guía qué pueda ayudar a tener una ligera idea de cómo plantear una infraestructura, ya sea en los proveedores de nubes qué usamos o en otros distintos. Por supuesto, podríamos decir lo mismo de los servicios qué desplegamos, ya queda en manos del usuario, la configuración del despliegue que vea conveniente.

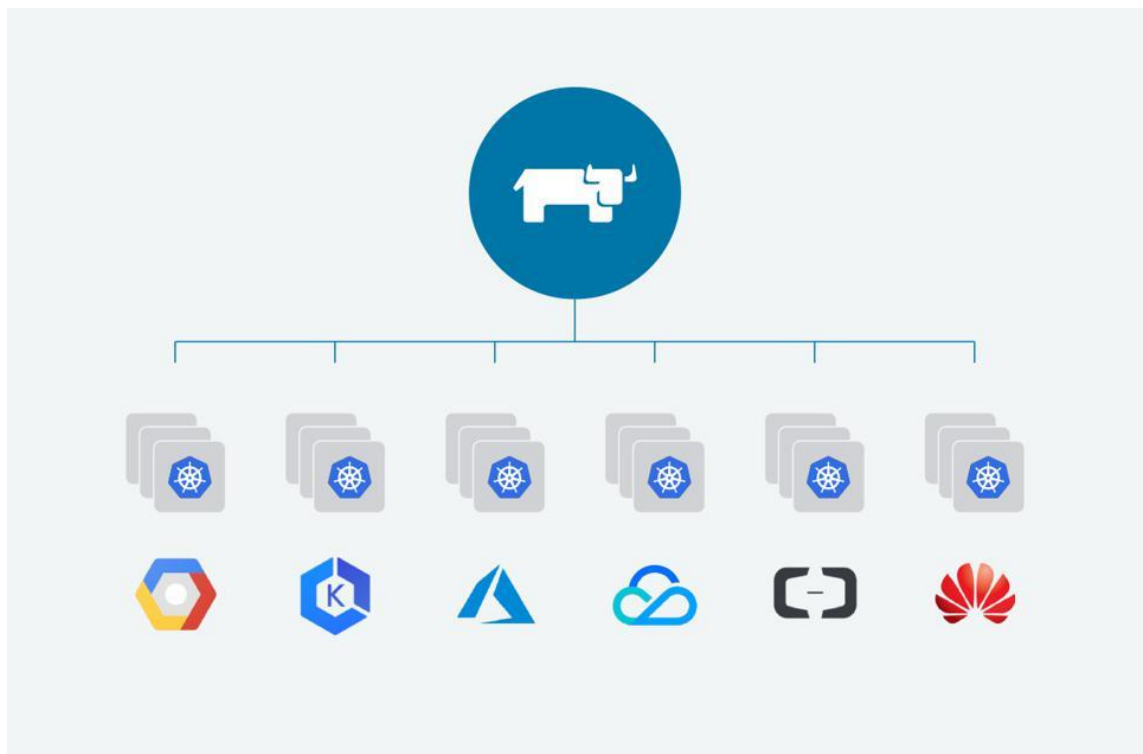
Esperamos que le sea de utilidad e interesante.

2. Preparación de cosas generales.

- ¿Qué es Rancher?

[Rancher](#) es un sistema software que tiene como fin gestionar paquetes de Clústeres, estos clústeres son hosts y nodo, que automáticamente al crearse insertan contenedores de Kubernetes para poder ser utilizados en el despliegue de servicios.

Desde Rancher podremos desplegar de una forma muy sencilla servicios a través de la herramienta Fleet, donde podremos controlar los servicios, tal como se mostraría en la imagen siguiente.



Para crear clústeres en Rancher utilizaremos las plantillas, dependiendo del proveedor con el que tratemos tendremos que utilizar plantillas RKE (OpenStack), plantillas de nodos (Azure) o configuración instantánea en la creación de Clústeres (GCloud). Dependiendo del entorno necesitaremos unos datos y unos permisos, estos serán detallados más adelante en el documento.

En este momento, la versión de Rancher que utilizamos es la v2.5.3.

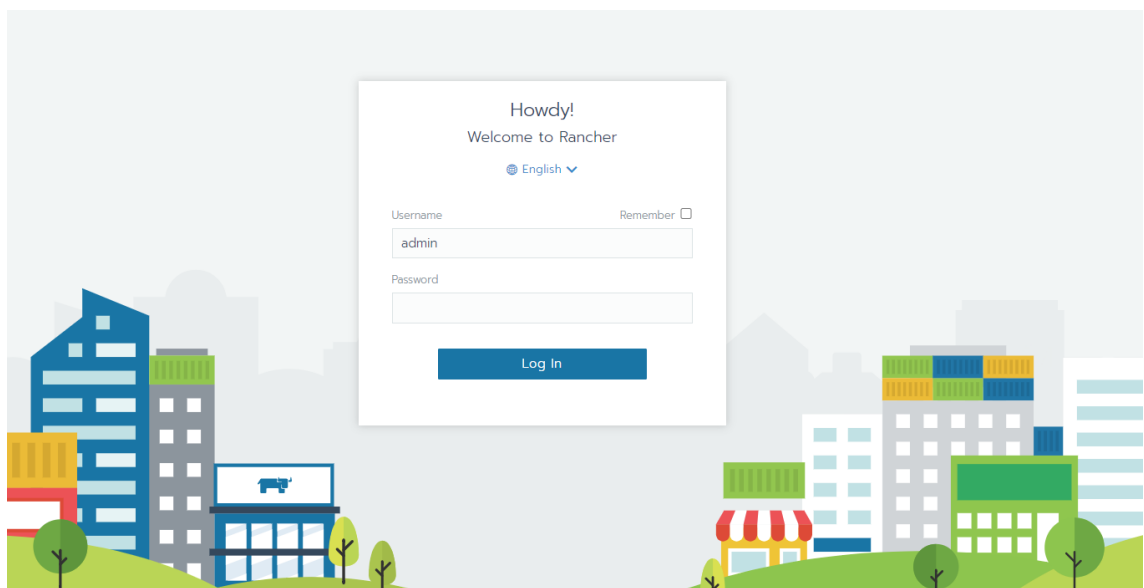
A día 20 de diciembre de 2020 que fue cuando empezamos a realizar el proyecto, Docker tenía la versión 20.01, pero Rancher todavía no era compatible con esta, por lo tanto, la versión de Docker utilizada fue la 19.03.

La instalación de Rancher la vamos a realizar siempre en una máquina y vamos a desplegarla a través de Docker, con el comando:

```
sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
```

Y entonces podremos acceder mediante el puerto 80 del navegador con la IP de la MV a la página principal de Rancher, donde nos pedirá que introduzcamos una contraseña y aceptemos los permisos y condiciones de uso.

Una vez creada la contraseña, cada vez que accedamos se nos pedirá un usuario y la contraseña, por defecto, el usuario administrador es “admin”



- ¿Qué es Kubernetes?

Kubernetes es una plataforma de código abierto para facilitar la automatización y el despliegue de servicios o contenedores en un entorno. Su uso puede ser una ayuda para el despliegue en diferentes hosts y que puede hacer que una aplicación sea introducida y extendida o reducida de una manera fácil y óptima, por ejemplo, el despliegue de una página web, que en un momento determinado necesita que destinemos más recursos a ella por una alta llegada de carga, o que necesitemos que, si hay algún imprevisto en uno de los hosts, no se caiga todo nuestro despliegue. En resumen, Kubernetes es como ellos describen, un “orquestador” y como tal, va a orquestar a nuestros servicios a otro nivel.

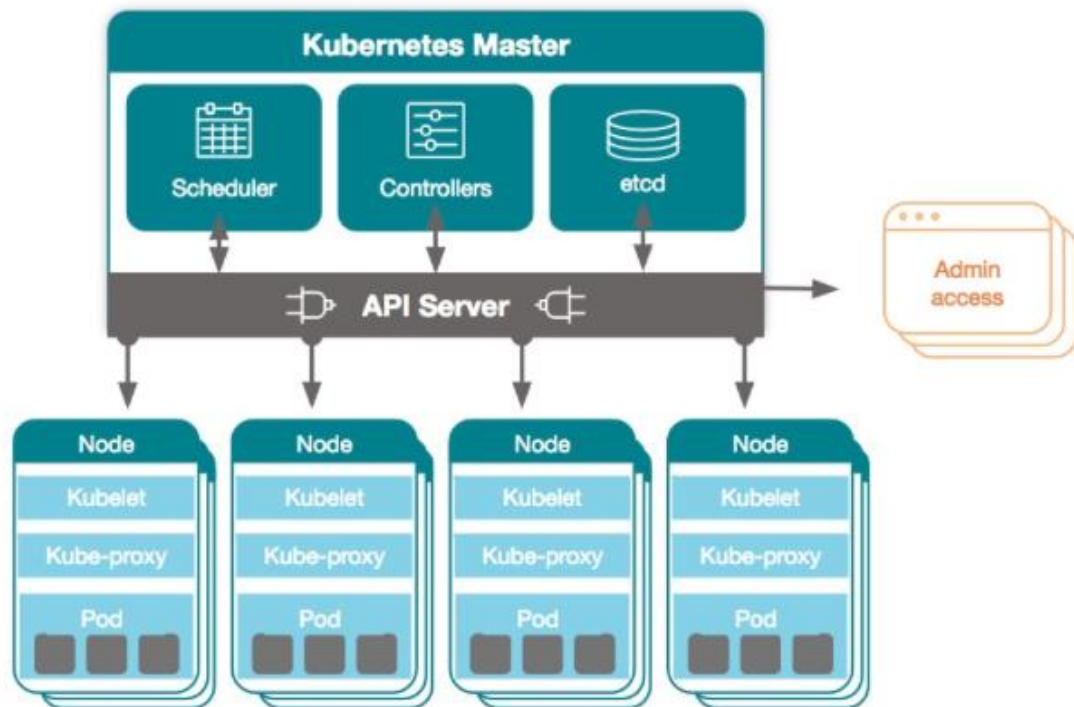
Kubernetes lo usamos en este proyecto con Docker, pero hay muchos más tipos de Contenedores como [Containerd](#), [Rkt container](#) o [Linux Container](#).

En este proyecto utilizamos la versión 1.17, 1.18 y 1.19 dependiendo del Proveedor que utilicemos.

Kubernetes se compone de unos componentes que van a permitir llevar una jerarquía en la gestión de los nodos, tendremos por un lado el **Control de Plano** que se compone de un servidor de Api (Kube-apiserver) que se ocupa de un servidor que recibe las peticiones de la API, un Almacén de datos persistente (Etcd) se ocupa de almacenar toda la información de los clústeres, un Kube-scheduler, que es quien está pendiente de nos *pods* para asignarlos a los nodos que corresponda si no tienen uno ya asignado, kube-controller-manager que gestiona los controladores que tiene Kubernetes y un cloud-controller-manager que su cometido es gestionar los controladores que se comunican con los proveedores de la nube.

Un mapa de cómo funciona Kubernetes lo podemos ver en la siguiente imagen.

Kubernetes Architecture

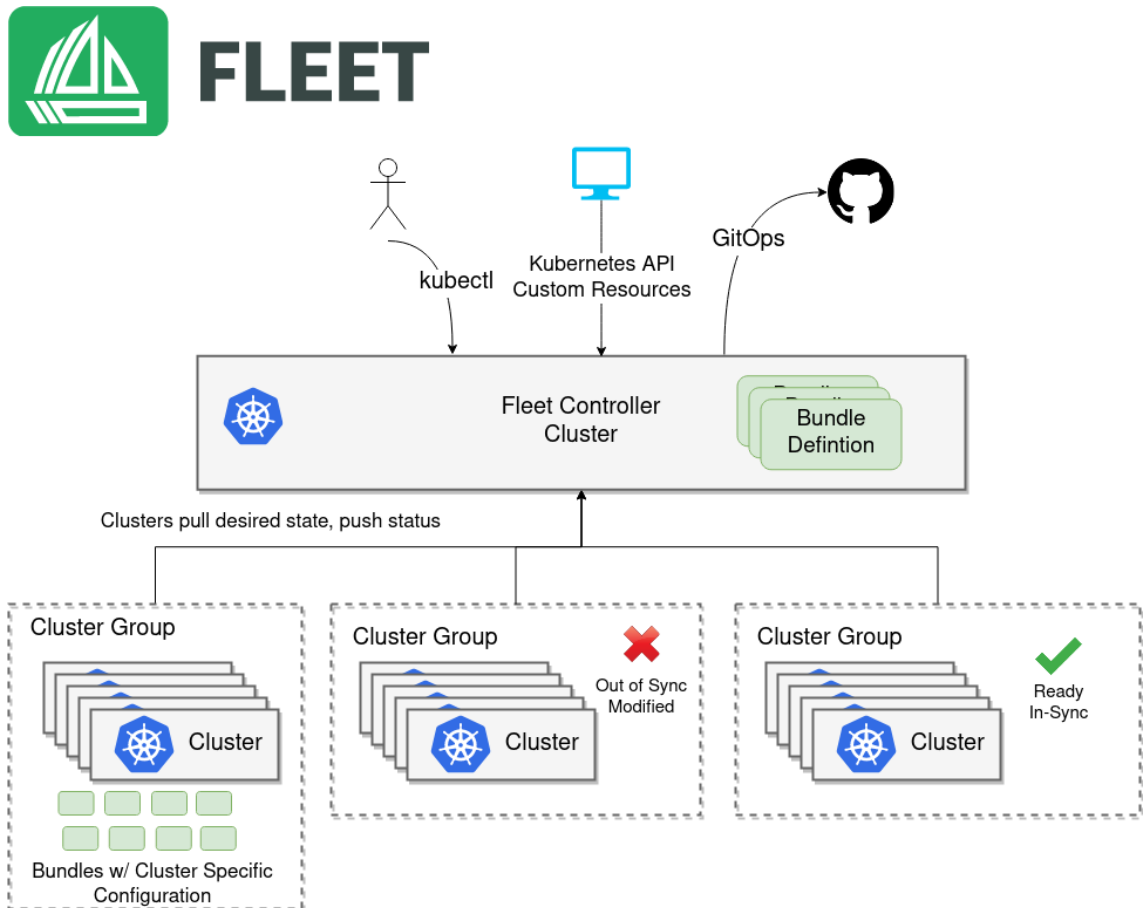


A parte de estas cosas, Kubernetes tiene su propio proxy para no comprometer los puertos del nodo anfitrión, un agente que se ocupa de la gestión de cada nodo (Kubelet) y es capaz redireccionar los procesos mediante DNS a través del Addons correspondiente.

- ¿Qué es Fleet?

Fleet es un “administrador” de clústeres, según sus desarrolladores, es capaz de controlar hasta un millón de clústeres, aunque nosotros actualmente no podemos probarlo.

Lo que podemos hacer con esto es cargar unos archivos .yaml en un repositorio git y así de una forma sencilla desplegar nuestros servicios rápidamente en grupo de clústeres. El comportamiento es el siguiente:



Nosotros tenemos todos nuestros despliegues en:

<https://github.com/Labraca/ARSO.git>

Lo tenemos estructurado de la siguiente manera:

- Tenemos 3 ramas: Una para cada proveedor, porque cada uno necesita unas características distintas que más adelante explicaremos.
- Cada Branch contiene carpetas, estas carpetas son nuestros servicios (Nginx, WordPress...)
- Dentro de las carpetas hay archivos con la extensión yaml donde su funcionamiento tiene un cierto parecido a los Dockerfile o a los Docker-compose.yml pero para entrar más en el tema hay que investigar en la página de [Kubernetes](https://kubernetes.io/) y ver los ejemplos que hemos puesto y que cogerán más sentido a lo largo de este informe.

Una vez comprendido el uso de Fleet lo que hace falta es configurarlo y aprender a interpretarlo. Más adelante en **Despliegue de servicios** explicamos cómo se despliegan aplicaciones en Fleet.

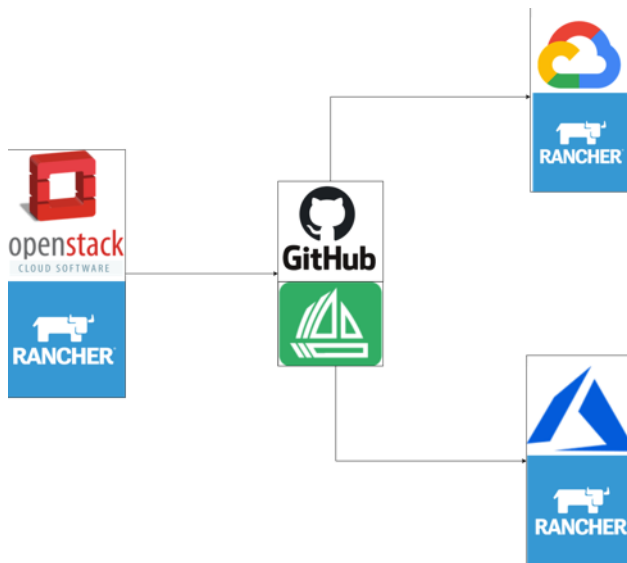
- Git en nuestro proyecto

Para desplegar los servicios anteriormente mencionados ya hemos comentado que utilizamos git, en especial el repositorio creado para este proyecto <https://github.com/Labraca/ARSO.git>. En este git almacenaremos todos los YAML que componen nuestros servicios, y algunos que están en desarrollo. Cuando se produzca en el git un commit o cambio del repositorio, la herramienta Fleet actualizará automáticamente los contenedores en Rancher, es por esta simpleza que hemos optado por Git para este fin.

Explicar el funcionamiento de los YAML no es una tarea fácil, es por ello por lo que es mejor ir a las fuentes oficiales de Kubernetes o páginas anexadas para entender todo el potencial que nos aportan crear estos archivos. Os dejamos la información aquí:

- [Trabajar con Objetos](#)
- [Deployment](#)

La implementación de Git y de Fleet en nuestra implementación se verá abstractamente de la siguiente manera:

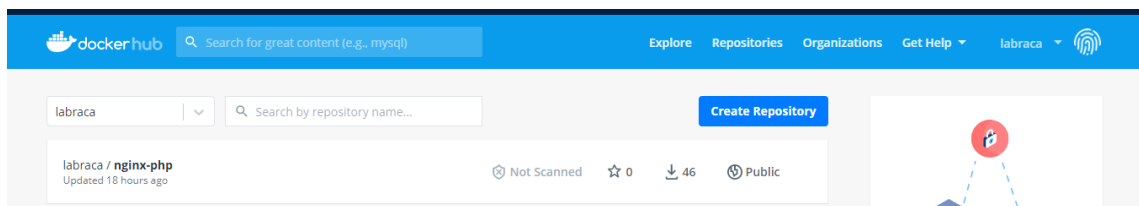


- Preparación de las imágenes

- *Nginx:*



Esta imagen la hemos hecho en una máquina virtual que contiene Docker. Usando una imagen, en concreto: <https://hub.docker.com/r/webdevops/php-nginx>



Hemos elegido esta imagen porque contiene php y Nginx ya incluidos para facilitar la implementación en un sistema Ubuntu, no obstante, se podría crear desde el principio bajo cualquier sistema operativo siempre y cuando pueda instalar los servicios de php y Nginx, aunque este es nuestro ejemplo, se podría haber usado otro tipo de servicios web como JavaScript,

```
ubuntu@docker-alm685:~$ docker pull webdevops/php-nginx_
```

Una vez elegida la imagen, la ejecutaremos con Docker run y accederemos al contenedor creado con su id copiando los archivos que veamos convenientes para nuestro servicio web.

```
ubuntu@docker-alm685:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
697191159f56   webdevops/php-nginx:latest          "/entrypoint supervi..." 2 days ago
62a8ad0df07b   labraca/nginxbueno:v0              "/docker-entrypoint..." 2 days ago
williamson
44b1dc19ea5c   nginx                               "/docker-entrypoint..." 7 days ago
ubuntu@docker-alm685:~$ docker exec -it 697191159f56 /bin/bash/
```

En nuestro caso, hemos preparado una página en php que incluye datos y además nos permite ver el nombre del Pod que está ejecutando esa imagen.

Una vez hemos puesto los datos en el espacio correspondiente, en función de la documentación de la imagen usada o si lo hemos hecho nosotros, la configuración personalizada que usemos, pasaremos a reiniciar los servicios para comprobar su funcionalidad.

Cuando comprobemos que nuestro contenedor funciona correctamente y nos ofrece el servicio web, haremos un commit junto a un tag sobre nuestro contenedor modificado, creando así una imagen modificada del mismo.

```
ubuntu@docker-alm685:~$ docker commit exciting_sanderson labraca/nginx-php:v0
```

Una vez hecho esto, subiremos la imagen a nuestro repositorio personal y habríamos finalizado.

```
login succeeded
ubuntu@docker-alm685:~$ docker push labraca/nginx-php:v0
The push refers to repository [docker.io/labraca/nginx-php]
272469c238d0: Pushed
2335437c7bee: Layer already exists
c9021a0df109: Layer already exists
b8167a4f532b: Layer already exists
780f768e11e9: Layer already exists
06eda94af7f8: Layer already exists
5a8729e90d94: Layer already exists
3daf0cc5c641: Layer already exists
b32f54152b11: Layer already exists
6be7fa578c62: Layer already exists
9bab65932166: Layer already exists
015de4fe565b: Layer already exists
16542a8fc3be: Layer already exists
6597da2e2e52: Layer already exists
977183d4e999: Layer already exists
c8be1b8f4d60: Layer already exists
v0: digest: sha256:dc236ce8b59cf5a08ef9d860c326f21603ccdb10b9536e1517d6acd16d1fe4f8 size: 3677
```

Con este método también se crean las distintas versiones de las imágenes, al hacer una nueva revisión de la imagen se debe cambiar el tag para poder actualizar la versión en los archivos con extensión yaml y que Rancher instale la nueva versión.

- *WordPress*

Para el caso de las imágenes de WordPress, no vamos a modificar las imágenes puesto que este servicio se modifica una vez desplegado. Las imágenes que usaremos serán las de WordPress y mysql:5.6.

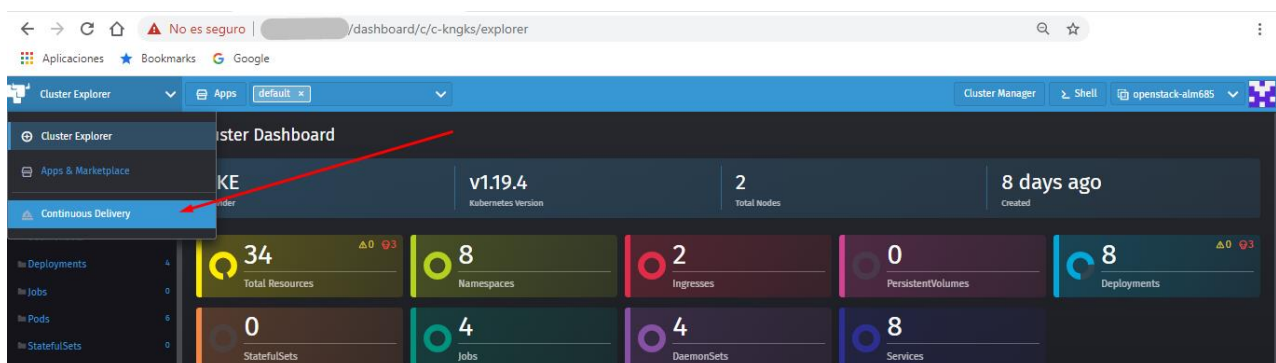
- FileZilla

Nuestra idea para usar una imagen de FileZilla es coger la del siguiente repositorio: <https://hub.docker.com/r/jlesage/filezilla/> Es importante ver las instrucciones de cada contenedor para entender cómo usar la imagen, esto implica, ver la documentación para averiguar puertos que son necesarios entre otras cosas.

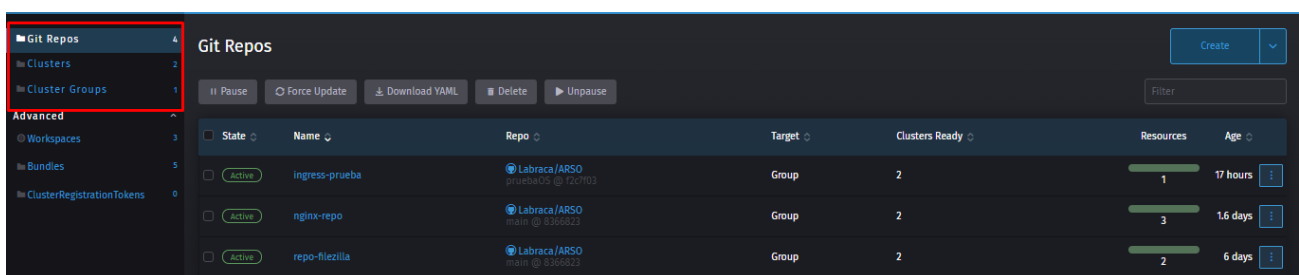
• Despliegue de servicios

Una vez creado e inicializado el clúster ya podemos desplegar apps en él. En Rancher se puede hacer manualmente accediendo a cada clúster y generando un Workload, pero en este proyecto hemos decidido usar la herramienta Fleet que nos permite desplegar en varios proyectos varias apps y servicios desde un git.

○ Fleet



Para acceder a Fleet debemos entrar en la sección *Continuous Delivery* del *dashboard* de Rancher (*Cluster Explorer*) donde podremos establecer un grupo de clústeres, un repositorio y ver los que ya se encuentran en el espacio de trabajo.



Aquí podemos añadir un repositorio de donde sacar nuestros archivos *yaml*. Para cada repositorio de la imagen realmente son el mismo, pero señalando a una carpeta diferente del *git*.

Al crear el repositorio podemos señalar en qué clústeres se desplegará la aplicación:

The screenshot shows the 'Git Repo: Create' form. The 'Name' field is 'example-repo'. The 'Repository URL' is 'https://github.com/example/example.git'. The 'Watch' dropdown is set to 'A Branch' and the 'Branch Name' is 'master'. The 'Authentication' dropdown is set to 'None'. The 'Paths' section has a text input field and an 'Add Path' button. The 'Deploy To' section has a 'Target Type' dropdown set to 'All Clusters in the Workspace', a 'Service Account Name' field, and a 'Target Namespace' field. The 'Labels' section has an 'Add Label' button. A red arrow points to the 'All Clusters in the Workspace' option in the 'Target Type' dropdown.

Nosotros tenemos un grupo de 3 clústeres, uno por cada proyecto de los integrantes del grupo, en el que vertemos todas las aplicaciones.

Con esto ya se tendrían las aplicaciones operativas a no ser que no estuvieran bien configuradas. Para configurarlas hemos seguido estas pautas:

- *Servicio web*

Para la creación de un servicio web, en nuestro caso, hemos usado la imagen previamente subida a nuestro repositorio personal de Docker Hub, así que esa será la que generaremos en nuestro despliegue en el clúster.

Para crear nuestro despliegue de la imagen, crearemos un archivo *yaml* que definirá como se creará el servicio y qué imagen usaremos.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: labraca/nginx-php:v0
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: regcred
```

Vamos a observar el archivo y comentar las cosas más relevantes del despliegue.

Por una parte, el campo kind, qué tendremos que definirlo como Deployment puesto que vamos a realizar un despliegue.

En la etiqueta metadata definiremos el nombre, el cual nos servirá para identificar el despliegue, al igual que su nombre con el campo app puesto que necesitaremos acceder a él para poder definir las propiedades del despliegue.

En el campo replicas, seleccionaremos cuantos Pods vamos a crear con este despliegue.

Otro campo importante es el selector, el cual hará la selección de la aplicación en su conjunto, aquí crearemos un par de campos más, entre ellos otro spec, campo que contiene los nodos de configuración del objeto.

Aquí incluiremos containers, este es el campo fundamental que definirá los distintos contenedores creados en el pod.

Definiremos una vez más una etiqueta de nombre, la imagen, en la que haremos referencia a la imagen guardada en nuestro repositorio personal y finalmente los puertos usados. Como es un servicio web, internamente usará el puerto 80.

Como estamos usando un repositorio personal, es necesario definir el campo `imagePullSecrets`, donde escribiremos el campo `name` y “`regcred`”.

Esto es importante ya que como accedemos a un repositorio personal, necesitamos las credenciales del repositorio privado, así que adicionalmente crearemos otro archivo en la misma carpeta donde generaremos el secreto.

```
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJodHRwczovL2luZGV4
kind: Secret
metadata:
  name: regcred
type: kubernetes.io/dockerconfigjson
```

Este archivo incluye las credenciales de nuestro repositorio privado, las cuales se generan cuando hacemos uso de `Docker login`, no obstante, están encriptadas para dar cierta seguridad, no obstante, en nuestro caso están en GitHub para aprovechar la integración continua, por tanto, tenemos que asegurarnos de mantener el repositorio en privado para evitar el robo de credenciales.

- *FileZilla*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: filezilla
  name: filezilla-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: filezilla
  template:
    metadata:
      labels:
        app: filezilla
    spec:
      containers:
        -
          image: jlesage/filezilla
          name: filezilla
          ports:
            -
              containerPort: 5800
```

Para desplegar nuestro servicio de FileZilla, comenzaremos creando un despliegue en un archivo yaml. Nuestro archivo, debe de ser de tipo Deployment.

En el apartado *metadata*, pondremos el nombre qué le vamos a dar a nuestro despliegue, a su vez, nos permitirá declarar una *label* (etiqueta). Las etiquetas al igual que el nombre nos sirve para poder hacer referencia a nuestro despliegue.

En spec podemos declarar entre otros, el número de replicas qué haremos de cada despliegue, es decir, podemos elegir con cuantos contenedores

comenzaremos en cada despliegue.

Otro punto importante es la selección de la imagen de los contenedores que vamos a desplegar.

Además de poner la imagen, pondremos los puertos qué usará nuestro contenedor internamente.

- *WordPress*

A diferencia del anterior servicio, en este caso, para desplegar WordPress, no usaremos una imagen personalizada, sin embargo, al ser este un servicio más complejo, crearemos un par de archivos yaml.


```
apiVersion: apps/v1 # for versions above 1.9.0
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
```

Empezaremos con el archivo que desplegará una imagen de MySQL, en nuestro caso, comenzaremos poniendo campos para definir etiquetas que nos sirven para seleccionar la configuración.

```
spec:
  containers:
    - image: mysql:5.6
      name: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
      ports:
        - containerPort: 3306
          name: mysql
      volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
  volumes:
    - name: mysql-persistent-storage
      persistentVolumeClaim:
        claimName: mysql-pv-claim
```

A continuación, especificaremos distintas características como la imagen que usaremos, en este caso MySQL para montar un servicio donde podamos guardar datos del WordPress, pero de manera permanente. Para esto comenzaremos definiendo algunas variables de entorno como

MYSQL_ROOT_PASSWORD, esta variable de entorno tendrá el valor de un “secret” o contraseña guardada en Rancher que previamente hemos dejado en el repositorio.

```
apiVersion: v1
data:
  password: d29yZHB5ZXNzUGFzcw==
kind: Secret
metadata:
  name: mysql-pass
  namespace: default
  selfLink: /api/v1/namespaces/default/secrets/mysql-pass
  uid: c62c1885-179e-43c3-9743-a247125b4cd5
type: Opaque
```

De esta manera, lo que estamos haciendo es poner el valor de una contraseña que llamaremos mysql-pass, esta será la contraseña de la base de datos que crearemos que pasaremos al contenedor con el campo secretKeyRef.

Por otra parte, declararemos los puertos, MySQL usa 3306, así que cambiará frente al puerto 80 usado previamente. Con el campo volumeMounts, seleccionaremos el volumen que posteriormente declararemos y elegiremos la ruta interna donde se creará el espacio de almacenamiento de la base de datos.

Finalmente, con volumes, crearemos un mysql-persistent-storage, el cual es un espacio persistente de donde crearemos los volúmenes como especificaremos en la configuración.

Antes de crear los objetos persistentes, continuaremos con el archivo de despliegue de WordPress.

```
apiVersion: apps/v1 # f
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
```

El archivo de despliegue de WordPress comenzará con una estructura similar a las vistas anteriores, usando el campo Deployment y usando etiquetas para más tarde poder seleccionarse.

```
spec:
  containers:
    - image: wordpress:4.8-apache
      name: wordpress
      env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
      ports:
        - containerPort: 80
          name: wordpress
      volumeMounts:
        - name: wordpress-persistent-storage
          mountPath: /var/www/html
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
            claimName: wp-pv-claim
```

Después, pasaremos a editar los distintos campos del contenedor usado, en concreto usaremos la imagen de wordpress:4.8-apache, para este contenedor, además usaremos la variable de entorno de la contraseña que hemos creado previamente en otro archivo yaml. Especificaremos el puerto 80 y la ruta donde montaremos el contenido de HTML.

Por otra parte, volveremos a usar el espacio de almacenamiento persistente desde donde crearemos el volumen.

Tendremos qué montar dos, uno para cada despliegue.

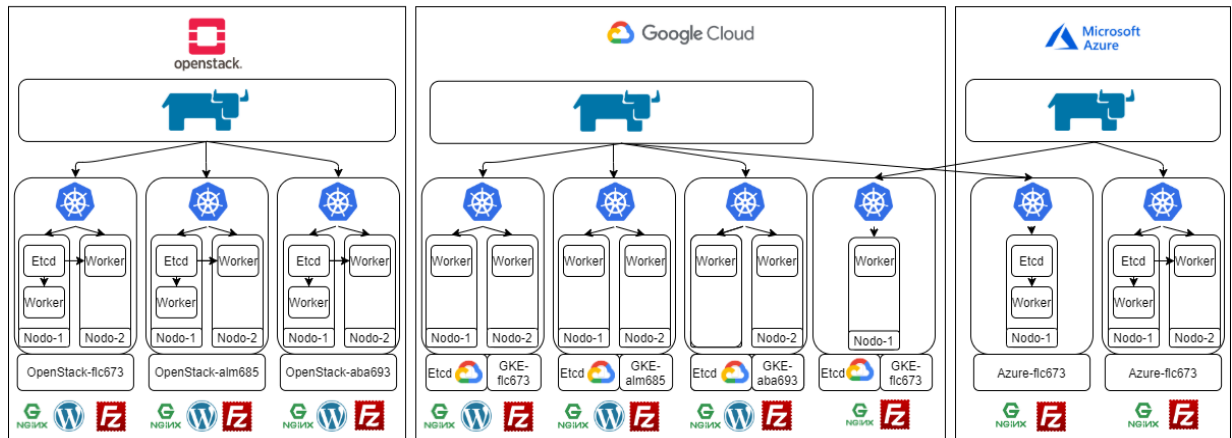
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

Aquí por ejemplo tendríamos un ejemplo del despliegue en GKE, donde vamos a crear un espacio de 20Gi. Una vez creado, este se conectará a nuestro proveedor de almacenamiento declarado en StorageClass el cual puede depender del proveedor. Como solo tenemos uno, no especificaremos el nombre de la clase y se cogerá por defecto el de GKE.

Para otra plataforma como puede ser OpenStack, usaremos el plugin de Cinder y, por tanto, tendremos qué especificar otro storageClassName ya qué la integración con OpenStack es distinta a la que se pueda dar en Azure, Google GKE u otros proveedores.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  volumeName: wordpress-pv
  resources:
    requests:
      storage: 10Gi
  storageClassName: cep-cinder-sc
```

Por tanto, la topología de nuestro proyecto tras la creación de las máquinas Rancher y los clústeres de Kubernetes, y desplegar todos los servicios en Fleet es la siguiente es la siguiente:



La idea principal era implementar toda la topología desde un único Rancher, pero en la Red de la universidad, las máquinas de otras nubes no pueden acceder, por lo tanto, hay problemas de conectividad en Kubernetes, es por ello que al final optamos por un Rancher en cada nube y al final interconectar la nube de Azure con la de Google Cloud para demostrar que era posible sin problemas.

• Volúmenes

StorageClass

Parte importante de nuestro despliegue o para cualquiera, es el espacio de almacenamiento, el cual puede ser que necesitemos ampliar en caso de cualquier imprevisto o simplemente por la necesidad de una ampliación de la infraestructura.

Históricamente, lo que se ha hecho ha sido sustituir o añadir un nuevo disco duro al servidor que estuviera ofreciendo determinado servicio, pero esto con la nube ha cambiado. Ahora simplemente se pide al proveedor un determinado espacio de almacenamiento en determinado volumen que se quiere para guardar los datos que deseemos y que podemos especificar en el lugar donde queramos montar ese volumen.

Esto lo hacemos a través de los volúmenes persistentes como hemos visto en el despliegue de WordPress, no obstante, necesitamos un proveedor de ese espacio. Ese proveedor del espacio de almacenamiento no va a ser sino nuestro proveedor de la nube, para el cual Rancher se ocupará de usar el plugin del servicio correspondiente de nuestro proveedor de la nube que usaremos.

Rancher cuenta con diversos *plugin* en función del proveedor de almacenamiento:

NAME	PLUGIN
Amazon EBS Disk	aws-ebs
AzureFile	azure-file
AzureDisk	azure-disk
Google Persistent Disk	gce-pd
Longhorn	flex-volume-longhorn
VMware vSphere Volume	vsphere-volume
Local	local
Network File System	nfs
hostPath	host-path

Con nuestros proveedores dinámicos funcionando a través del plugin, podemos ver como crearán los diversos volúmenes.

Cada proveedor habitualmente tiene su propio proveedor de almacenamiento, veamos algunos ejemplos:

- OpenStack

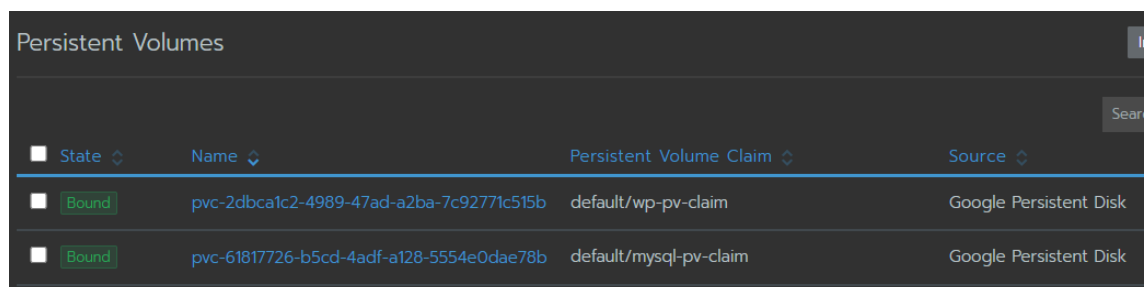
OpenStack tiene su propio sistema de almacenamiento, si queremos usarlo, en versiones anteriores, usamos el plugin de Cinder, no obstante, este plugin ha dejado de ser soportado por las últimas versiones. Esto ha implicado, la imposibilidad de crear volúmenes dinámicamente en Rancher fuera del modo manual.

- Google GKE

GKE tiene su proveedor también, este funcionará con la siguiente definición:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  fstype: ext4
  replication-type: none
```

Gracias a la definición de la clase con la definición de los volúmenes persistentes previamente, se crearán aquí:



State	Name	Persistent Volume Claim	Source
Bound	pvc-2dbca1c2-4989-47ad-a2ba-7c92771c515b	default/wp-pv-claim	Google Persistent Disk
Bound	pvc-61817726-b5cd-4adf-a128-5554e0dae78b	default/mysql-pv-claim	Google Persistent Disk

También podemos observarlos en la consola de Google Cloud.

✓ gke-c-gqzrj-27b083ae-d-pvc-2dbca1c2-4989-47ad-a2ba-7c92771c515b

DETALLES

MONITORIZACIÓN

Propiedades

Descripción	{"kubernetes.io/created-for/pv/name":"pvc-2dbca1c2-4989-47ad-a2ba-7c92771c515b","kubernetes.io/created-for/pvc/name":"wp-pv-claim","kubernetes.io/created-for/pvc/namespace":"default"}	
Tipo	Disco persistente estándar	
Tamaño ?	20 GB	
Zona	europe-west2-c	
Etiquetas	display-name : gke-aba693 goog-gke-volume	
Usado por	gke-c-gqzrj-default-0-1c83839e-0j11	
Programación de capturas	Ninguna	

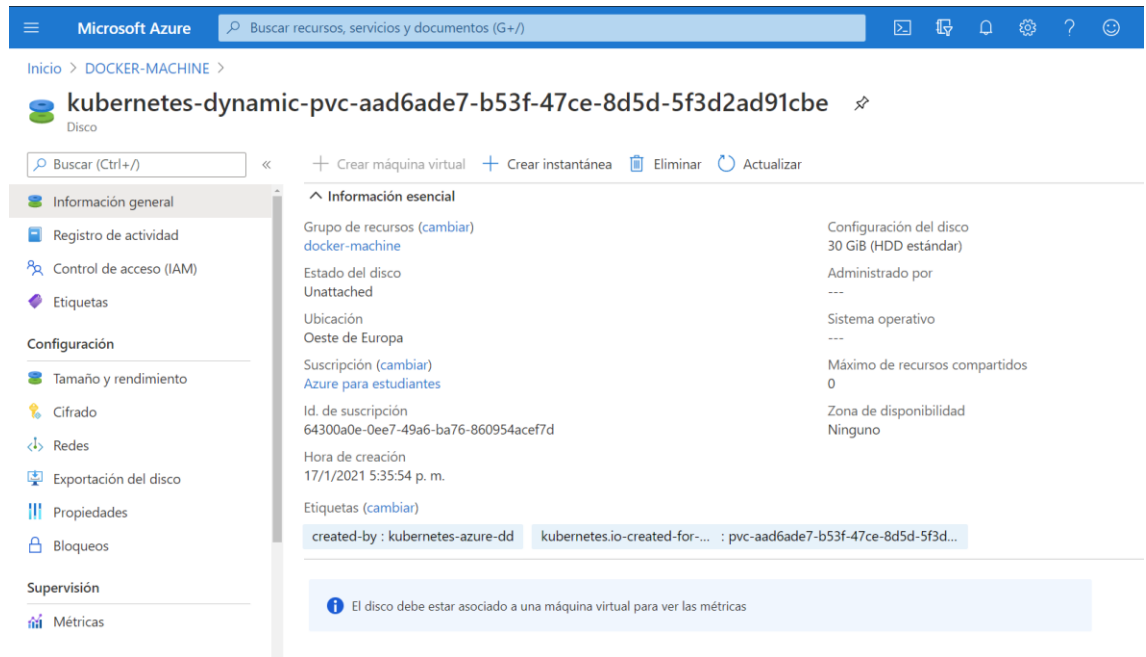
- Azure

Azure usa varios, esto es porque nos permite cambiar en función al que más se adapte a nuestras necesidades.

Veamos un ejemplo de la clase en nuestro Rancher:

Storage Classes		
Delete		
<input type="checkbox"/> State	Name	Provisioner
<input checked="" type="checkbox"/> Active	slow	Azure Disk

Y por otra parte, el espacio persistente creado por la clase en la consola de azure.



- **LoadBalancers**

Una vez tenemos nuestra aplicación desplegada, surge el problema de poder exponerla al uso. Nuestro despliegue de Nginx está configurado para recibir peticiones por el puerto 80 del contenedor, pero esto no es posible puesto que en cada clúster se crearán varios contenedores de Nginx y estos utilizan la misma IP que su nodo contenedor. Por lo tanto, usamos herramientas para exponer la aplicación al público, los LoadBalancers que se dividen en capa 4 y 7. También se puede exponer una aplicación utilizando otros métodos como NodePort que es un servicio que redirecciona desde un puerto de la IP publica a un puerto interno, pero a diferencia de los balanceadores de carga estos no tienen un método de repartir las solicitudes, ese es el punto fuerte.

- **Layer7**

Los balanceadores de carga de capa 7 solo redireccionan en http y https, en Kubernetes se utilizan los servicios Ingress, que exponen las aplicaciones por un puerto de la IP publica del Nodo. Estos servicios pueden además unir varios servicios en una misma IP y puerto puesto que podemos redireccionar mediante URI.

El problema principal es que no transportan archivos estáticos refiriéndonos a archivos CSS y JS, lo cual para nuestra aplicación de Nginx son fundamentales por lo que descartamos esta opción.

▪ *Layer4*

A diferencia de los balanceadores de capa7, estos pueden redireccionar TCP y consigo los archivos anteriormente mencionados. La principal ventaja de estos balanceadores es que reparten la carga de solicitudes entre un pool de direcciones, lo que ayuda al sistema a mantener la disponibilidad. Esto puede definirse manualmente o de la siguiente manera, que es la que hemos seguido:

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-lb
5  spec:
6    type: LoadBalancer
7    ports:
8      - port: 80
9    selector:
10     app: nginx
```

En la anterior imagen se muestra nuestro balanceador de carga para la app de Nginx. Utilizando este archivo yaml desde fleet, Rancher despliega un servicio LoadBalancer en el Cloud Provider (en este caso OpenStack), que se asocia con los contenedores que estén ejecutando la aplicación Nginx, indicando que se llega a estos por el puerto 80. Los balanceadores de carga tienen su propia IP publica suministrada por el cloud provider.

Una vez tenemos los balanceadores de carga se necesitaría una forma de unirlos, puesto que se necesita un balanceador de carga por cada servicio, lo que puede hacerse costoso, además los balanceadores normalmente vienen predefinidos para utilizar el algoritmo Round Robin lo cual carga mucho el clúster. Para unirlos todos utilizar Ingress no es suficiente, aunque sea su ventaja más notable el unirlos, pues nosotros en nuestra aplicación seguimos necesitando nuestros archivos CSS, para poder hacer esto hemos recurrido a un dominio con el que direccionar hacia nuestros balanceadores de carga.

En OpenStack esta parte no funciona puesto que nuestros proyectos se encuentran en una red privada.

- DNS

Vamos a usar un DNS que usaremos junto a un dominio, el cual nos ayudará a la accesibilidad de los distintos usuarios a los servicios en cada clúster, para explicar que es, usaremos la entrada que viene en Wikipedia.

- El **sistema de nombres de dominio** (*Domain Name System* o **DNS**, por sus siglas en inglés) es un sistema de nomenclatura jerárquico descentralizado para dispositivos conectados a redes IP como Internet o una red privada. Este sistema asocia información variada con nombres de dominio asignados a cada uno de los participantes. Su función más importante es "traducir" nombres inteligibles para las personas en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente.

Entendiendo que un usuario no se debe de preocupar de estas cosas, para facilitarle la vida, compraremos un dominio, este será el nombre que emplearemos en lugar de las distintas direcciones IP. Nosotros partiremos de nuestro dominio registrado como warso.studio, el cual hemos registrado en <https://www.name.com/>.

En nuestro espacio del dominio comprado, será donde añadiremos nuestros registros tipo A, este tipo de registro funciona para la traducción propiamente dicha, de una IP a un nombre.

<input type="checkbox"/>	TIPO	SERVIDOR	RESPUESTA	TTL	PRIO	ACCIONES
<input type="checkbox"/>	A	web.warso.studio	35.246.78.187	300	N/A	<input type="button" value="EDITAR"/> <input type="button" value="BORRAR"/>
FECHA DE CREACIÓN: 2021-01-15						
<input type="checkbox"/>	A	wordpress.warso.studio	35.246.13.70	300	N/A	<input type="button" value="EDITAR"/> <input type="button" value="BORRAR"/>
FECHA DE CREACIÓN: 2021-01-15						

Por tanto, debemos ir asociando los distintos nombres que queramos a la IP, en nuestro caso, a las IP asociadas a nuestro balanceador de carga. Por supuesto, esto cambiará en la forma de añadirlo en función del proveedor de dominio.

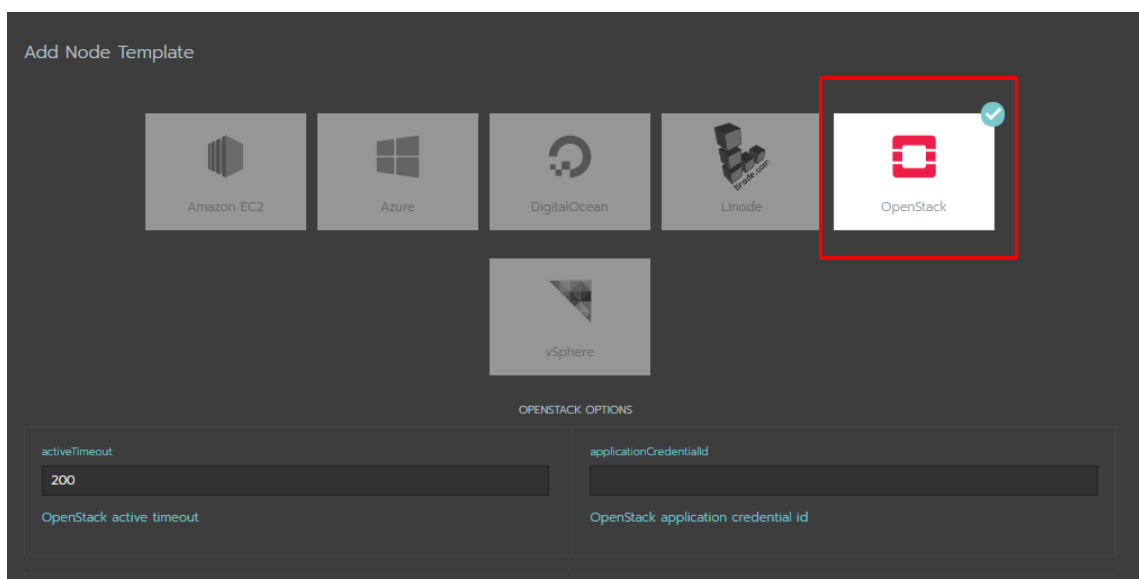
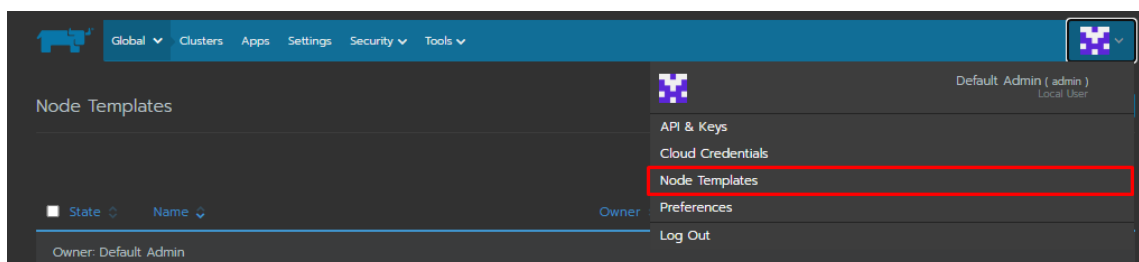
3. Creación de MV y despliegue de Clústeres.

- OpenStack

- Preparación del entorno

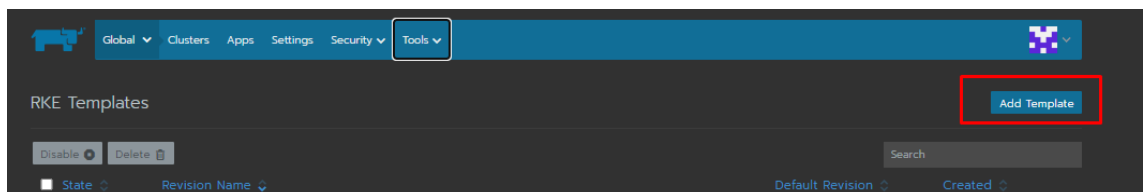
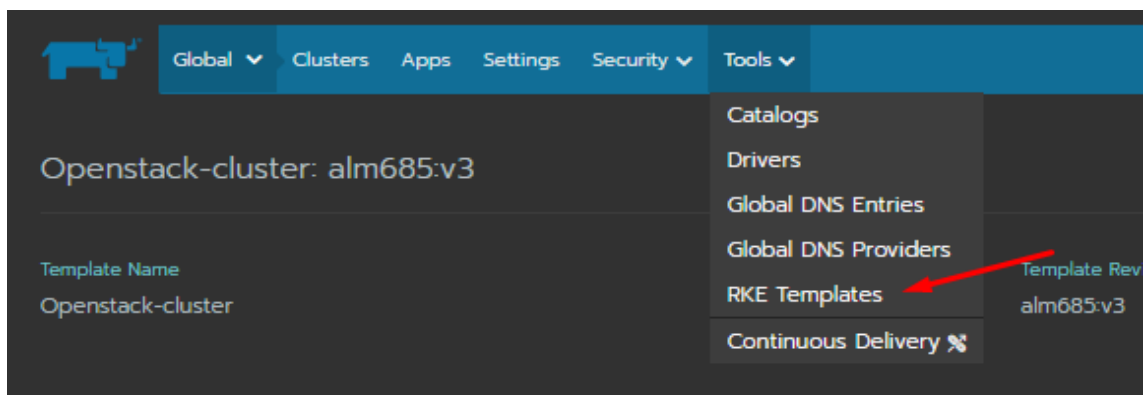
El objetivo principal es crear un clúster Kubernetes de los nodos que deseemos y utilizarlo para desplegar varias aplicaciones.

Lo primero que hay que hacer es crear una plantilla para la configuración de los nodos en la nube que van a crearse.



Aquí se debe introducir los datos del proyecto de OpenStack donde se van a crear los nodos, casi todo se puede sacar de las credenciales de la API de la cloud (OpenStack RC File) y en el apartado redes del proyecto.

Cuando tengamos una plantilla para nuestros nodos podríamos crear nuestro clúster, pero en OpenStack debemos especificar a nuestro clúster que su proveedor es este, por lo que necesitamos crear una plantilla para el clúster, esto lo hacemos en las RKE Templates donde la especificamos al motor de Rancher Kubernetes como queremos crear el clúster.



En la siguiente pestaña al darle a “Add Template” lo importante es editar el YAML pues OpenStack se configura manualmente en el documento. Este método está siendo poco a poco eliminado del programa puesto que los desarrolladores están intentando volver más modular el programa y poder configurar esto de una manera más externa. El problema es que el soporte con OpenStack está decayendo por lo que la forma de hacerlo no es muy clara, hemos optado entonces por usar esta forma pues es más directa.

En el YAML hay que modificar el apartado cloud-provider de esta manera:

```

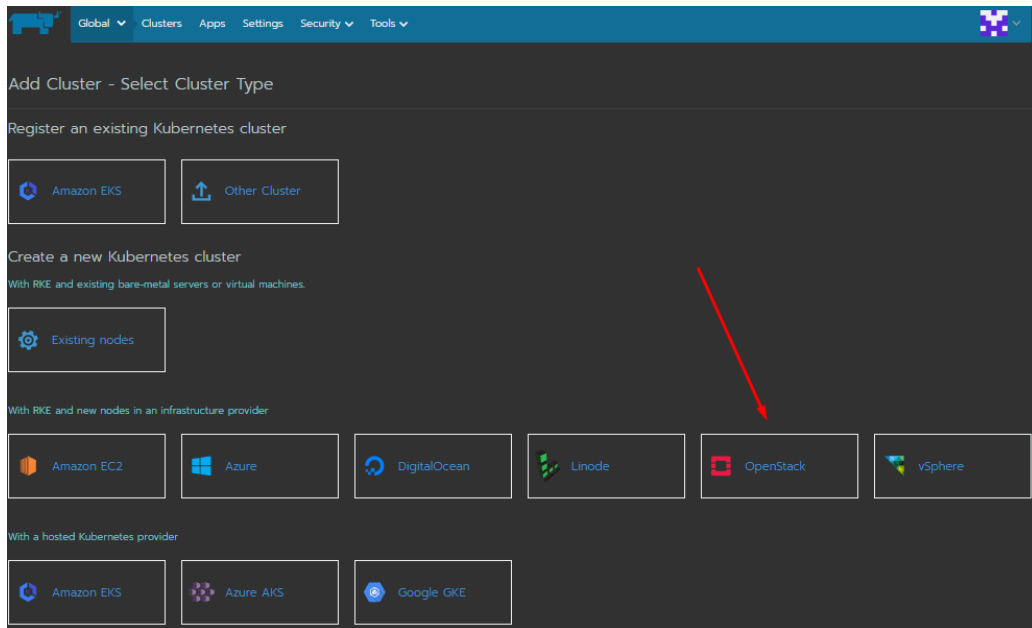
cloud_provider:
  name: openstack
  openstack_cloud_provider:
    block_storage:
      bs-version: v2
      ignore-volume-az: true
      trust-device-path: false
    global:
      auth-url: 'http://192.168.64.12:5000/v3/'
      domain-name: Default
      tenant-id: [REDACTED]
      username: [REDACTED]
    load_balancer:
      create-monitor: false
      floating-network-id: [REDACTED]
      manage-security-groups: false
      monitor-max-retries: 0
      subnet-id: [REDACTED]
      use-octavia: false
    metadata:
      request-timeout: 0

```

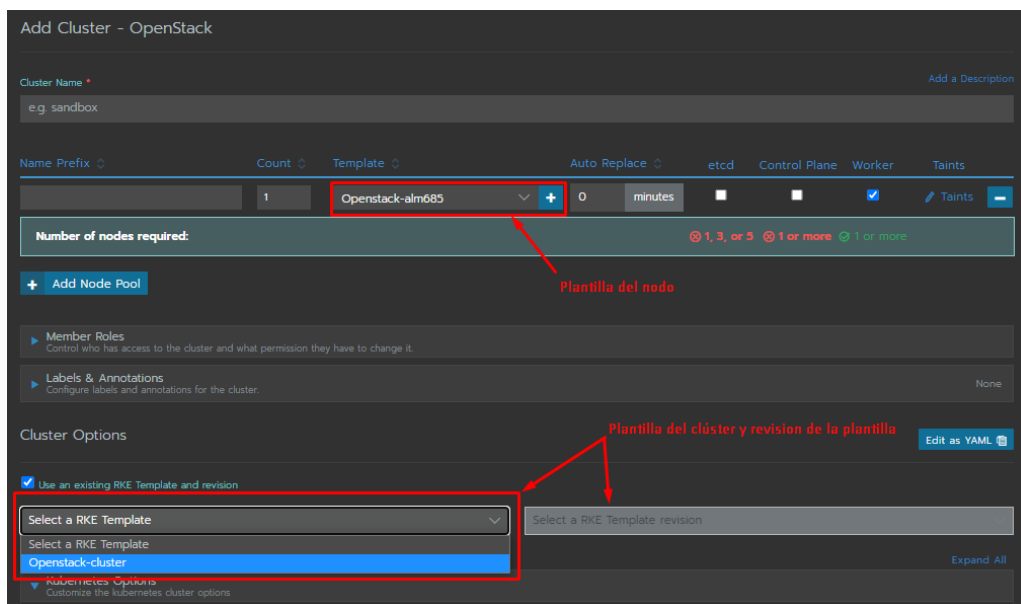
Parecido a la plantilla de los nodos, pero aquí aparte de la configuración de dónde va a trabajar el clúster se configura lo balanceadores de carga y los volúmenes que se vayan a crear que nos harán falta más adelante. (Además hay que introducir un nuevo campo password con la contraseña del proyecto)

Para crear un clúster en Rancher solo hay que seleccionar la opción en la pestaña Global, seleccionar el proveedor de infraestructura (En este caso OpenStack), añadir los nodos y configurar el clúster.

State	Cluster Name	Provider	Nodes	CPU	RAM	Actions
Active	local	Imported v18.8-k3s1	1	0.2/4 Cores 5%	0.1/7.8 GB 2%	Explorer
Active	openstack-aba693	OpenStack v19.4	5	15/20 Cores 7%	0.2/38.5 GB 0%	Explorer
Active	openstack-alm685	OpenStack v19.4	2	0.7/8 Cores 9%	0.2/5.4 GB 1%	Explorer



Introducimos los valores del Nodo Kubernetes, y seleccionamos los Node Template y RKE Template de OpenStack.



Un clúster ya creado quedaría de esta forma en su configuración:

Cluster Name ^{*} [Add a Description](#)

Name Prefix	Count	Template	Auto Replace	etcd	Control Plane	Worker	Taints
rchrman-alm	1	Openstack-alm685	0 minutes	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Taints
rchr-alm	1	Openstack-alm685	0 minutes	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Taints

Number of nodes required: 👤 1, 3, or 5 👤 1 or more 👤 1 or more

Changed node template will only affect newly created nodes.

[+ Add Node Pool](#)

Member Roles
Control who has access to the cluster and what permission they have to change it.

Labels & Annotations
Configure labels and annotations for the cluster. 10 Configured

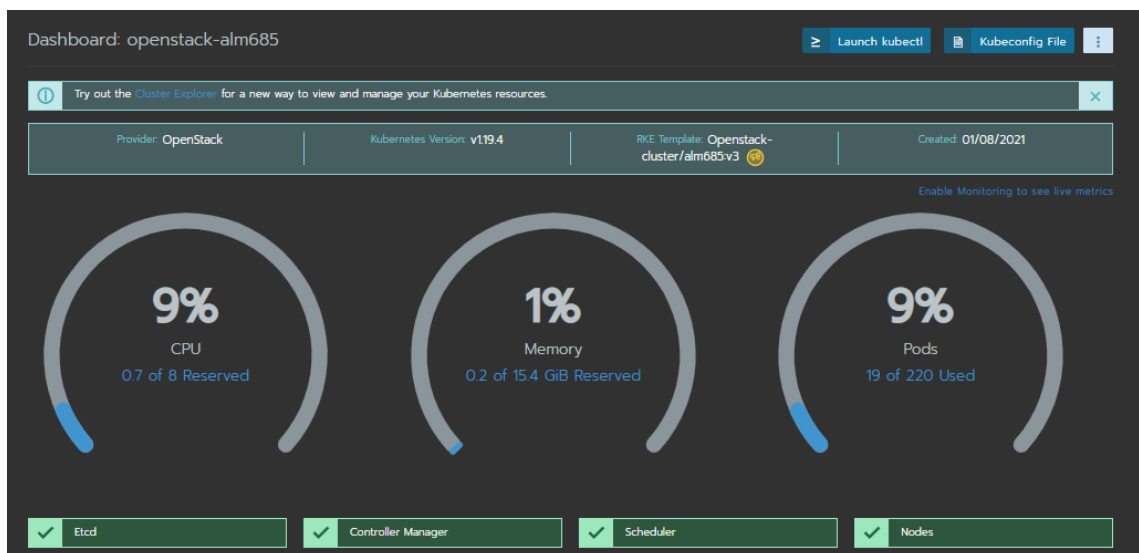
Cluster Options

☒ Use an existing RKE Template and revision

alm685v3 (Created 3 days ago)

[Collapse All](#)

Cuando esté listo, solo falta esperar a que Rancher termine de inicializar los nodos del clúster.



○ Troubleshooting

El despliegue en OpenStack actualmente está hecho en la red privada de la universidad, por lo que cualquier servicio viene capado hablando de puertos, hemos aprovechado que

el puerto 80 está abierto para poder sacar los servicios, pero sólo son usables a través de VPN.

El servicio de OpenStack como cloud provider en Rancher está cada vez más obsoleto, en esta configuración de Rancher v2.5.3 hemos podido usar el In-tree cloud provider que es la especificación de este en el archivo YAML de configuración del clúster RKE. Se ha anunciado que esta función será eliminada eventualmente para lo cual OpenStack lleva un proyecto de Cloud-controller para Kubernetes el cual hemos evitado involucrarnos.

Al desplegar las aplicaciones en Fleet pueden verse desactualizadas, para amenizar el proceso borrábamos los Bundles que se generaban de nuevo con los nuevos cambios hechos.

○ Conclusión

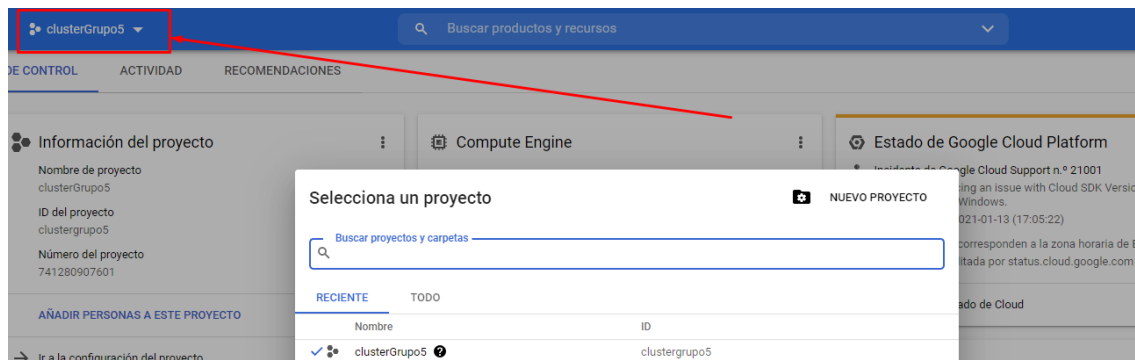
El despliegue en OpenStack tiene más obstáculos que en GKE y Azure, puesto que la empresa está yendo en una dirección más modular y comercial, además de la complicación de ser una red privada. Pero este despliegue es perfecto para realizar las pruebas de funcionamiento dado que es una red privada con más seguridad y se buscan maneras más sencillas de sacar los servicios.

● Despliegue en Google Cloud

○ Preparación del entorno

Para la creación de un entorno en Google Cloud lo primero que necesitaremos es tener una cuenta de Google Cloud en <https://cloud.google.com/>

Tras entrar en la plataforma (<https://console.cloud.google.com>) lo que debemos hacer es crear un nuevo proyecto.



Nuevo proyecto

⚠ Te quedan 16 projects en la cuota. Solicita un aumento o elimina proyectos. [Más información](#)
[MANAGE QUOTAS](#)

Nombre de proyecto *
proyectoarso

ID del proyecto *
proyectoarso-1

El ID del proyecto puede estar formado por letras minúsculas, dígitos o guiones, y debe empezar por una letra minúscula y terminar con una letra o un número.

Ubicación *
Ninguna organización [EXPLORAR](#)

Carpeta u organización principal

[CREAR](#) [CANCELAR](#)

Asignamos un nombre al proyecto, un ID de proyecto si no nos gusta el que se asigna automáticamente y una ubicación. Ya tendremos nuestro proyecto creado.

Para que nos permita trabajar deberemos asignarle una facturación y ya estaría listo para los siguientes pasos.

Una vez creada tenemos que crear un proyecto y habilitar Compute Engine y la API de Kubernetes Engine. Para ello tenemos que buscarlo en la barra de “Buscar productos y recursos” nuestras APIs que queremos habilitar. Como nosotros ya lo tenemos habilitado no nos sale la opción, pero si es la primera vez que lo habilitamos, nos dirá **Habilitar API**.



Kubernetes Engine API

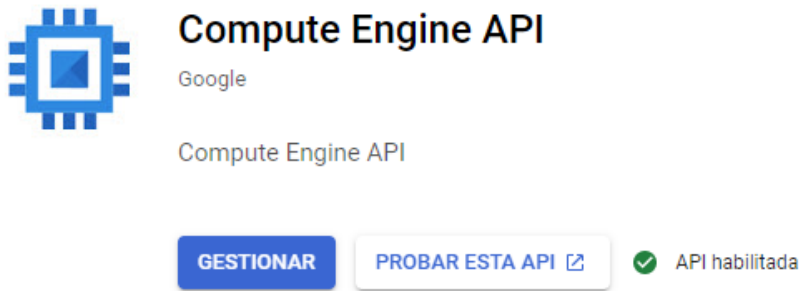
Google

Builds and manages container-based applications, powered by the open source Kubernetes technology.

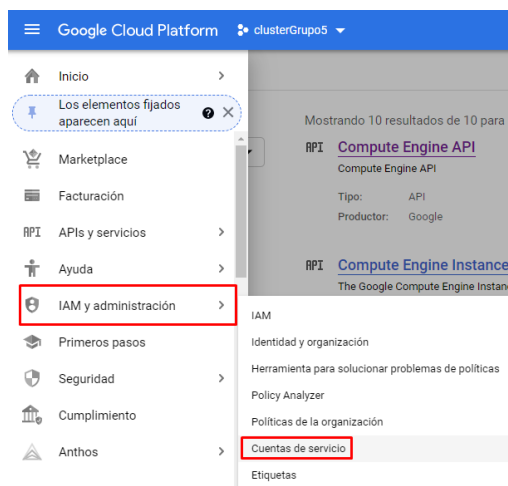
[GESTIONAR](#)

[PROBAR ESTA API](#)

✓ API habilitada



Cuando hemos habilitado estas dos API lo que tenemos que hacer es dar a nuestro proyecto una serie de permisos que permitirán a nuestro Rancher crear los clústeres en Google Cloud automáticamente.



Para empezar, tenemos que seleccionar en el menú lateral de nuestra consola **IAM y administración** e irnos a **Cuentas de Servicio**.

Una vez allí seleccionamos **Crear nuevo servicio**.

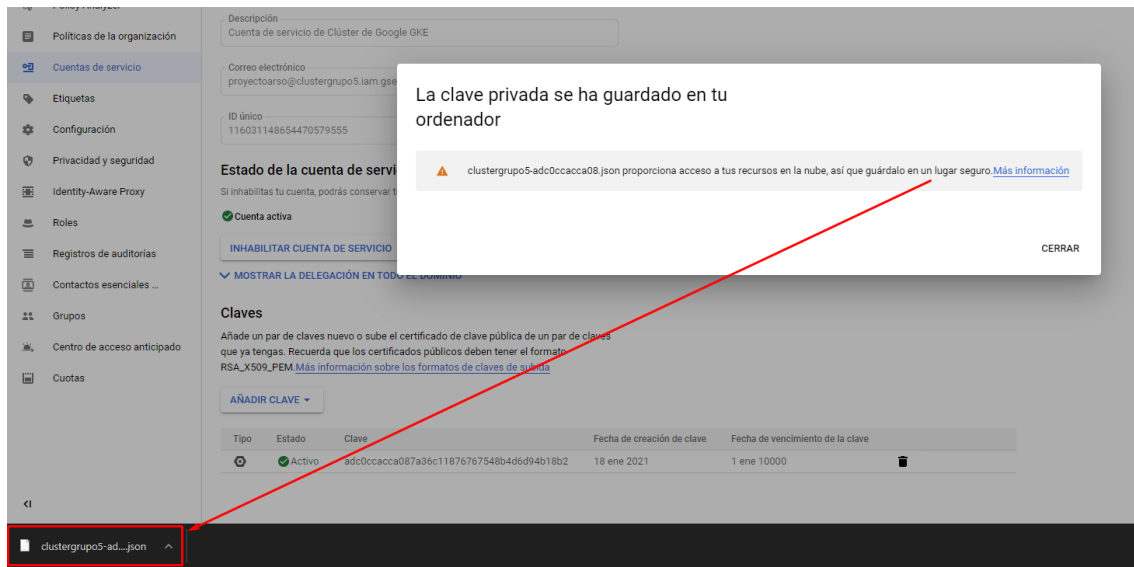
Al crear la cuenta de servicio tenemos que darle un nombre y opcionalmente una descripción, al darle continuar nos pedirá que le introduzcamos unos roles de acceso al proyecto, los necesarios para que Rancher pueda gestionar y trabajar con nuestro proyecto son los que se pueden observar en la imagen:

Para finalizar, si queremos, podemos designar cuentas que también estarán vinculadas a nuestro rol de miembro, como usuarios o como administrador.

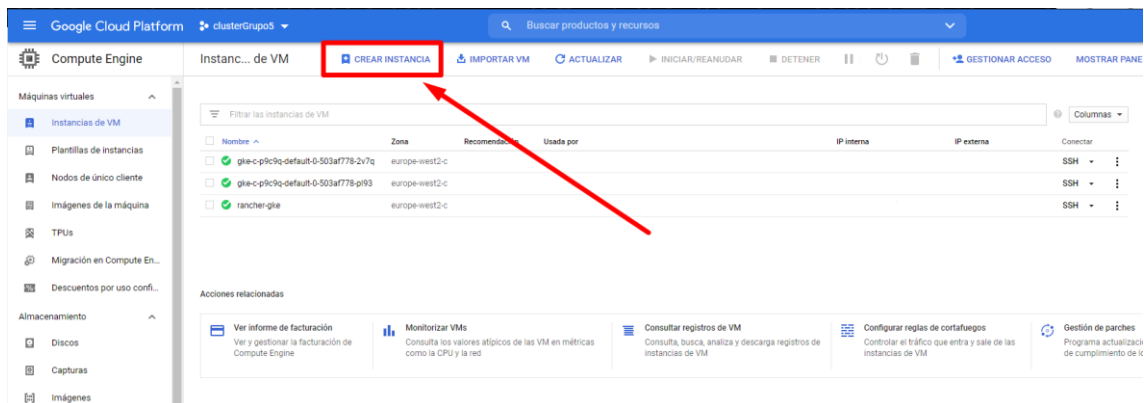
<input type="checkbox"/>		<input checked="" type="checkbox"/>	proyectoarso	Cuenta de servicio de Clúster de Google GKE	No hay claves
	proyectoarso@clustergrupo5.iam.gserviceaccount.com				

Una vez creado el rol, deberemos clicar en él y entraremos a crear una clave JSON que admita Rancher.

Para crear la clave tendremos que ir al apartado **Añadir Clave** y seguidamente **Crear Clave**, eligiendo **JSON**, hay que guardar esta clave en un sitio seguro porque con ella tendremos acceso a la creación y gestión de recursos en nuestro proyecto de Google.



Para crear una máquina virtual en Google Cloud lo primero que debemos hacer es activar la API de Compute Engine e ir a la parte de Instancias.



Elegimos los datos que consideremos mejor para el tamaño de la instancia, y su posición, en nuestro caso es Europe-West2 (Londres) en la zona c (por la eficiencia en sus discos) y una instancia mediana de 4GiB de RAM para que no sea muy cara, con una imagen de Ubuntu 18.04 LTS. Podemos controlar el precio en cualquier momento con los datos que nos brinda el asistente, es una cantidad estimada porque Google trabaja con la facturación por segundos para que si tu desactivas la máquina o recibes menos flujo de datos no estés pagando por un servicio inutilizado.

Nombre ?
El nombre es permanente.

Etiquetas ? (Opcional)

Región ?
La región es permanente.

europa-west2 (Londres)

Zona ?
La zona es permanente.

europa-west2-c

Configuración de la máquina

Familia de máquinas

Uso general Optimizada para la computación

Con memoria optimizada

Tiempo de máquinas para cargas de trabajo habituales, optimizadas en cuanto al coste y a la flexibilidad

Serie

E2

La plataforma de CPU se elige según las que haya disponibles

Tipo de máquina

e2-medium (2 vCPU, 4 GB de memoria)

	vCPU	Memoria	GPUs
	1 núcleo compartido	4 GB	-

Plataforma de CPU y GPU

Servicio de VM confidencial ?

☐ Habilitar el servicio de computación confidencial en esta instancia de VM.

Contenedor ?

☐ Desplegar una imagen de contenedor en esta instancia de VM. [Más información](#)

Disco de arranque ?

Nuevo disco persistente estándar de 10 GB

Imagen

Ubuntu 18.04 LTS

Cambiar

Precio mensual estimado: 31,99 \$

Eso significa 0,044 \$ por hora

Paga por lo que uses: facturación por segundos, sin gastos por adelantado.

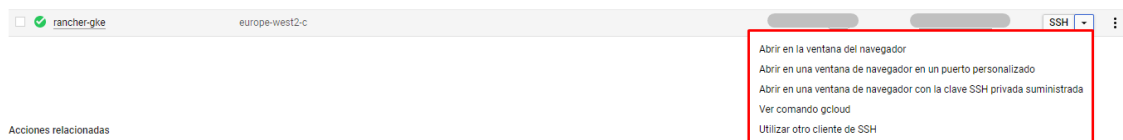
Elemento	Costes estimados
1 vCPU compartida + 4 GB de memoria	31,51 \$/mes
Disco persistente estándar de 10 GB	0,48 \$/mes
Descuento por uso continuado ?	- 0,00 \$/mes
Total	31,99 \$/mes

[Precios de Compute Engine](#)

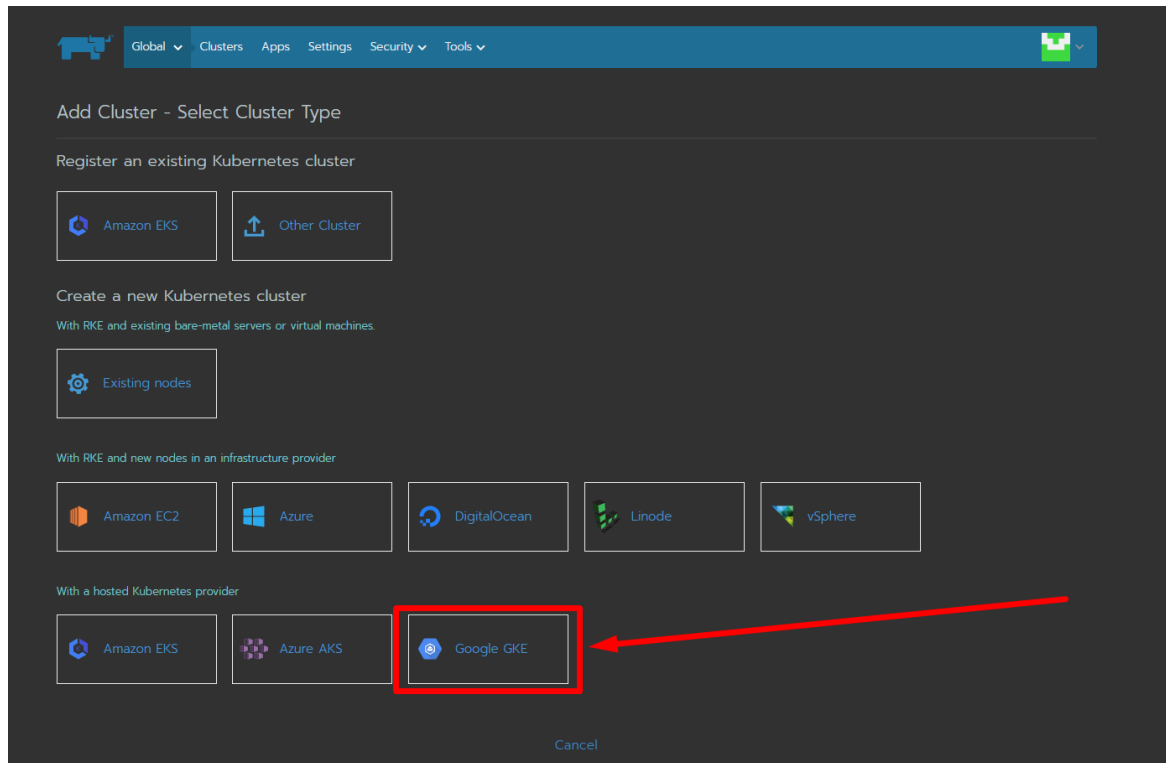
[Menos](#)

Una vez lo tenemos bien definido todo, creamos la instancia.

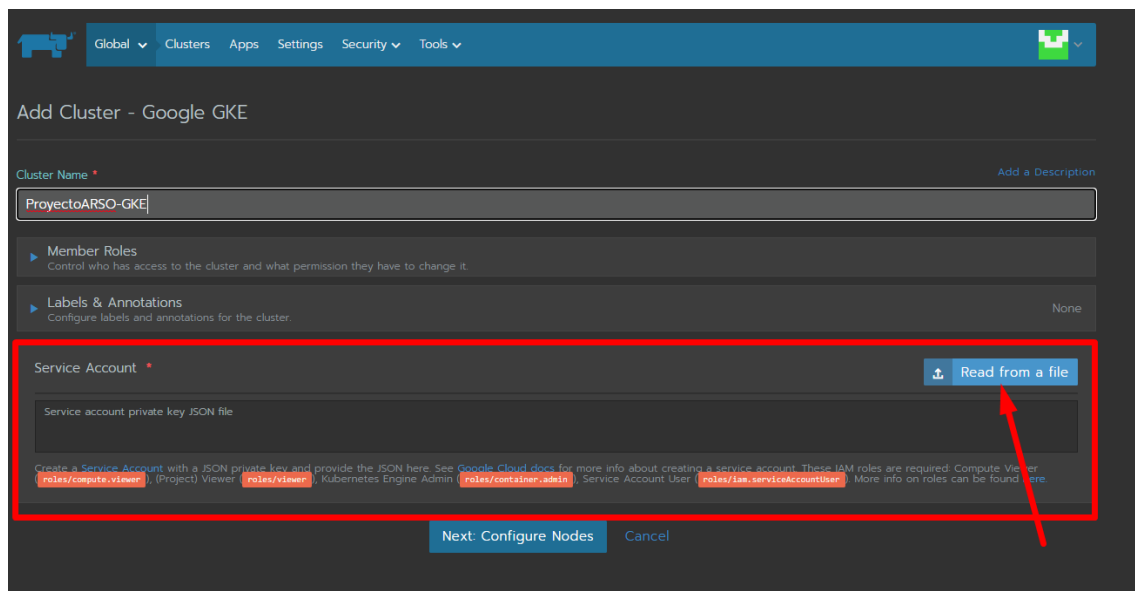
Esperamos un poco a que nos asignen la red, y entramos en ella mediante **ssh** para instalar Docker y posteriormente la imagen de Rancher. Una forma rápida de acceder mediante **SSH** es hacerlo mediante el comando en GCloud, para ello deberemos seguir los pasos que nos da Google para utilizarlo.



Una vez tengamos configurado Rancher, debemos de crear un clúster en Google GKE.



Y una vez dentro, asignamos un nombre e insertamos la clave que anteriormente hemos conseguido al asignar los permisos y crear una clave para el Bot.



Una vez insertado y validado, nos aparecerán las demás configuraciones de nuestro clúster.

En estas configuraciones tendremos que elegir la zona de nuestros nodos, la versión de Kubernetes, la red en la que se va a ubicar en nuestro Google Cloud, el número de nodos y su tamaño entre otras configuraciones.

Nuestra configuración es la que se ve en la siguiente imagen:

The image displays two screenshots of the Google Cloud Kubernetes Engine configuration interface, showing various settings for a new cluster.

Top Screenshot: Kubernetes Options

- Location Type:** ☒ Zonal, ☐ Regional
- Zone:** europe-west2-c (Additional Zones: europe-west2-a, europe-west2-b)
- Kubernetes Version:** 117.15-gke.800
- Container Address Range:** e.g. 10.42.0.0/16
- Alpha Features:** ☐ Enabled, ☒ Disabled
- Stackdriver Logging:** ☒ Enabled, ☐ Disabled
- Kubernetes Dashboard:** ☐ Enabled, ☒ Disabled
- Horizontal Pod Autoscaling:** ☒ Enabled, ☐ Disabled
- Network:** default
- Node Subnet:** Auto Create Subnetwork
- Legacy Authorization:** ☐ Enabled, ☒ Disabled
- Stackdriver Monitoring:** ☒ Enabled, ☐ Disabled
- Http Load Balancing:** ☒ Enabled, ☐ Disabled
- Maintenance Window:** Any Time
- Ip Aliases:** ☒ Enabled, ☐ Disabled
- Pod address range:** e.g. 10.96.0.0/11
- Service address range:** e.g. 10.94.0.0/18

Bottom Screenshot: Node Configuration

- Node Count:** 2
- Machine Type:** e2-small (Efficient Instance, 2 vCPU (1/4 shared physical core) and 2 GB RAM)
- Image Type:** Ubuntu
- Root disk type:** Standard persistent disk
- Root Disk Size:** 100 GB
- Local SSD disks:** 0 GB
- Preemptible nodes (beta):** ☐ Enabled, ☒ Disabled
- Auto Upgrade:** ☐ Enabled, ☒ Disabled
- Auto Repair:** ☐ Enabled, ☒ Disabled
- Node Pool Autoscaling:** ☐ Enabled, ☒ Disabled
- Taints:** + Add Taint
- Node Labels:** + Add Label
- Security Options:**
 - Service Account:** Compute Engine default service account
 - Access scopes:** ☒ Allow default access, ☐ Allow full access to all Cloud APIs, ☐ Set access for each API

Un aspecto para tener en cuenta es que, si señalamos regional o las 3 zonas, el número de zonas será multiplicado por el número de nodos que seleccionemos, es decir, si seleccionamos 2 nodos, y regional en Europe-West2 se nos crearán 6 nodos, uno en cada zona “a, b, c” en este caso.

En el último punto de la imagen, “Security Options” tenemos el acceso a la API, por tanto, hay que saber bien qué permisos le estamos dando a nuestro Clúster, por si hubiese algún problema, este acceso tenerlo controlado, en nuestro caso hemos dejado que Google con su configuración por defecto, porque nos ha parecido segura. Para ver qué acceso tiene el miembro de la API solo hay que ir a la consola de Google Cloud y observarlo desde IAM y Administración.

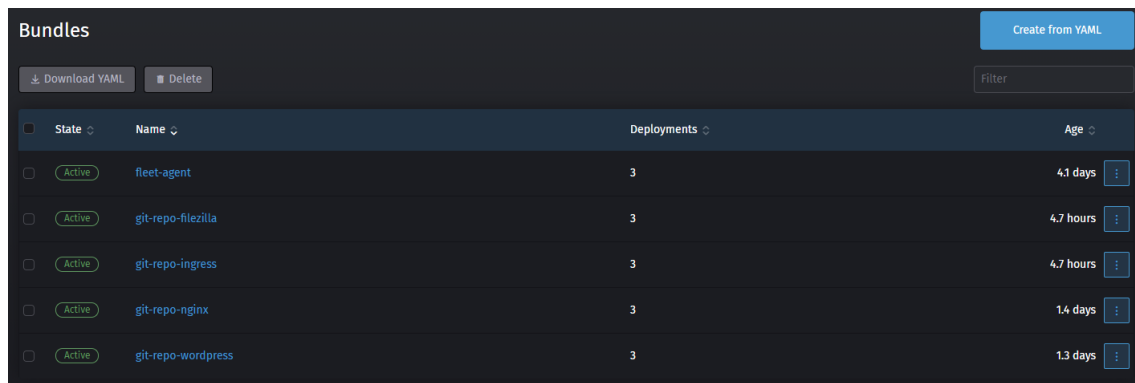
<input type="checkbox"/>	Tipo	Miembro ↑	Nombre	Rol	Permisos analizados (excedentes/total) ?
<input type="checkbox"/>		@developer.gserviceaccount.com	Compute Engine default service account	Editor	3418/3425

Una vez configurado todo, bastará con clicar en “Create” y esperar a que se cree nuestro nuevo clúster.

State	Name	Roles	Version	CPU	RAM	Pods
Active	gke-c-p9c9q-default-0-503af778-2v7q 10.13.48.2	Worker	v117.14-gke.1600 19.3.2	0.6/0.9 Cores	0.6/11 GiB	13/110
Active	gke-c-p9c9q-default-0-503af778-pl93 10.13.48.3	Worker	v117.14-gke.1600 19.3.2	0.5/0.9 Cores	0.5/11 GiB	12/110

○ Creación de servicios

Para crear los servicios en Google Cloud lo haremos de la misma forma que se han creado en OpenStack accediendo al menú de Fleet.



The screenshot shows the 'Bundles' page in Rancher. At the top right is a 'Create from YAML' button. Below it are 'Download YAML' and 'Delete' buttons, and a 'Filter' input field. The main table has columns for 'State', 'Name', 'Deployments', and 'Age'. All bundles listed are in an 'Active' state.

State	Name	Deployments	Age
Active	fleet-agent	3	4.1 days
Active	git-repo-filezilla	3	4.7 hours
Active	git-repo-ingress	3	4.7 hours
Active	git-repo-nginx	3	1.4 days
Active	git-repo-wordpress	3	1.3 days

Tenemos implementados un servicio de FileZilla, un servicio de Nginx, un balanceo de carga de Nginx y un WordPress.

○ Puntos que considerar

A la hora de generar clústeres en Google Cloud lo que más debemos tener en cuenta es que Kubernetes en cierto grado pertenece o ha pertenecido a Google, por lo tanto, la integración con toda la infraestructura es muy alta.

A la hora de crear clústeres Kubernetes en Google Cloud a través de Rancher nos damos cuenta que esta vez no nos deja elegir qué propiedades tienen estos nodos, como pasa en OpenStack o en Azure que nos deja elegir entre Worker, Control Plane y Etcd, para tener más o menos control sobre los nodos, y Kubernetes, en este caso, automáticamente se nos asignan nodos Workers y no podemos modificarlo, este hecho tiene lugar porque Google se reserva estos dos tipos para que sea gestionado desde el propio Google y así hacernos un poco más “dependiente” a Google Cloud como gestor de servicios, y para facilitar la monitorización y gestión de los nodos por si surgen errores en algún punto.

○ Troubleshooting

Como se ha mencionado anteriormente, Google tiene una alta integridad en Rancher y en Kubernetes, por lo tanto, la implementación y el despliegue en el proveedor es bastante sencilla, aunque sí que hay algunos puntos a considerar.

- Los permisos para trabajar con la API son a veces un tanto confusos, porque sin querer podemos quedarnos cortos dando permisos o dar demasiados, es por ello

por lo que está bien repasar la documentación y ver cuáles son las exigencias de Rancher para crear Clústeres y gestionarlos.

- A veces, la aprobación de una clave puede hacerse larga en tiempo y, por tanto, no poder crear los clústeres todo lo rápido que querríamos, o simplemente una de las API de Google se ha quedado desactivada por un error y entonces tengamos que ir y activarla, es común que Kubernetes Engine se desactive o no lo hayamos habilitado correctamente.
- El acoplamiento a redes ya creadas a la hora de crear un clúster tiende a dar error, por lo que debemos tenerlo en cuenta si queremos comunicar clústeres en la misma red.
- Si sobrecargamos la red del nodo, *Kubelet* cierra ese nodo hasta que se descongestione, pudiendo tirar el resto de los servicios de ese clúster si se desincronizan.



○ Conclusión

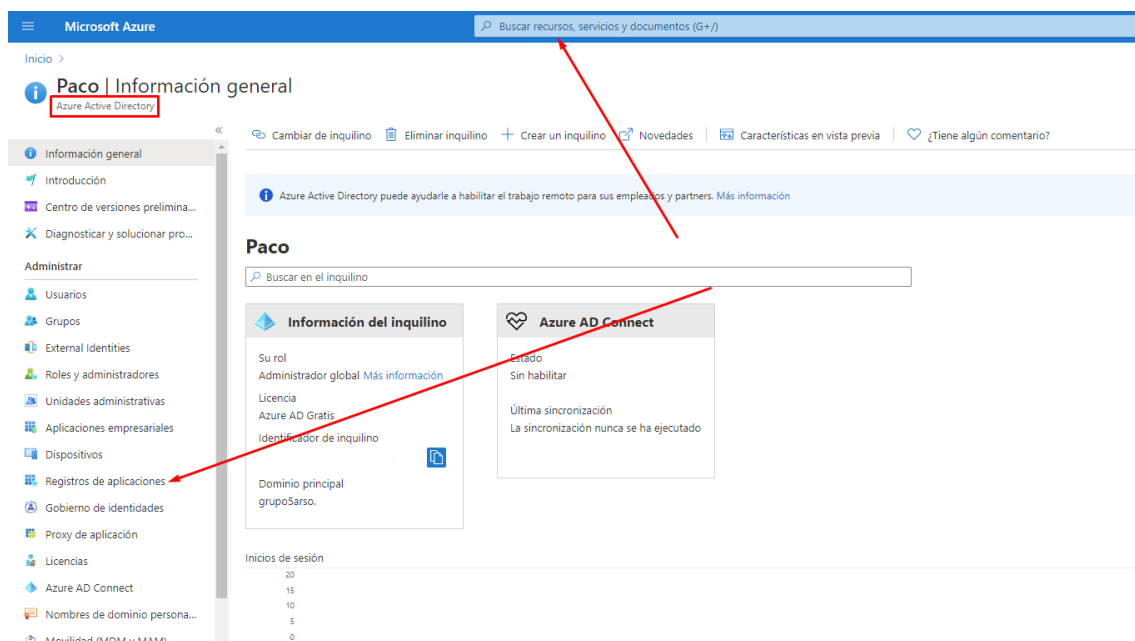
Google Cloud es un proveedor que ha sido fácil de implementar, fácil de gestionar y fácil de trabajar con él, pero a la vez se puede llegar a echar en falta poder gestionar algunas de las máquinas por tu mano.



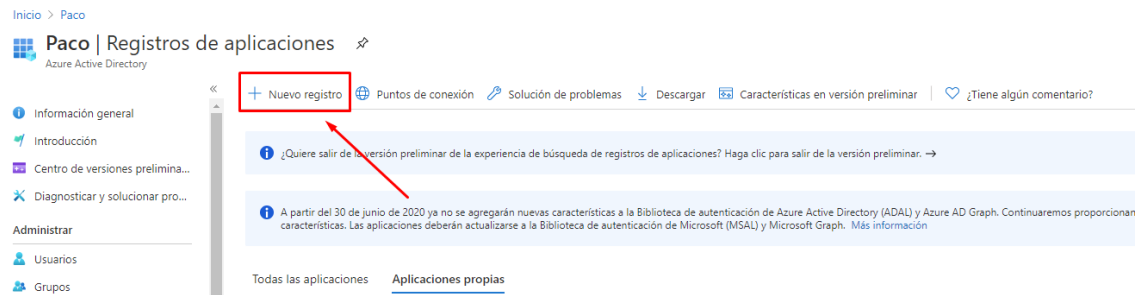
Como dato curioso, hay que añadir que Google tiene monitorizando los servidores en todo momento, por lo que, si usas un nodo para minar criptomonedas, aunque sea por error o un virus (o conscientemente), Google no dudará en cerrar ese servidor y la cuenta si se repite esta acción.

- Despliegue en Azure
 - Preparación del entorno

Para empezar con Azure como proveedor deberemos tener una suscripción activa y acceso por parte de la administración a *Azure Active Directory*. Una vez en este apartado de Azure tendremos que crear una nueva aplicación, mediante **Registro de aplicaciones**.



Para crear nuestra aplicación debemos darle al botón de **Nuevo Registro** para comenzar.



Le debemos asignar un nombre, a quién le vamos a dar acceso y una dirección. La configuración de a quién le vamos a dar acceso debe ser Cuentas de cualquier directorio organizativo porque con este clúster vamos a usarlo con el mismo inquilino, pero si quisiéramos adoptar en Rancher clústeres de otras cuentas de Azure o nos fuera a adoptar otra cuenta de Azure necesitamos que tenga acceso a nuestra app.

Inicio > Paco >

Registrar una aplicación

* Nombre

Nombre para mostrar accesible por los usuarios de esta aplicación. Se puede cambiar posteriormente.

warso

Tipos de cuenta compatibles

¿Quién puede usar esta aplicación o acceder a esta API?

- ☐ Solo cuentas de este directorio organizativo (solo de Paco: inquilino único)
- ☒ Cuentas en cualquier directorio organizativo (cualquier directorio de Azure AD: multiinquilino)
- ☐ Cuentas en cualquier directorio organizativo (cualquier directorio de Azure AD: multiinquilino) y cuentas de Microsoft personales (como Skype o Xbox)
- ☐ Solo cuentas personales de Microsoft

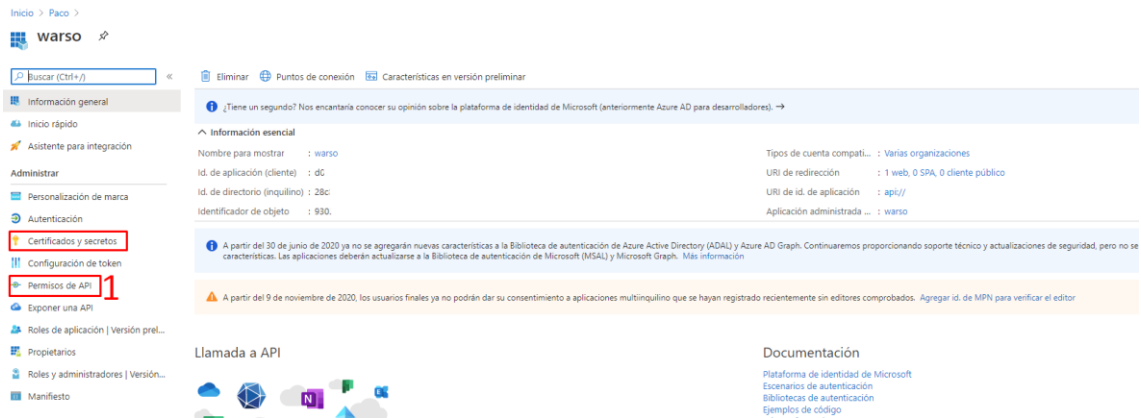
[Ayudarme a elegir...](#)

URI de redirección (opcional)

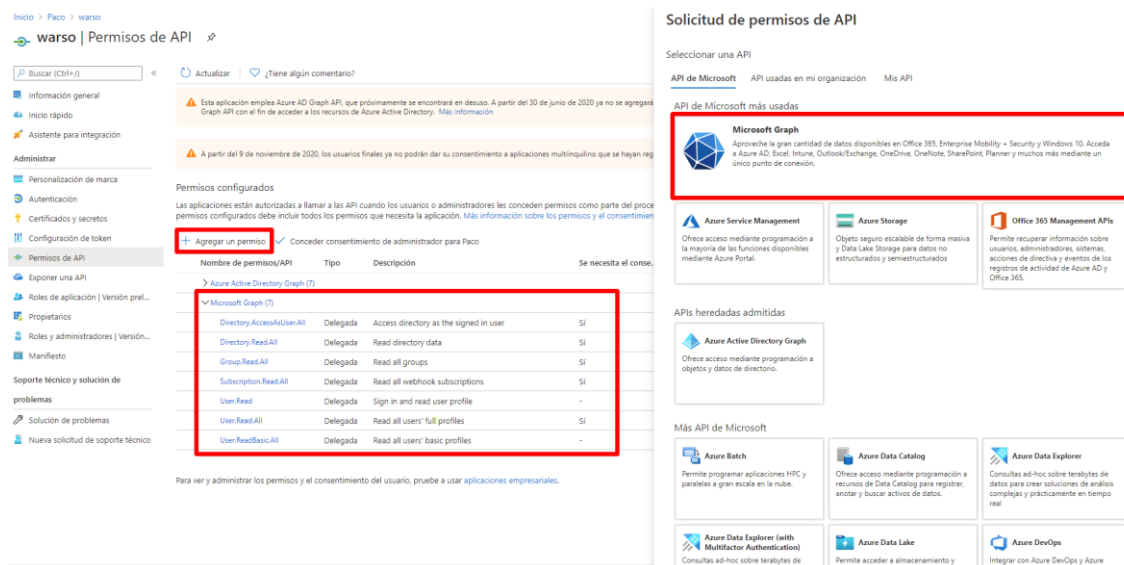
Devolveremos la respuesta de autenticación a esta dirección URI después de autenticar correctamente al usuario. Este dato es opcional y se puede cambiar más tarde, pero se necesita un valor para la mayoría de los escenarios de autenticación.

Web https://proyectoarso

Después tenemos que ir a la parte de **Permisos de API** para configurar qué accesos tiene la app creada a nuestro *Azure Active Directory*.



En nuestro caso, para que una aplicación pueda crear Clústeres en nuestro nombre necesitamos darle los permisos de API de *Microsoft Graph*, para ello elegimos **Agregar un permiso** y agregamos los que aparecen en la lista de la imagen.



La interfaz que nos debe aparecer es la siguiente:

Solicitud de permisos de API

[← Todas las API](#)

Microsoft Graph

<https://graph.microsoft.com/> [Docs](#) [↗](#)

¿Qué tipo de permiso necesita la aplicación web?

Permisos delegados

Su aplicación necesita acceder a la API como el usuario que haya iniciado la sesión.

Permisos de la aplicación

Su aplicación se ejecuta como servicio en segundo plano o demonio sin un usuario que haya iniciado la sesión.

Seleccionar permisos

[expandir todo](#)

Permiso	Se necesita el consentimiento del ...
✓ Permisos de OpenId	
<input type="checkbox"/> email ⓘ View users' email address	-
<input type="checkbox"/> offline_access ⓘ Maintain access to data you have given it access to	-
<input type="checkbox"/> openid ⓘ Sign users in	-
<input type="checkbox"/> profile ⓘ View users' basic profile	-
> AccessReview	
> AdministrativeUnit	
> AgreementAcceptance	
> Agreement	

Para poder completar los requisitos de Rancher necesitaremos también asignar una clave de acceso a nuestra APP, ara ello nos iremos al apartado de **Certificados y secretos** y allí crearemos un **Nuevo secreto de cliente**

Inicio > Paco > warso

warso | Certificados y secretos

Buscar (Ctrl+/) « ¿Tiene algún comentario?

Información general

Inicio rápido

Asistente para integración

Administrar

Personalización de marca

Autenticación

Certificados y secretos

Configuración de token

Permisos de API

Exponer una API

Roles de aplicación | Versión prel...

Propietarios

Roles y administradores | Versión...

Manifiesto

Soporte técnico y solución de problemas

Solución de problemas

Nueva solicitud de soporte técnico

Las credenciales permiten a las aplicaciones confidenciales identificarse con el servicio de autenticación al recibir tokens y una ubicación web direccionable (con un esquema HTTPS). Para obtener un mayor nivel de garantía, le recomendamos que use un certificado como credencial, en lugar de un secreto de cliente.

Certificados

Los certificados pueden usarse como secretos para probar la identidad de la aplicación al solicitar un token. También se conocen como claves públicas.

Cargar certificado

Huella digital	Fecha de inicio	Expira	Id.
No se ha agregado ningún certificado para esta aplicación.			

Secretos de cliente

Se trata de una cadena de secreto que la aplicación usa para probar su identidad al solicitar un token. También se conoce como contraseña de aplicación.

+ Nuevo secreto de cliente

Descripción	Expira	Valor	Id.
Password uploaded on Sat Jan 16 2021	31/12/2299	*****	7a14f161-7.
Password uploaded on Sat Jan 16 2021	31/12/2299	*****	5078b178-7C

Le podremos añadir una descripción y el tiempo de expiración.

Agregar un secreto de cliente

Descripción

Expira

☒ En 1 año

☐ En 2 años

☐ Nunca

Agregar Cancelar

Una vez creado **guardaremos** la clave a buen recaudo para que no haya problemas.

Para finalizar la zona de permisos en Azure tendremos que dar a la aplicación permisos de acceso a la suscripción activa, para ello, iremos al apartado de **suscripciones**, después iremos a **Control de acceso (IAM)** y agregaremos una asignación de roles, el rol será colaborador para poder ver la suscripción pero no modificarla, el acceso será para Usuarios, grupo o entidad de servicio, donde buscaremos el nombre de la aplicación, la elegiremos y la guardaremos.

1. Buscar recursos, servicios y documentos (G+/)

2. Control de acceso (IAM)

3. Agregar asignaciones de roles

4. Agregar asignación de roles

¿Quiere conceder acceso a este recurso?

Para conceder acceso a los recursos, asigne un rol.

Agregar asignaciones de roles Más información

Consulta del acceso a este recurso

Consulte las asignaciones de roles que conceden acceso a este y a otros recursos.

Ver Más información

Ver asignaciones de denegación

Consulte las asignaciones de roles a las que se les ha denegado el acceso a acciones específicas en este ámbito.

Ver Más información

Creación de un rol personalizado

Cree un rol personalizado para los recursos de Azure con su propio conjunto de permisos para satisfacer las necesidades específicas de su organización.

Miembros seleccionados:

No se seleccionó ningún miembro. Busque y agregue uno o varios miembros que quiera asignar al rol de este recurso.

Más información sobre RBAC

Guardar Descartar

Una vez hemos terminado la implementación de los permisos de nuestra aplicación de Azure vamos a crear una máquina virtual para que contenga a Rancher. Para ello nos vamos a ir a la sección de máquinas virtuales, para acceder la buscamos en la barra de búsqueda y una vez allí seleccionamos **Agregar** y **Máquina virtual**

1. Buscar recursos, servicios y documentos (G+/)

2. Máquinas virtuales

3. Agregar

Máquinas virtuales

+ Agregar

+ Máquina virtual

Suscripciones: Azure para estudiantes - ¿No ve ninguna suscripción? Abrir la configuración Directorio + suscripción

Filtrar por nombre...

Todos los grupos de recursos

Todos los tipos

Todas las ubicaciones

Todas las etiquetas

Nombre	Tipo	Estado	Grupo de recursos	Ubicación	Origen	Estado de
azure-wor1	Máquina virtual	En ejecución	DOCKER-MACHINE	Oeste de Europa	Marketplace	-
azure-wor2	Máquina virtual	En ejecución	DOCKER-MACHINE	Oeste de Europa	Marketplace	-
RancherAzure	Máquina virtual	En ejecución	RancherAzure_group	Sur de Reino Unido	Marketplace	-

Elegimos la suscripción, el grupo de recursos que nosotros queramos, el nombre de la máquina, el lugar y el Sistema Operativo, después seleccionamos el tamaño y elegimos el que más nos convenga. Para finalizar este apartado seleccionamos un usuario y una contraseña para nuestro equipo.

Microsoft Azure

Buscar recursos, servicios y documentos

[Inicio](#) > [Máquinas virtuales](#) >

Crear una máquina virtual

Detalles del proyecto

Seleccione la suscripción para administrar recursos implementados y los costes. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción * ⓘ

Azure para estudiantes

Grupo de recursos * ⓘ

docker-machine

Crear nuevo

Detalles de instancia

Nombre de máquina virtual * ⓘ

rancher

Región * ⓘ

(Europe) Oeste de Europa

Opciones de disponibilidad ⓘ

Zona de disponibilidad

Zona de disponibilidad * ⓘ

1

Imagen * ⓘ

Ubuntu Server 18.04 LTS - Gen1

Ver todas las imágenes

Instancia de Azure de acceso puntual ⓘ

☐

Tamaño * ⓘ

Standard_D2s_v3 - 2 vcpu, 8 GiB de memoria (73,87 €/mes)

Ver todos los tamaños

Cuenta de administrador

Tipo de autenticación ⓘ

☐ Clave pública SSH

☒ Contraseña

Nombre de usuario * ⓘ

rancher

Contraseña * ⓘ

.....

Confirmar contraseña * ⓘ

.....

Revisar y crear

< Anterior

Siguiente: Discos >

En la parte de redes tenemos que dar acceso a los puertos 22, 80 y 443.

[Inicio](#) > [Máquinas virtuales](#) >

Crear una máquina virtual


[Datos básicos](#) [Discos](#) [Redes](#) [Administración](#) [Opciones avanzadas](#) [Etiquetas](#) [Revisar y crear](#)

Configure la tarjeta de interfaz de red (NIC) a fin de definir la conectividad de red para la máquina virtual. Puede controlar los puertos y la conectividad entrante y saliente con reglas de grupos de seguridad o bien aplicar una solución de equilibrio de carga ya existente. [Más información](#)

Interfaz de red

Al crear una máquina virtual, se crea una interfaz de red automáticamente.

Red virtual *	<div><div>docker-machine-vnet</div><div>▼</div></div> <div>Crear nuevo</div>
Subred *	<div><div>docker-machine (192.168.0.0/16)</div><div>▼</div></div> <div>Administrar configuración de subred</div>
IP pública	<div><div>(nuevo) rancher-ip</div><div>▼</div></div> <div>Crear nuevo</div>
Grupo de seguridad de red de NIC	<div><div><input type="radio"/> Ninguno</div><div><input checked="" type="radio"/> Básico</div><div><input type="radio"/> Opciones avanzadas</div></div>
Puertos de entrada públicos *	<div><div><input type="radio"/> Ninguno</div><div><input checked="" type="radio"/> Permitir los puertos seleccionados</div></div>
Seleccionar puertos de entrada *	<div><div>HTTP (80), HTTPS (443), SSH (22)</div><div>▼</div></div>

 **Esto permitirá que todas las direcciones IP accedan a la máquina virtual.**
Esto solo se recomienda para las pruebas. Use los controles avanzados de la pestaña Redes a fin de crear reglas para limitar el tráfico entrante a las direcciones IP conocidas.

Redes aceleradas ☐ El tamaño de máquina virtual seleccionado no admite redes aceleradas.

[Revisar y crear](#)[< Anterior](#)[Siguiente: Administración >](#)

Si no tenemos nada más que configurar, en nuestro caso no tocamos nada más de las configuraciones, terminamos dándole a **Revisar y crear** donde nos darán un precio por hora de nuestro servicio.

[Inicio](#) > [Máquinas virtuales](#) >

Crear una máquina virtual

✓ Validación superada

[Datos básicos](#) [Discos](#) [Redes](#) [Administración](#) [Opciones avanzadas](#) [Etiquetas](#) [Revisar y crear](#)

DETALLES DEL PRODUCTO

Standard D2s v3

por Microsoft

[Términos de uso](#) | [Directiva de privacidad](#)

Se aplican créditos de suscripción ⓘ

0,1012 EUR/h

[Precios de otros tamaños de máquinas virtuales](#)

TÉRMINOS

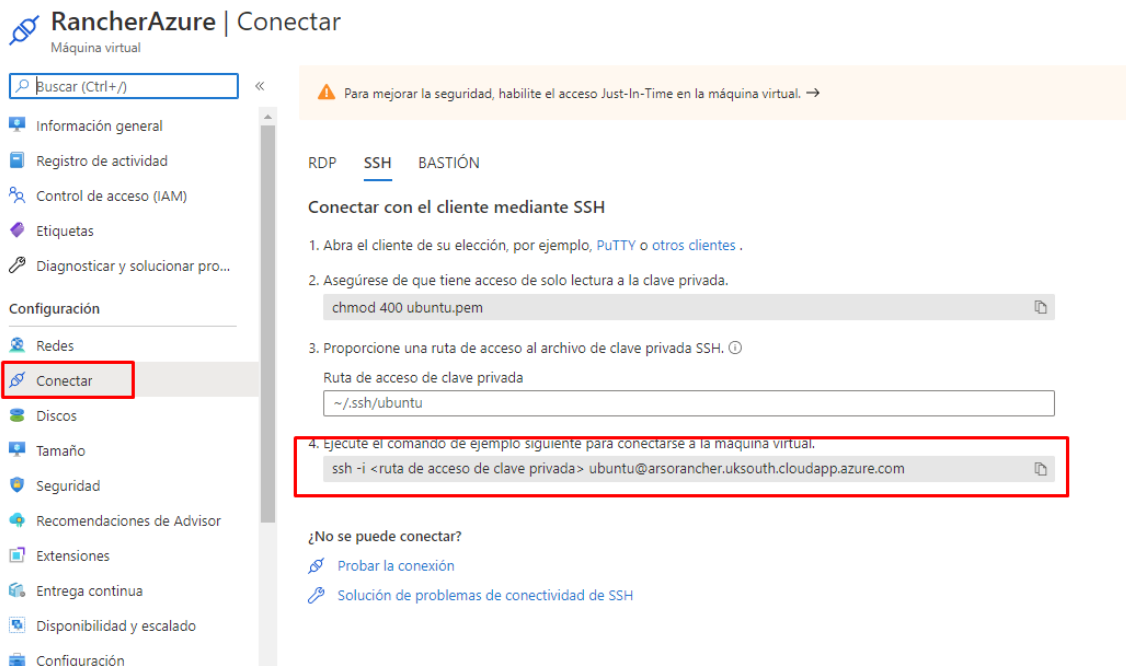
Al hacer clic en "Crear", (a) acepto los términos legales y las declaraciones de privacidad relacionados con cada oferta de Marketplace que se enumeró previamente; (b) autorizo a Microsoft a facturar con mi método de pago actual las cuotas relacionadas con las ofertas, con la misma frecuencia de facturación que mi suscripción de Azure; y (c) autorizo a Microsoft a compartir mi información de contacto y los datos de transacción y uso con los proveedores de dichas ofertas. Microsoft no proporciona derechos sobre ofertas de terceros. Para obtener información adicional, consulte los [Términos de Azure Marketplace](#).

⚠ **Ha establecido los siguientes puertos abiertos para Internet: SSH.** Esto solo se recomienda para las pruebas. Si quiere cambiar esta configuración, vuelva a la pestaña de aspectos básicos.

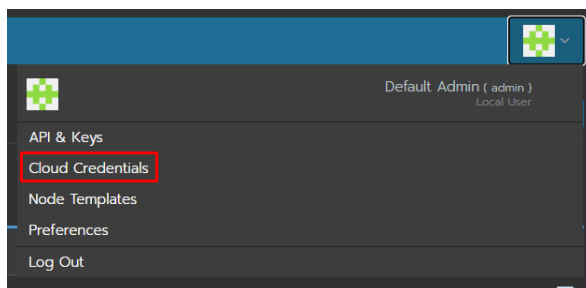
Datos básicos

Suscripción	Azure para estudiantes
Grupo de recursos	docker-machine
Nombre de máquina virtual	rancher
Región	Oeste de Europa
Opciones de disponibilidad	Zona de disponibilidad
Zona de disponibilidad	1
Imagen	Ubuntu Server 18.04 LTS - Gen1

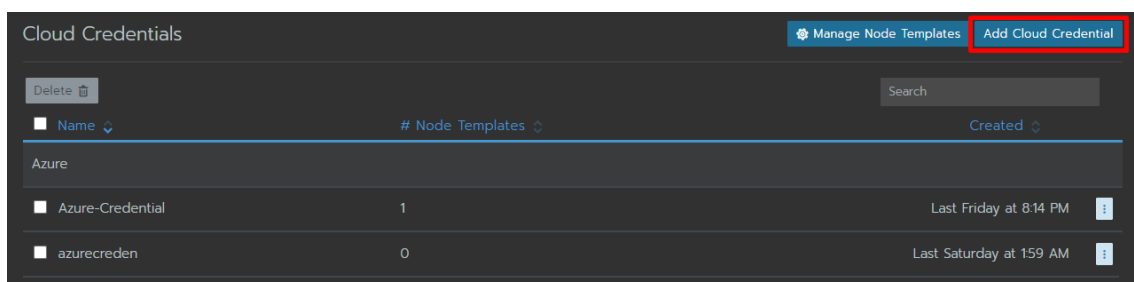
Tras esperar un poco de tiempo podremos acceder al servicio de Azure mediante SSH yendo al apartado de **Conectar** ubicado en la máquina virtual.



Para que ahora Rancher tenga acceso a nuestro proyecto tenemos que darle las credenciales y crear una plantilla, para ello nos vamos a la información que hay en la parte superior derecha de la pantalla y clicamos en nuestro usuario, ahí nos aparecerá **Cloud Credentials**.

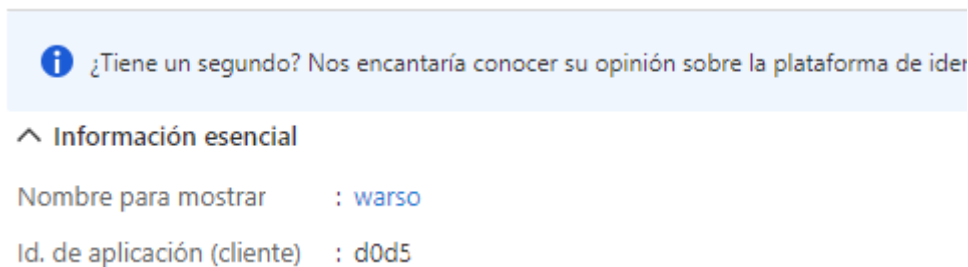


La credencial de Azure viene activada por defecto, por lo que le damos a **Add Cloud Credential**.

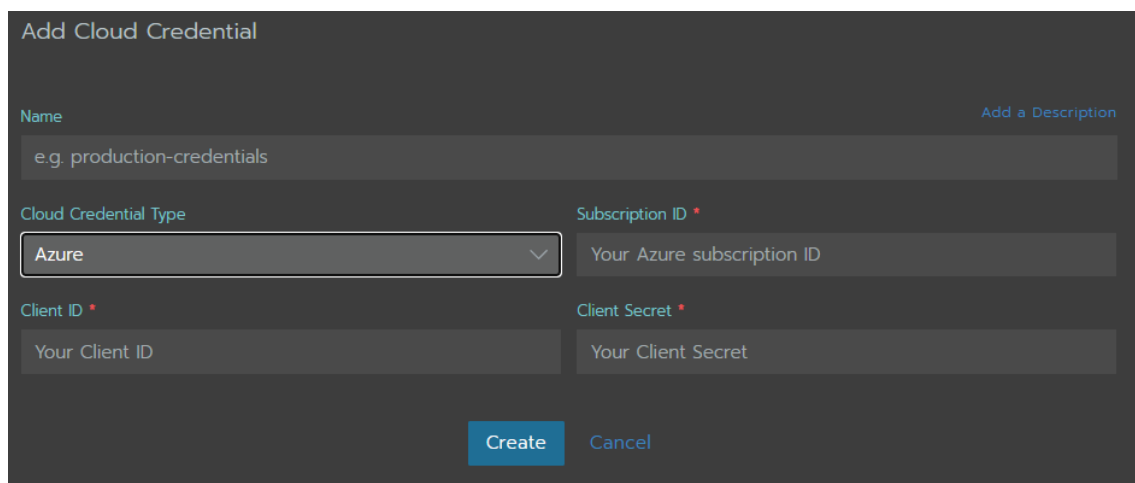


Una vez empecemos a crearla le damos un nombre y seleccionamos **Azure** como *Cloud Credential Type*.

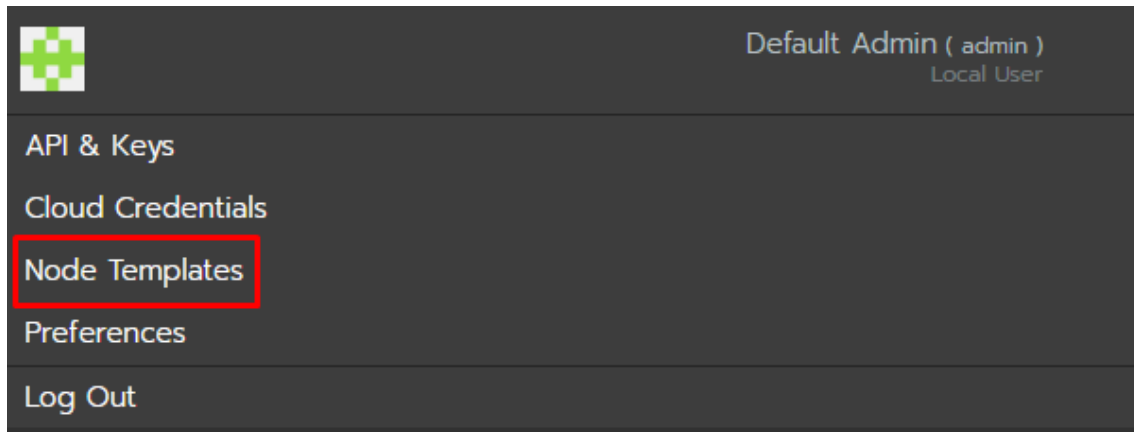
En el apartado de *Subscription ID* vamos a insertar el ID de la suscripción que podemos encontrar en el apartado de **Suscripciones** de portal Azure. El ID de cliente va a ser el ID de aplicación que nos encontramos en *Azure Active Directory* en nuestra aplicación en la información de cabecera de nuestra aplicación.



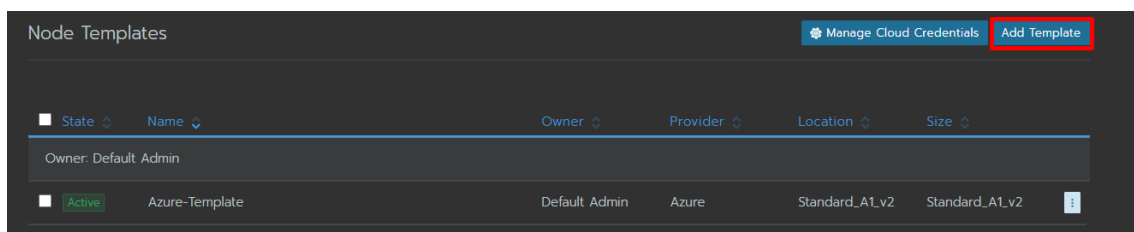
El *Client Secret* que nos queda por poner es el secreto que anteriormente creamos en las **credenciales y secretos** de la aplicación de Azure que viene dado como *Value*. Creamos la Credencial.



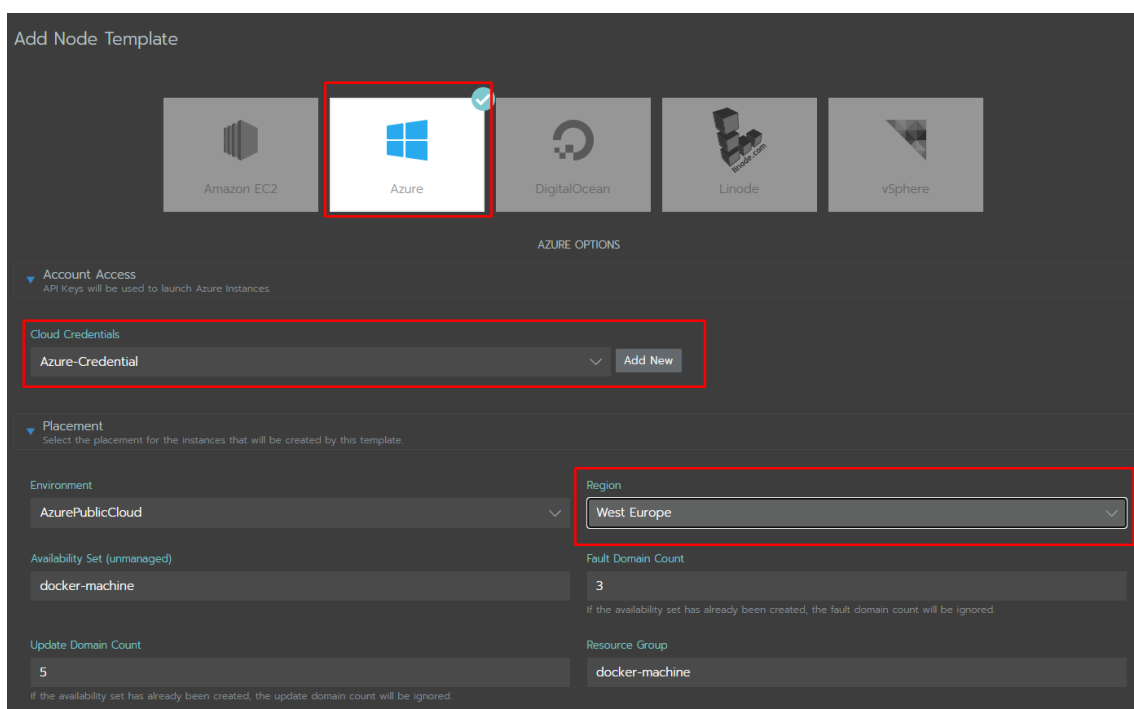
Tras haber creado la credencial tenemos que crear un **Node Template** para agilizar la creación de clústeres en nuestro entorno de Azure. Para ello, en la interfaz de usuario de Rancher, seleccionaremos la opción de **Node Templates**.



Allí añadiremos una nueva plantilla para la gestión de Azure clicando en **Add Template**.



Una vez dentro de la plantilla elegimos Azure como Principal configuración y vamos rellenando los campos como más nos interese, nuestra configuración es la siguiente.



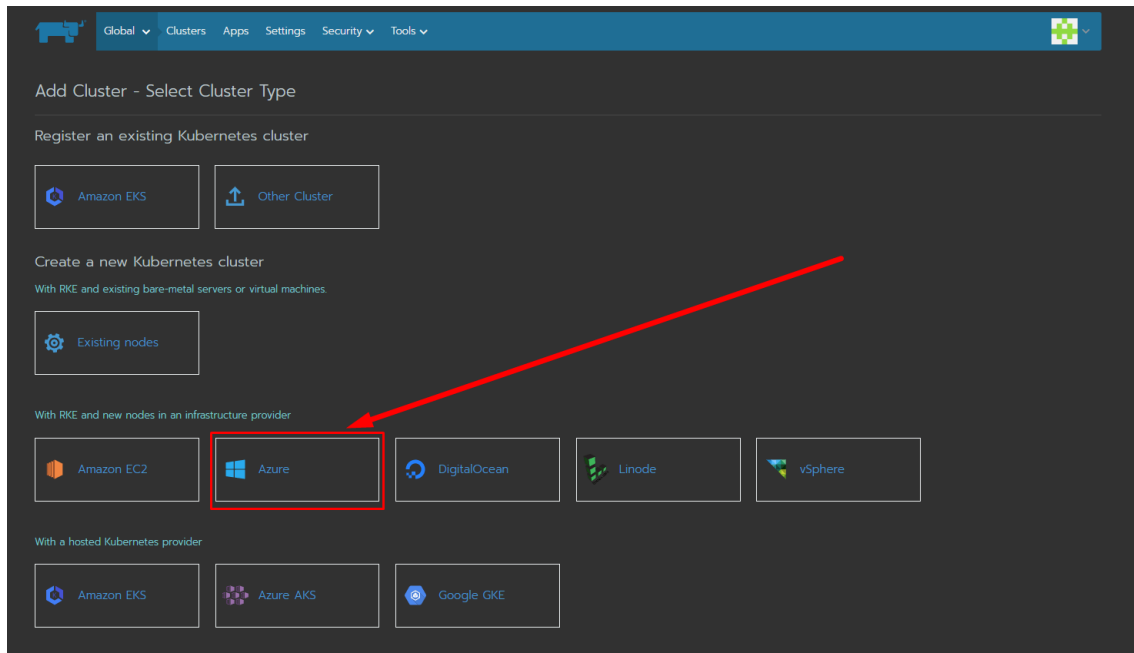
En la sección de Red elegimos le subnet que prefiramos, si no está creada se creará, y el grupo de seguridad que queramos.

The screenshot shows the 'Network' configuration section for an Azure template. It includes fields for Subnet (docker-machine), Subnet Prefix (192.168.0.0/16), VNet (docker-machine-vnet), Public IP (Static), Private IP (127.0.0.1), and Network Security Group (rancher-managed-0a3urmRT). A note at the bottom states: 'When using a Rancher managed or providing an existing NSG, all nodes using this template will use the supplied NSG. If no NSG is provided, a new NSG will be created for each node.'

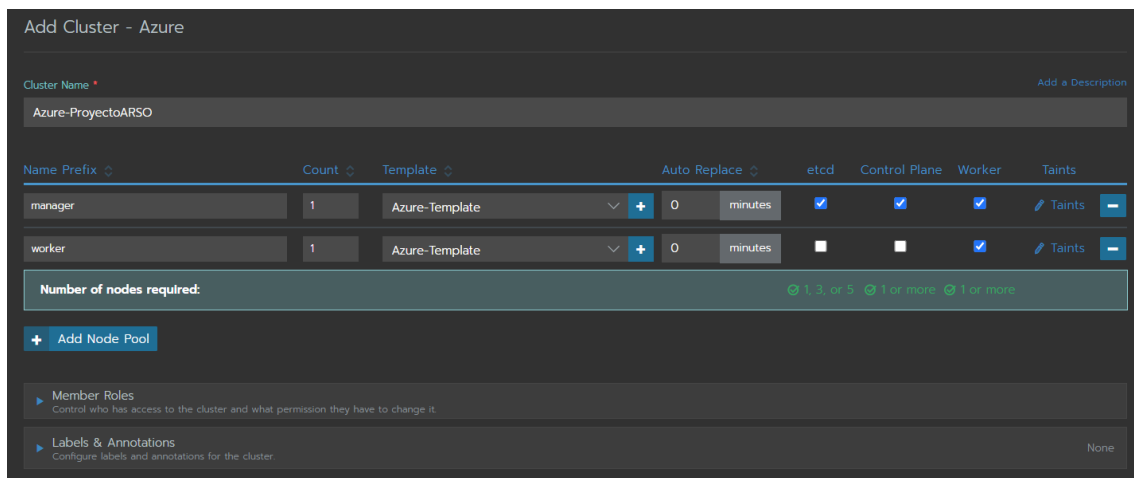
Utilizamos Unmanaged Disk porque los Managed nos han traído problemas en la configuración de los Clúster. Y una memoria de 30GB porque Kubernetes no podía soportar menos.

The screenshot shows the 'Instance' configuration section for an Azure template. It includes fields for Image (canonical/UbuntuServer:18.04-LTS:latest), Size (Standard_D2_v2), Docker Port (2376), Open Port (6443/tcp,2379/tcp,2380/tcp,8472/udp,4789/udp,9796/tcp,10256/tcp,10250/tcp,10251/tcp), SSH User (docker-user), Storage Type (Standard LRS), Disk Type (Unmanaged Disk), and Disk Size (30). A note at the bottom states: 'Creates a unmanaged availability set. Changing this value after the availability set has been created may cause errors.'

Una vez completado, nos vamos al menú principal y procedemos a Añadir un nuevo clúster. Seleccionamos Azure como Tipo de Clúster.



Rellenamos los datos de nuestros nodos, podemos elegir entre nodos Worker, ETCD o Control Plane, siempre siguiendo las restricciones que nos da el asistente.



Elegimos la versión de Kubernetes y **muy importante**, hay que elegir **Azure** como *In Tree Cloud Provider* y rellenar al menos la información que pide de forma obligatoria para que cuando se configuren los nodos, puedan comunicarse bien entre ellos en el proyecto y puedan determinar el uso del proveedor de forma correcta.

Respecto las anteriores configuraciones, solamente nos falta el *tenantId* que está en la información de nuestra aplicación junto a el Id de Cliente.

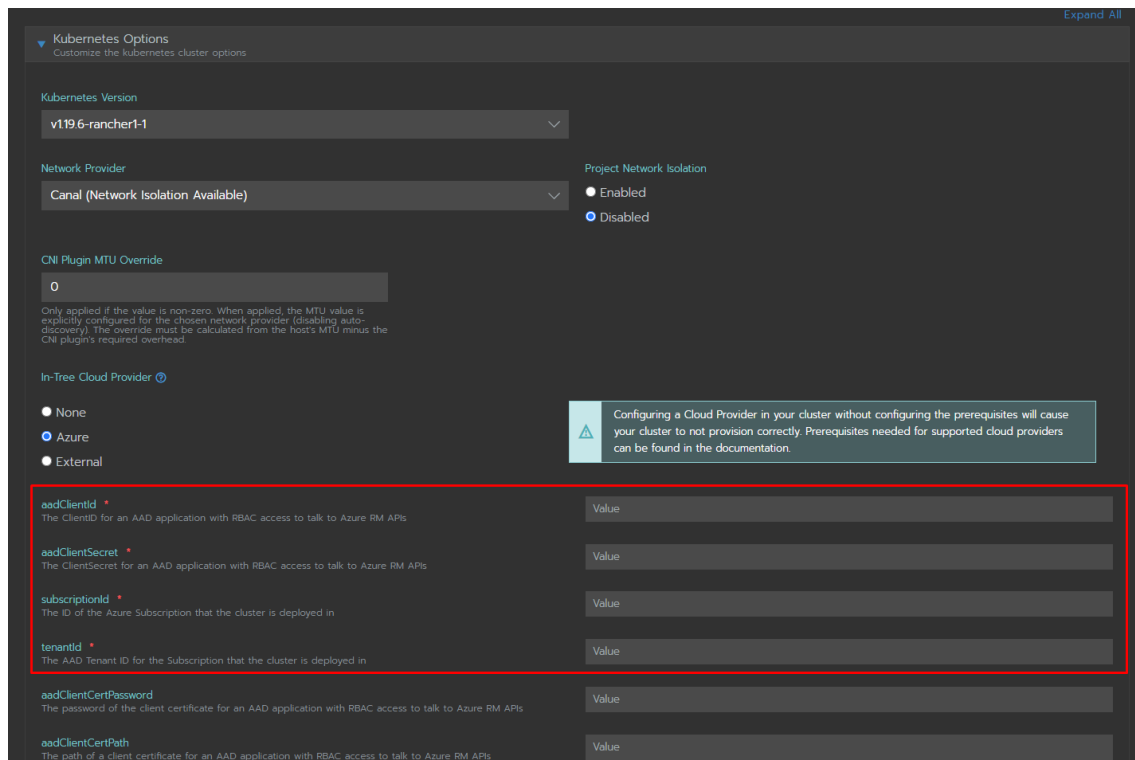
^ Información esencial

Nombre para mostrar : warso

Id. de aplicación (cliente) : d0d57b38-b.

Id. de directorio (inquilino) : 28c6

Una vez sabemos este dato, procederemos a rellenar con los datos que nos faltan, que ya sabemos de los apartados anteriores.



Kubernetes Options
Customize the kubernetes cluster options

Kubernetes Version
v19.6-rancher1-1

Network Provider
Canal (Network Isolation Available)

Project Network Isolation
☐ Enabled
☒ Disabled

CNI Plugin MTU Override
0
Only applied if the value is non-zero. When applied, the MTU value is explicitly configured for the chosen network provider (disabling auto-discovery). The override must be calculated from the host's MTU minus the CNI plugin's required overhead.

In-Tree Cloud Provider ⓘ
☐ None
☒ Azure
☐ External

Warning: Configuring a Cloud Provider in your cluster without configuring the prerequisites will cause your cluster to not provision correctly. Prerequisites needed for supported cloud providers can be found in the documentation.

aadClientid *
The ClientID for an AAD application with RBAC access to talk to Azure RM APIs
Value

aadClientSecret *
The ClientSecret for an AAD application with RBAC access to talk to Azure RM APIs
Value

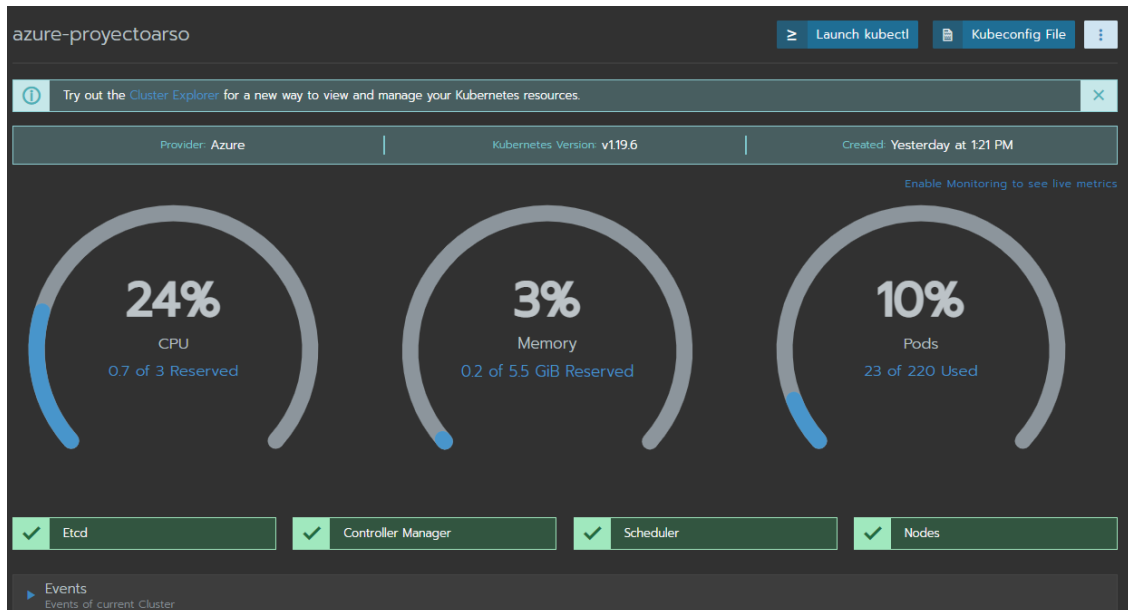
subscriptionId *
The ID of the Azure Subscription that the cluster is deployed in
Value

tenantId *
The AAD Tenant ID for the Subscription that the cluster is deployed in
Value

aadClientCertPassword
The password of the client certificate for an AAD application with RBAC access to talk to Azure RM APIs
Value

aadClientCertPath
The path of a client certificate for an AAD application with RBAC access to talk to Azure RM APIs
Value

Una vez terminada la configuración solo tendremos que esperar y tendremos nuestro clúster listo y funcional para aplicarle servicios.



○ Troubleshooting

- Hay que prestar atención en los errores que nos da Rancher en la creación de los nodos, ya que son muy precisos y a partir de ahí podemos adivinar o *forear* qué le pasa al nodo en cuestión.
- A veces queremos ahorrar en la aplicación de recursos de nuestros hosts, entonces escatimamos en RAM o CPU e introducimos demasiados servicios, a causa de esto en nuestros nodos se puede llegar a producir un desbordamiento de memoria y dejarán de comunicarse con Rancher un tiempo indefinido, Rancher nos dará un error de **Comunicación la API** si esto pasa, la solución que debemos aplicar es reiniciar la máquina y ver qué servicios están drenando a nuestros nodos, en el caso que lo requiera habrá que ampliar los recursos a nuestros nodos.

○ Conclusión

Azure no es igual que Google, eso lo sabemos, entonces debemos tener en cuenta dependiendo del uso que le vayamos a dar a nuestros servicios cuál de los proveedores nos conviene. En este caso, Azure nos ha brindado unas características en cuando a manejo de los contenedores que Google nos limitaba, es decir, Azure nos deja tener el control total de nuestros contenedores de Kubernetes, dejándonos designar *ETCD*, *Control Plane*, y *Worker* en nuestros nodos, esto es un avance para usuarios experimentados en Kubernetes.

En cuando al precio de los recursos, al ser un proyecto de corta vida, no hemos tenido tiempo de apreciar bien cuál de los dos proveedores ha consumido más crédito, pero a primera vista, Azure parece ser más caro por los planes que tiene.

En cuando a navegabilidad por su interfaz de APIs, Azure tiene su parte buena en la que mediante *Active Directory* podemos manejar casi todos los aspectos que necesitamos para la creación de clústeres, pero a la vez, la asignación de permisos y demás sin una previa formación puede ser una tarea muy tediosa.

En general, la experiencia con Azure ha sido buena, pero ha habido aspectos que hay que para encontrar una solución se ha tenido que navegar por los foros y por la documentación de Azure de una manera muy confusa.

4. Conclusión

Kubernetes es una forma muy útil de manejar contenedores, además de ser producto de Google lo cual está hecho para ser un producto compatible y con mucho soporte. A la par, Rancher es una herramienta muy útil para manejar varios de estos clústeres, en nuestro proyecto el despliegue de aplicaciones sobre varios Kubernetes es tan fácil como una pequeña configuración de los nodos y clúster (al inicio del uso de Rancher), preparar la aplicación en los archivos yaml y asignar por Fleet donde quieres que se desplieguen. Pero en este proyecto se ha hecho palpable que estas herramientas en pleno cambio de dirección, Rancher está dejando obsoletas varias de las funciones que hemos usado como la configuración In-tree del cloud provider por otra alternativa más compleja que es desarrollada ahora por el proveedor más que por la herramienta. Otras funciones como Fleet nos han resultado más que útiles aun siendo de las ultimas actualizaciones.

El despliegue de microservicios con estas herramientas aun con todo es tremendamente sencillo y la experiencia con estas herramientas ayuda a entender como funcionan varios de los sistemas reales usados actualmente y por qué están en alza, pero es un mundo ahora mismo en constante cambio.

En definitiva, este proyecto nos ha sido muy útil para ampliar los conocimientos aprendidos en la asignatura, y para aprender tecnologías actuales. Nos hemos quedado con ganas de ampliar aún más nuestros conocimientos del estilo de los servicios y los monitoreos, además, el trabajo en grupo nos ha ayudado a consolidarnos y ha servido para aprender de nosotros mismos y los demás.

5. Bibliografía

1&1 IONOS España S.L.U. (2020, 9 diciembre). *DNS*. IONOS Digitalguide.

<https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/registro-a/>

colaboradores de Wikipedia. (2021, 12 enero). *Sistema de nombres de dominio*.

Wikipedia, la enciclopedia libre.

https://es.wikipedia.org/wiki/Sistema_de_nombres_de_dominio

containerd. (s. f.). Containerd. <https://containerd.io/>

CoreOS. (s. f.). *CoreOS*. <https://coreos.com/rkt/>

Deployments. (2020, 8 diciembre). Kubernetes.

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

Dynamically Provisioning New Storage in Rancher. (s. f.). Rancher Labs.

<https://rancher.com/docs/rancher/v2.x/en/cluster-admin/volumes-and-storage/provisioning-new-storage/#1-add-a-storage-class-and-configure-it-to-use-your-storage>

Example: Deploying WordPress and MySQL with Persistent Volumes. (2020, 20 junio).

Kubernetes. <https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/>

Kubernetes. (2019, 16 julio). *The Future of Cloud Providers in Kubernetes*.

<https://kubernetes.io/blog/2019/04/17/the-future-of-cloud-providers-in-kubernetes/>

Kubernetes. (2020, 30 mayo). *Entender los Objetos de Kubernetes*.

<https://kubernetes.io/es/docs/concepts/overview/working-with-objects/kubernetes-objects/>

Kubernetes in-tree providers · Kontena Pharos. (s. f.). k8spharos.

https://docs.k8spharos.dev/cloud_providers/kube-in-tree.html

Layer 4 and Layer 7 Load Balancing. (s. f.). Rancher Labs. Recuperado 26 de

diciembre de 2020, de <https://rancher.com/docs/rancher/v2.x/en/k8s-in-rancher/load-balancers-and-ingress/load-balancers/>

M. (2020, 21 septiembre). *Creación dinámica de volúmenes de Azure Disks - Azure*

Kubernetes Service. Microsoft Docs. <https://docs.microsoft.com/es-es/azure/aks/azure-disks-dynamic-pv>

Microsoft. (2020, 20 junio). *Solucionar problemas comunes de Azure Kubernetes*

Service - Azure Kubernetes Service. Microsoft Docs.

<https://docs.microsoft.com/es-es/azure/aks/troubleshooting>

Openstack Cloud Provider. (s. f.). Rancher Labs. Recuperado 21 de diciembre de 2020,

de <https://rancher.com/docs/rke/latest/en/config-options/cloud-providers/openstack/>

Persistent Volumes. (2021, 10 enero). Kubernetes.

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

R. (s. f.). *rancher/fleet*. GitHub. <https://github.com/rancher/fleet>

Rancher Labs. (s. f.-a). *Azure Cloud Provider*.

<https://rancher.com/docs/rke/latest/en/config-options/cloud-providers/azure/>

Rancher Labs. (s. f.-b). *Quick Start*. Recuperado 20 de diciembre de 2020, de

<https://rancher.com/quick-start/>

Rancher Labs. (2019, 30 abril). *Load Balancing with Kubernetes: concepts, use cases, and implementation details*. YouTube.

<https://www.youtube.com/watch?v=FG0ZW5eX1JY>

Torres, M. (2019a, noviembre). *ualmtorres/CursoKubernetes*. GitHub.

<https://github.com/ualmtorres/CursoKubernetes>

Torres, M. (2019b, noviembre 28). *Kubernetes. Un orquestador de contenedores que debes poner en tu vida*. Github Pages.

<https://ualmtorres.github.io/SeminarioKubernetes/>

webdevops/php-nginx — Dockerfile Documentation documentation. (s. f.). readthedocs.

<https://dockerfile.readthedocs.io/en/latest/content/DockerImages/dockerfiles/php-nginx.html>