

Contenedores

Los contenedores ofrecen un modo estándar de empaquetar código, configuraciones y dependencias de una aplicación en un único objeto. Éstos contenedores comparten un sistema operativo instalado en el servidor y se ejecutan como procesos aislados de los recursos, garantizando así implementaciones rápidas, fiables y consistentes.

Azure

Cuando buscamos información acerca de contenedores en Azure, encontramos que hay cinco posibles formas de implementarlos: mediante la creación de un clúster de Kubernetes con Azure Kubernetes Service (AKS), mediante la creación de una aplicación en contenedores con Azure Web App para contenedores, mediante la creación de un registro de Docker privado en Azure Container Registry, mediante la ejecución a petición de aplicaciones de contenedor en Azure Container Instances o mediante la implementación de una aplicación contenedora Windows con Service Fabric.

Creación de un clúster de Kubernetes con Azure Kubernetes Service (AKS)

Lo primero que he hecho es comprobar la versión de docker que tengo instalada. En este apartado estoy utilizando macOS Sierra debido a que para instalar docker en Windows se necesita que éste sea Windows 10 Pro, el cuál no poseo.

```
iMac-de-Angela:~ angelamorenos$ docker --version
Docker version 18.09.2, build 6247962
```

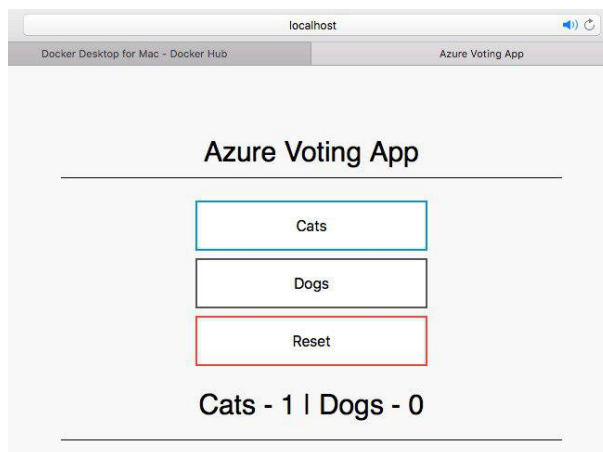
A continuación voy a desplegar un contenedor a partir de unas imágenes de un repositorio en github que ofrece una aplicación de votos online. Para ello, clonamos el repositorio.

```
iMac-de-Angela:~ angelamorenos$ git clone https://github.com/Azure-Samples/azure-voting-app-redis.git
```

A continuación, mostramos los contenedores que hay en ejecución.

```
iMac-de-Angela:~ angelamorenos$ cd azure-voting-app-redis
iMac-de-Angela:azure-voting-app-redis angelamorenos$ ls
LICENSE          azure-vote-all-in-one-redis.yaml
README.md        docker-compose.yaml
azure-vote       jenkins-tutorial
```

El front de la aplicación está corriendo en localhost, en el puerto 8080. Por tanto, si vamos al explorador e intentamos acceder al puerto 8080 del localhost debería de aparecer la aplicación:



A continuación, voy a mostrar de nuevo los contenedores que hay en ejecución:

```
iMac-de-Angela:azure-voting-app-redis angelamorenos$ docker container ls
CONTAINER ID   IMAGE     PORTS                NAMES                CREATED
439695be98e8   redis    0.0.0.0:6379->6379/tcp  azure-vote-back      3 hours ago
3b380df7b8c1   azure-vote-front  "/entrypoint.sh /sta..." 3 hours ago
Up 3 hours    443/tcp, 0.0.0.0:8080->80/tcp  azure-vote-front
```

Voy a intentar borrar uno de ellos, en concreto azure-vote-back:

```
iMac-de-Angela:azure-voting-app-redis angelamorenos$ docker container rm azure-vote-back
Error response from daemon: You cannot remove a running container 439695be98e8bd40d9af979c3d8177040623cb4c37d60049885f974d4c764d2f. Stop the container before attempting removal or force remove
```

Como se puede ver en la imagen, no deja borrar un contenedor que se esté ejecutando a no ser que lo paremos o que forcemos su borrado. Por tanto, fuerzo su borrado:

```
iMac-de-Angela:azure-voting-app-redis angelamorenos$ docker container rm -f azure-vote-back
azure-vote-back
iMac-de-Angela:azure-voting-app-redis angelamorenos$ docker container ls
CONTAINER ID   IMAGE     PORTS                NAMES                CREATED
3b380df7b8c1   azure-vote-front  "/entrypoint.sh /sta..." 3 hours ago
Up 3 hours    443/tcp, 0.0.0.0:8080->80/tcp  azure-vote-front
```

Como se puede ver, el contenedor "azure-vote-back" ya no está activo, y si intentamos acceder de nuevo a la página donde está la aplicación nos vamos a encontrar con que ya no funciona:

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Creación de una aplicación en contenedores con Azure Web App for Containers

App Service Linux proporciona pilas de aplicaciones predefinidas en Linux con compatibilidad con .NET, PHP y Node.js entre otros. En el ejemplo que viene a continuación se va a crear una web e implementar una imagen de GO desde Docker Hub. Para ello, voy a utilizar la CLI de Azure.

Crear un grupo de recursos Un grupo de recursos es un contenedor lógico en el que se implementan y administran recursos de Azure como aplicaciones web, bases de datos y cuentas de almacenamiento.

Para crear dicho grupo de recursos, en la CLI de Azure, se introduce la siguiente instrucción:

```
az group create --name miGru --location "West Europe"
```

El grupo creado se llama "miGru" y tiene como localización el oeste de Europa. Como respuesta a la instrucción ejecutada la CLI de Azure mostrará un JSON donde hará saber que la instrucción ha sido ejecutada correctamente y se ha creado el grupo.

```
moreno@Azure:~$ az group create --name miGru --location "West Europe"
{
  "id": "/subscriptions/b34b87a1-4b15-49a7-8a95-1af1f3771d19/resourceGroups/miGru",
  "location": "westeurope",
  "managedBy": null,
  "name": "miGru",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": null
}
```

Crear un plan de Azure App Service Se crea un plan App Service a través de la CLI de Azure:

```
az appservice plan create --name miPlan
--resource-group miGru --sku B1 --is-linux
```

El plan creado se llama miPlan y pertenece al grupo creado anteriormente. Con el valor "B1" asignamos un plan de tarifa básico y con --is-linux determinamos

que es un contenedor Linux.

```
moreno@Azure:~$ az appservice plan create --name miPlan --resource-group miGru --sku
B1 --is-linux
{
  "freeOfferExpirationTime": "2019-11-25T20:46:25.0333333",
  "geoRegion": "West Europe",
  "hostingEnvironmentProfile": null,
  "hyperV": false,
  "id": "/subscriptions/b34b87a1-4b15-49a7-8a95-1af1f3771d19/resourceGroups/miGru/pr
oviders/Microsoft.Web/serverfarms/miPlan",
  "isSpot": false,
  "isXenon": false,
```

Creación de una aplicación web Ahora, a través de la CLI, se va a crear una aplicación web.

```
az webapp create --resource-group miGru --plan miPlan --name
nombreApplication --deployment-container-image-name
microsoft/azure-appservices-go-quickstart
```

Con el comando `--deployment-container-image-name` apuntamos a la imagen pública de Docker Hub.

```
moreno@Azure:~$ az webapp create --resource-group miGru --plan miPlan --name nombrea
pplication --deployment-container-image-name microsoft/azure-appservices-go-quicksta
rt
{
  "availabilityState": "Normal",
  "clientAffinityEnabled": true,
  "clientCertEnabled": false,
  "clientCertExclusionPaths": null,
  "cloningInfo": null,
  "containerSize": 0,
  "dailyMemoryTimeQuota": 0,
  "defaultHostName": "nombreaapplication.azurewebsites.net",
  "enabled": true,
```

Navegación hasta la aplicación Debido a que el nombre elegido para el nombre de la aplicación web es "nombreApplication", la URL que deberemos de introducir en el navegador para ir a nuestra página web es la siguiente:

`http:nombreaapplication.azurewebsites.net/hello`

Mostrándose en el navegador lo siguiente:

The screenshot shows a web browser window. The address bar contains the URL `http://nombreaapplication.azurewebsites.net/hello`. Below the address bar, the text `Hello, world!` is displayed.

Ejecución a petición de aplicaciones de contenedor en Azure Container Instances

En este apartado se va a usar la CLI de Azure para implementar un contenedor de docker aislado y hacer que la aplicación esté disponible con un nombre de dominio completo.

Creación de un grupo de recursos

```
az group create --name myGroup --location "west Europe"
```

Creación de un contenedor Para crear una instancia de contenedor de la CLI de Azure hay que proporcionar un nombre al grupo de recursos, un nombre de instancia de contenedor y una imagen de contenedor de Docker. En este ejemplo se usa una pública, que empaqueta una aplicación en Node.js que sirve una página HTML estática.

Los contenedores se pueden exponer en Internet mediante la especificación para que se abran uno o varios puertos.

Mediante la etiqueta `-dns-name-label` se le establece un valor DNS.

```
az container create --resource-group myGroup --name mycontainer --image
mcr.microsoft.com/azuredocs/aci-helloworld
--dns-name-label acidemo --ports 80
```

A continuación, comprobamos el estado:

```
az container show --resource-group myGroup --name mycontainer
--query "FQDN:ipAddress.fqdn,ProvisioningState:provisioningState-out table"
```

FQDN	ProvisioningState
acidemo.westeurope.azurecontainer.io	Succeeded

El estado es `SSucceeded`", por lo que sí introducimos el FQDN en el navegador, lo que se muestra es lo siguiente:



Extraer los registros del contenedor

```
az container logs --resource-group myGroup --name mycontainer
```

```
C:\Users\angel>az container logs --resource-group myGroup --name mycontainer
Listening on port 80
:ffff:10.240.255.56 - - [25/May/2019:22:35:16 +0000] "GET / HTTP/1.1" 200 1663 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36"
:ffff:10.240.255.56 - - [25/May/2019:22:35:16 +0000] "GET /favicon.ico HTTP/1.1" 404 150 "http://acidemo.westeurope.azurecontainer.io/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36"
```

Se muestran los registros del contenedor, además de las solicitudes HTTP GET generadas al ver la aplicación en el explorador.

Creación de un registro de Docker privado en Azure Container Registry

Para esta subsección, al contrario que en todas las anteriores desarrolladas, he tenido que instalarme la CLI de Azure, no pudiendo utilizar la versión Cloud.

Crear un grupo de recursos

```
az group create --name miGrupo --location "West Europe"
```

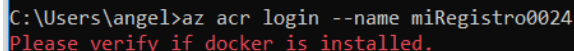
Crear un registro de contenedor

```
az acr create --resource-group miGrupo --name miRegistro0024 --sku Basic
```

Insertar la imagen en el registro

```
az acr login --name miRegistro0024
```

En este paso, sin embargo, se muestra lo siguiente por la CLI:

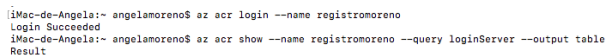


```
C:\Users\angel>az acr login --name miRegistro0024
Please verify if docker is installed.
```

Tal y como se puede ver, no se puede seguir adelante sin tener Docker instalado. El problema en este caso es que, al intentar instalar Docker una de las restricciones es que se debe de tener Windows 10 Pro, el cual yo no tengo.

Por tanto, se va a repetir todo el proceso desde el ordenador cuyo sistema operativo es macOS Sierra y donde sí que pude instalar Docker.

Antes de insertar imágenes hay que iniciar sesión en la instancia de Azure container Registry especificando el nombre del contenedor creado. En mi caso, y debido al cambio de ordenador, el nuevo nombre del registro es 'registromoreno'. También ejecuto un 'az acr show' para obtener el nombre del servidor de inicio de sesión completo de la instancia de Azure Container Registry.



```
iMac-de-Angela:~ angelamorenos$ az acr login --name registromoreno
Login Succeeded
iMac-de-Angela:~ angelamorenos$ az acr show --name registromoreno --query loginServer --output table
Result
-----
loginServer
```

A continuación se muestra la lista de imágenes locales con el comando 'docker images' y podemos que aparece 'aci-tutorial-app', que es la imagen creada en

apartados anteriores.

```
iMac-de-Angela:~ angelamorenos$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aci-tutorial-app	latest	5d94177ae362	About a minute ago	71MB
<none>	<none>	21f48f786ed1	About an hour ago	721MB
azure-vote-front	latest	18fc614c94a4	20 hours ago	952MB
tiangolo/uwsgi-nginx-flask	python3.6	0e5638c7a817	10 days ago	952MB
redis	latest	d3e3588af517	10 days ago	95MB
centos	7	9f38484d228f	2 months ago	282MB
node	8.9.3-alpine	144aaf4b1367	17 months ago	67.7MB

Como siguiente paso etiquetamos la imagen aci-tutorial-app.

```
iMac-de-Angela:~ angelamorenos$ docker tag aci-tutorial-app registromoreno.azurecr.io/aci-tutorial-app:v1
```

```
iMac-de-Angela:~ angelamorenos$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
aci-tutorial-app	latest	5d94177ae362	2 minutes ago	71MB
registromoreno.azurecr.io/aci-tutorial-app	v1	5d94177ae362	2 minutes ago	71MB
<none>	<none>	21f48f786ed1	About an hour ago	721MB
azure-vote-front	latest	18fc614c94a4	20 hours ago	952MB
tiangolo/uwsgi-nginx-flask	python3.6	0e5638c7a817	10 days ago	952MB
redis	latest	d3e3588af517	10 days ago	95MB
centos	7	9f38484d228f	2 months ago	282MB
node	8.9.3-alpine	144aaf4b1367	17 months ago	67.7MB

Inserción de imágenes en Azure Container Registry Ahora que se ha etiquetado la imagen con el nombre del servidor de inicio de sesión completo, se puede insertar en el registro con el comando 'docker push'.

```
iMac-de-Angela:~ angelamorenos$ docker push registromoreno.azurecr.io/aci-tutorial-app:v1
```

```
The push refers to repository (registromoreno.azurecr.io/aci-tutorial-app)
```

```
8119cbdb4863: Pushed
```

```
ec25618d9d2d: Pushed
```

```
e5859de8f58b: Pushed
```

```
1dfbdf388b77: Pushed
```

```
2ec948494cc8: Pushed
```

```
6dfaec39e726: Pushed
```

```
v1: digest: sha256:d1a23f655e64c5e56fe72f99accc7d5e8f889170fd0dec158df8f1df4e9946b size: 1577
```

Lista de imágenes en Azure Container Registry Ahora ejecuto el comando 'az acr repository list' para ver si la imagen se ha insertado correctamente. Y por último, con el comando 'az acr repository show-tags' hago que se imprima la etiqueta de la imagen.

```
iMac-de-Angela:~ angelamorenos$ az acr repository list --name registromoreno --output table
```

```
Result
```

```
-----
```

```
aci-tutorial-app
```

```
iMac-de-Angela:~ angelamorenos$ az acr repository show-tags --name registromoreno --repository aci-tutorial-app --output table
```

```
Result
```

```
-----
```

```
v1
```

AWS

Con respecto a AWS, los casos de uso de los contenedores son: microservicios, para aislar procesos; procesamiento de lotes, para arrancarlos con rapidez y escalarlos de forma dinámica dependiendo de la demanda; aplicaciones híbridas, puesto que los contenedores le permiten administrar de un modo uniforme la forma en que se implementa el código y la migración de aplicaciones a la nube, puesto que los contenedores facilitan la tarea de empaquetar aplicaciones enteras y trasladarlas a la nube sin cambiar nada del código. También permiten diseñar plataformas para que los desarrolladores no tengan que administrar infraestructuras y aprendizaje automático. A su vez, Amazon tiene diferentes servicios: Amazon ECR, Amazon ECS, Amazon EKS, AWS Fargate y Amazon EC2.

Amazon ECR

Servicio para almacenar imágenes en contenedores. Los desarrolladores pueden usar la CLI de Docker para diseñar, insertar, extraer y administrar imágenes.

Componentes Registro: Cada cuenta de AWS recibe un registro de Amazon ECR donde puede crear repositorios de imágenes y guardar imágenes en ellos.

Token de autorización: Debe autenticar el cliente de Docker en los registros de Amazon ECR como usuario de AWS para que dicho cliente pueda insertar y extraer imágenes. A través del comando `get-login` de la AWS CLI se proporcionan credenciales para pasarlas al Docker.

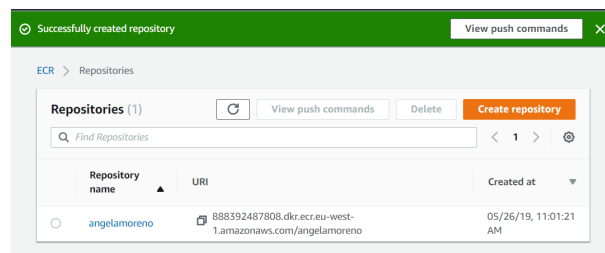
Repositorio: Contiene las imágenes de Docker.

Política sobre repositorios: Se puede controlar el acceso a los repositorios e imágenes.

Imagen: Se puede insertar y extraer imágenes de Docker en los repositorios y usarlas localmente.

Introducción a Amazon ECR En este apartado nos van a enseñar a crear un repositorio en la consola de Amazon ECR. Para ello, abro la consola de Amazon ECR y selecciono el nombre que va a tener el primer repositorio, que va a ser `angelamoreno`.

Una vez introducido el nombre y creado el repositorio, AWS muestra la siguiente interfaz:



Creación, etiquetado y envío de imágenes Docker Lo primero que hay que hacer es instalar la CLI de AWS. Una vez hecho, ejecutamos el comando `'aws configure'`.

Al ejecutar el comando `'aws configure'` nos va a pedir que introduzcamos una clave de acceso ID, una contraseña de acceso, el nombre de la región en la que se creó el repositorio y el tipo de output por defecto.

Para obtener el ID y la contraseña hay que ir al IAM Management Console de AWS, al apartado de Usuarios y generar una clave de acceso. El ID y la contraseña generadas en el momento son las que tenemos que introducir en la CLI de AWS.

A continuación, ejecutando el comando `'docker login'` se pedirán los credenciales de Docker.

```
iMac-de-Angela:~ angelamorenos$ docker login
Authenticating with existing credentials...
Stored credentials invalid or expired
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username (angela.morenop@edu.uah.es): wastelands
Password:
Login Succeeded
```

A partir de ese momento, a través de los comandos 'docker tag' y 'docker push' podremos insertar imágenes en el repositorio creado.

Introducción a Amazon ECS

Amazon Elastic Container (Amazon ECS) es un servicio de administración de contenedores muy escalable y rápido que facilita la tarea de ejecutar, detener y gestionar contenedores de Docker en un clúster de instancias de Amazon EC2. Para configurar un ECS hay que seguir los siguientes pasos: Primero creamos una cuenta de AWS en <http://aws.amazon.com/> y después darnos a nosotros mismos el permiso de ser administradores.

Nombre de usuario*

[Añadir otro usuario](#)

Seleccionar el tipo de acceso de AWS

Seleccione la forma en que estos usuarios accederán a AWS. Las claves de acceso y las contraseñas generadas automáticamente se proporcionan en el último paso. [Más información](#)

Tipo de acceso* ☐ Acceso mediante programación
Habilita una ID de clave de acceso y una clave de acceso secreta para el SDK, la CLI y la API de AWS, además de otras herramientas de desarrollo.

☒ Acceso a la consola de administración de AWS
Habilita una contraseña que permite a los usuarios iniciar sesión en la consola de administración de AWS.

Contraseña de la consola* ☐ Contraseña generada automáticamente
☒ Contraseña personalizada

☐ Mostrar contraseña

Requerir el restablecimiento de contraseña ☒ El usuario debe crear una contraseña nueva en el próximo inicio de sesión.
Los usuarios obtienen automáticamente la política [IAMUserChangePassword](#) que les permite cambiar su propia contraseña.

A continuación hay que crear un grupo, el cual va a tener de nombre Administrators donde, al usuario previamente creado, se le asignan unas políticas. Una vez asignadas y creado el usuario, la pantalla es la siguiente:

Añadir usuario(s) 1 2 3 4 5

Correcto

Ha creado correctamente los usuarios que se muestran a continuación. Puede ver y descargar las credenciales de seguridad de los usuarios. También puede enviar a los usuarios un correo electrónico con instrucciones para iniciar sesión en la consola de administración de AWS. Esta es la última vez que las credenciales estarán disponibles para descargarlas. Sin embargo, puede crear otras en cualquier momento.

Los usuarios con acceso a la consola de administración de AWS pueden iniciar sesión en: <https://986302467808.signin.aws.amazon.com/console>

[Descargar csv](#)

Usuario	Enviar instrucciones de inicio de sesión
Administrador	Enviar correo electrónico

Una vez creado el usuario, nos piden que accedamos a él.

Cuenta:

Nombre de usuario:

Contraseña:

Iniciar sesión

Tal y como se seleccionó en las opciones al crear el usuario, al iniciar sesión, el administrador tiene que cambiar la contraseña antigua por una nueva.

Cuenta de AWS 888392487808

Nombre de usuario de IAM Administrador

Contraseña anterior

Nueva contraseña

Volver a escribir la nueva contraseña

Confirmar cambio de contraseña

[Iniciar sesión utilizando credenciales de cuenta raíz](#)

Después de esto, ya podemos crear una VPC (nube virtual privada). Para ello, lo que hay que hacer es abrir la consola de Amazon VPC y lanzar el asistente de VPC, la cual tenga una única subred pública. También hay que ponerle un nombre, el cual va a ser angelamoreno. Una vez que se ha creado, la

16

pantalla que nos muestra AWS es la siguiente:



Introducción a Amazon EKS

Amazon Elastic Container Service for Kubernetes (Amazon EKS) es un servicio administrado que permite ejecutar Kubernetes AWS sin necesidad de crear ni mantener su propio plano de control de Kubernetes. Kubernetes es un sistema de código abierto para automatizar la implementación, escalado y administración de las aplicaciones en contenedores.

Amazon EKS detecta y reemplaza automáticamente las instancias del plano de control en mal estado, y proporciona actualizaciones de versiones y parches automatizados para ellas. También integra numerosos servicios AWS para ofrecer escalabilidad y seguridad a las aplicaciones, como IAM para la autenticación, Amazon VPC para el aislamiento...

Creación del clúster de Amazon EKS y los nodos de trabajo Para hacer esta parte de la práctica he tenido que instalar eksctl y kubectl.

Inicialmente creamos el clúster de Amazon EKS y los nodos de trabajo con el siguiente comando:

```
iMac-de-Angela:~ angelamorenos$ eksctl create cluster \
> --name prod \
> --version 1.12 \
> --nodegroup-name standard-workers \
> --node-type t3.medium \
> --nodes 3 \
> --nodes-min 1 \
> --nodes-max 4 \
[> --node-ami auto
```

Lanzar aplicación de libro de invitados Para lanzar la aplicación de libro de invitados hay que crear el controlador de replicación maestro Redis, además de crear el servicio, el controlador de replicación esclavo, el servicio esclavo Redis, también crear el controlador de replicación de guestbook y el servicio guestbook.

```
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/redis-master-controller.json
replicationcontroller "redis-master" created
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/redis-master-service.json
service "redis-master" created
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/redis-slave-controller.json
replicationcontroller "redis-slave" created
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/redis-slave-service.json
service "redis-slave" created
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/guestbook-controller.json
replicationcontroller "guestbook" created
iMac-de-Angela:~ angelamorenos$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/e
xamples/master/guestbook-go/guestbook-service.json
service "guestbook" created
```

Por último, hay que mirar la dirección externa del servicio guestbook a través del

```
iMac-de-Angela:~ angelamorenos$ kubectl get services -o wide
```

NAME	TYPE	PORT(S)	CLUSTER-IP	EXTERNAL-IP	SELECTOR
guestbook	LoadBalancer	10.100.214.50	10.100.214.50	a57c82d6c7ff01e9945c02233689946-425171167.eu-we	app=guestbook
st-1.elb.amazonaws.com	3000:31070/TCP	8s			
kubernetes	ClusterIP	10.100.0.1	10.100.0.1	<none>	<none>
redis-master	ClusterIP	443/TCP	10.100.32.219	8m	<none>
redis-slave	ClusterIP	6379/TCP	10.100.69.147	40s	app=redis,role=slave

comando 'kubectl get services -o wide'.

Uso de Helm con Amazon EKS Para esta parte de la práctica hay que crear, a través de la consola, un servidor y un cliente que se conecta a dicho servidor creado.

```
iMac-de-Angela:~ angelamorenos$ kubectl create namespace tiller
namespace "tiller" created
iMac-de-Angela:~ angelamorenos$ users
angelamorenos
iMac-de-Angela:~ angelamorenos$ export TILLER_NAMESPACE=tiller
iMac-de-Angela:~ angelamorenos$ tiller --listen=localhost:44134 --storage=secret --logtostderr
[main] 2019/05/26 22:06:57 Starting Tiller v2.14.0 (tls=false)
[main] 2019/05/26 22:06:57 GRPC listening on localhost:44134
[main] 2019/05/26 22:06:57 Probes listening on :44135
[main] 2019/05/26 22:06:57 Storage driver is Secret
[main] 2019/05/26 22:06:57 Max history per release is 0
```

En el terminal del servidor de tiller se establece la variable de entorno TILLER NAMESPACE y a continuación se inicia el servidor.

A continuación, en la terminal de cliente helm, establecemos la variable de entorno HELM HOST:44134 y nos conectamos al servidor.

```
iMac-de-Angela:~ angelamorenos$ export HELM_HOST=44134
iMac-de-Angela:~ angelamorenos$ helm init --client-only
SHELM_HOME has been configured at /Users/angelamorenos/.helm.
Not installing Tiller due to 'client-only' flag having been set
iMac-de-Angela:~ angelamorenos$ helm init
SHELM_HOME has been configured at /Users/angelamorenos/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes
Cluster.

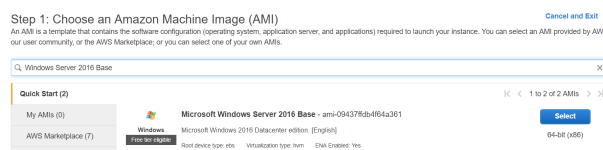
Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated
users' policy.
To prevent this, run 'helm init' with the --tiller-tls-verify flag.
For more information on securing your installation see: https://docs.helm.sh/usi
ng_helm/#securing-your-helm-installation
iMac-de-Angela:~ angelamorenos$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Skip local chart repository
...Successfully got an update from the "stable" chart repository
Update Complete.
iMac-de-Angela:~ angelamorenos$
```

Introducción a Amazon EC2

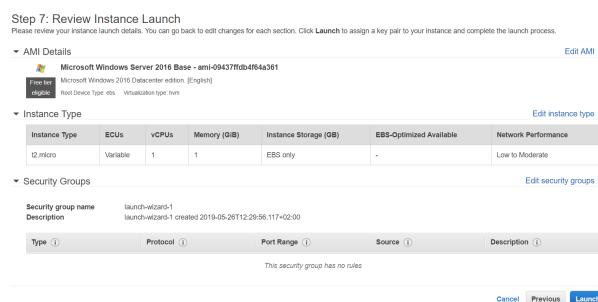
Amazon Elastic Compute Cloud (amazon EC2) proporciona capacidad de computación escalable en la nube de Amazon Web Services (AWS). El uso de Amazon EC2 elimina la necesidad de invertir en hardware.

Lanzar una instancia Para crear una instancia hay que hacer uso de la consola de Amazon EC2.

Una vez que se ha seleccionado el lanzar una instancia, hay que elegir una imagen de máquina de Amazon (AMI), que en mi caso ha sido la Windows Server 2016 Base, como se muestra a continuación. Ésta AMI está marcada como Free tier eligible.



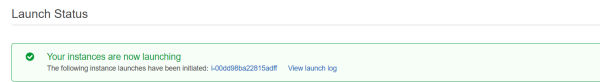
A continuación hay que elegir un tipo de instancia, quedándonos con la t2.micro que es la que viene por defecto; después de esto, revisamos y lanzamos.



Al ir a lanzar la instancia, nos hace descargarnos un par de claves, a las cuales les tienes que asignar un nombre, siendo angelamorenkey el elegido. Descarga

un archivo que se corresponde con esas llaves.

Por último, si le damos a lanzar, la instancia queda lanzada y, a partir de ese momento, podemos conectarnos a ella.

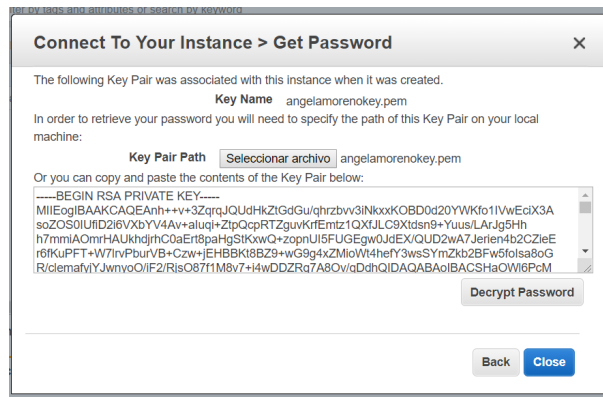


Conectarnos a la instancia Para conectarnos a la instancia tenemos que volver a hacer uso de Amazon EC2, seleccionar la instancia creada anteriormente y conectarnos a ella.

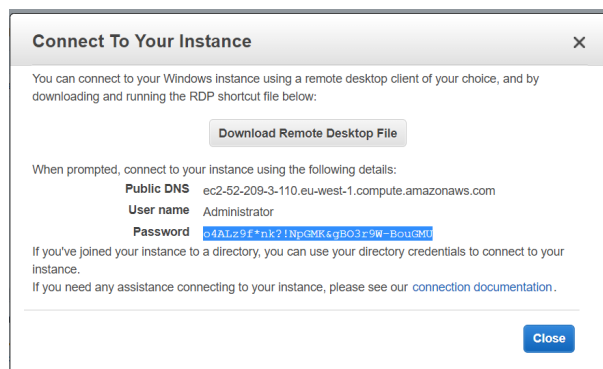
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
	i-00d598ba22815adff	t2.micro	eu-west-1b	running	2/2 checks	None	ec2-52-209-3...

Una vez que hayamos seleccionado el conectarnos a la instancia recién creada, AWS nos va a pedir que carguemos el archivo descargado previamente que contiene las dos llaves, el cual guardé bajo el nombre de angelamorenkey, que tiene

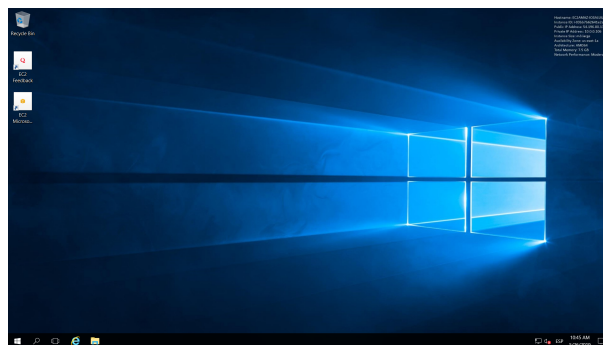
formato .pem.



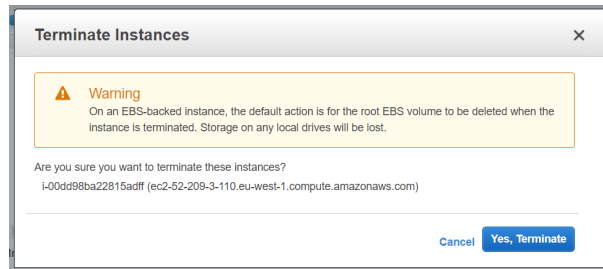
Una vez cargado y al hacer click sobre 'Decrypt Password' podemos ver la contraseña generada resultante de subir dicho archivo .pem, correspondiente al usuario 'Administrador' que creamos anteriormente.



Al hacer click sobre 'Download Remote Desktop File' se descarga un .rdp que, al ejecutarlo, va a permitir que nos conectemos a la instancia creada.



Una vez que hayamos terminado con la instancia, tal y como se indica en la guía proporcionada por AWS, hay que borrarla.



Implementar primer contenedor

En este primer apartado, en AWS te permiten crear un primer contenedor.

Container definition

Edit

Choose an image for your container below to get started quickly or define the container image to use.

sample-app

image : httpd:2.4

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

nginx

image : nginx:latest

memory : 0.5GB (512)

cpu : 0.25 vCPU (256)

tomcat-webserver

image : tomcat

memory : 2GB (2048)

cpu : 1 vCPU (1024)

custom

Image : --

memory : --

cpu : --

Configure

Task definition

Edit

También tienes que especificar el nombre del servicio y en qué puerto funcionará, entre otros.

Una vez especificados las características de este primer contenedor, AWS lo crea.

```

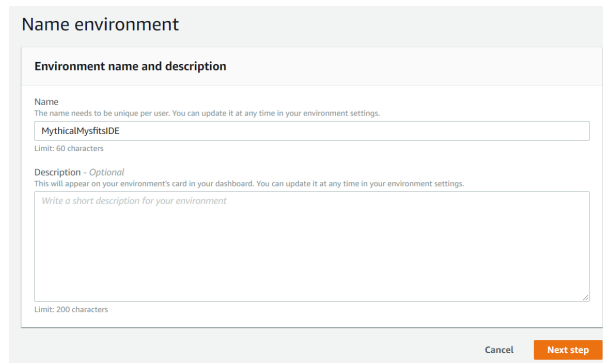
Preparing service : 9 of 9 complete
ECS resource creation ..... complete
Cluster miCluster ..... complete
Task definition first-run-task-definition:2 ..... complete
Service nginx-service ..... complete
Additional AWS service integrations ..... complete
Log group The log group [ /ecs/first-run-task-definition ] already exists ..... complete
CloudFormation stack EC2ContainerService-miCluster ..... complete
VPC vpc-0328845d9c96735e ..... complete
Subnet 1 subnet-0029f227a23a8574 ..... complete
Subnet 2 subnet-0b57c3b7c6604a168 ..... complete
Security group sg-0e6f1196d917d8be3 ..... complete
  
```

Service : nginx-service

Cluster	miCluster	Desired count	1
Status	ACTIVE	Pending count	0
Task definition	first-run-task-definition:2	Running count	1
Service type	REPLICA		
Launch type	FARGATE		
Platform version	LATEST(1.3.0)		
Service role	AWSServiceRoleForECS		

Crear una página web estática utilizando Amazon S3 y AWS Cloud 9

Después de haber seleccionado mi localización, hay que acceder al servicio "Cloud9" desde la consola AWS, que es una nube para diseñar y depurar código. Lo primero que hay que hacer, una vez hayamos accedido a la página que proporciona AWS de Cloud 9 es crear nuestro espacio de trabajo.



Name environment

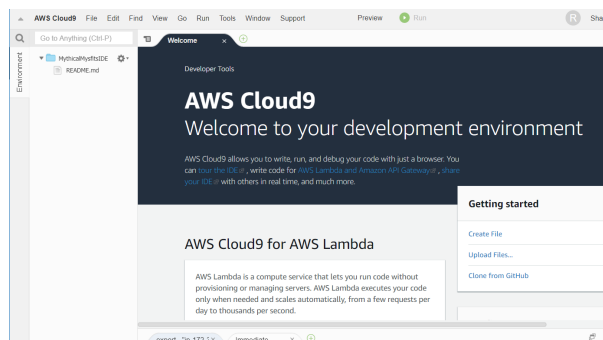
Environment name and description

Name
The name needs to be unique per user. You can update it at any time in your environment settings.
MythicalMysfitsIDE
Limit: 60 characters

Description - Optional
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.
Write a short description for your environment
Limit: 200 characters

Cancel Next step

Le he puesto "MythicalMysfitsIDE" como nombre al grupo de trabajo tal y como se muestra o indica en las instrucciones de AWS.

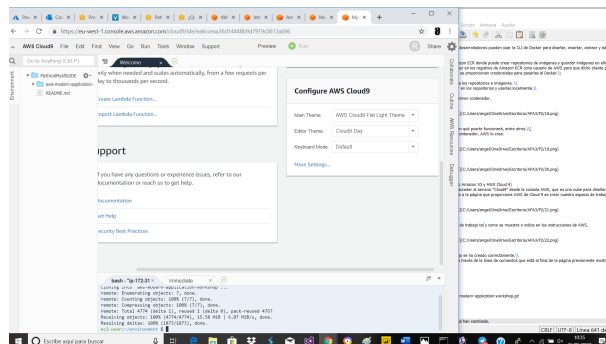


Tal y como se muestra en la imagen, el espacio de trabajo se ha creado correctamente.

Piden inicialmente que clonemos el repositorio existente a través de la línea de comandos que está al final de la página previamente mostrada. Para ello, utilizo el siguiente comando:

```
git clone -b python
https://github.com/aws-samples/aws-modern-application-workshop.git
```

Una vez clonado, podemos ver que los archivos contenidos han cambiado.



A continuación, cambiamos de directorio.

```
cd aws-modern-application-workshop
```

Siguiendo con este proceso, ahora toca crear la infraestructura donde se va a subir la página web que estamos creando.

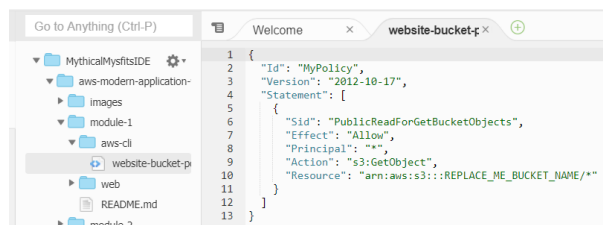
Primero se va a crear un S3 bucket, y para ello hay que darle un nombre, siendo angelamoreno el nombre del bucket que yo he creado.

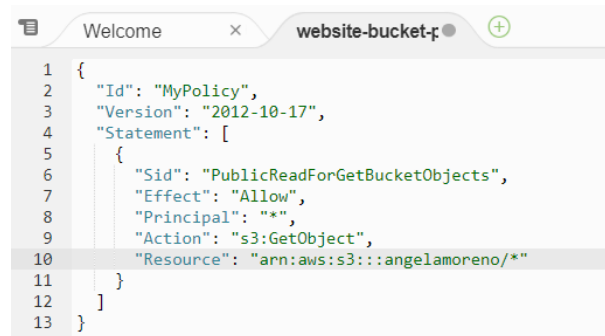
```
aws s3 mb s3://angelamoreno
```

Ahora, una vez el bucket tenga nombre, vamos a configurarlo.

```
aws s3 website s3://angelamoreno --index-document index.html
```

Por defecto, todos los buckets creados en AWS son privados, por lo que, para que puedan ser usados en una web pública, hay que cambiar la privacidad de dicho bucket, haciendo así que todos los objetos almacenados en él sean públicos para cualquiera. Esta configuración de la privacidad está localizada en un .json, el cuál tenemos que modificar. En concreto, el archivo que hay que modificar se llama "website-bucket-policy.json", al cuál se puede acceder a través de los archivos del Cloud 9 anteriormente comentados.





A continuación, la siguiente instrucción se ha ejecutado a través de la CLI de AWS para terminar de configurar la privacidad del bucket.

```
aws s3api put-bucket-policy --bucket angelamoreno
--policy file:///environment/aws-modern-application-workshop/module-1/aws-
cli/website-bucket-policy.json
```

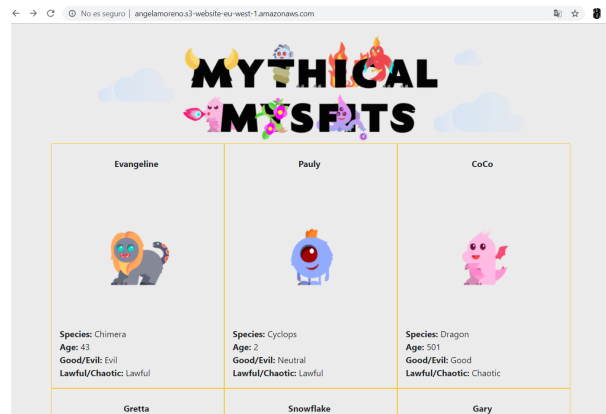
A continuación, publicamos el contenido de la web en S3.

```
aws s3 cp
/environment/aws-modern-application-workshop/module-1/web/index.html
s3://angelamoreno/index.html
```

Por último, una vez hecho esto, si vamos al navegador e introducimos la web que se detalla a continuación, tendremos acceso a la página web que hemos creado.

<http://angelamoreno.s3-website-eu-west-1.amazonaws.com/>

Siendo angelamoreno el nombre del bucket y ëu-west-1ël código correspondiente de localización de Irlanda (puesto que España no aparecía como posible localización).



Google Cloud

Google Cloud utiliza la aplicación 'Container Engine' (GKE) para administrar y organizar clústeres que ejecuta contenedores Docker. Container Engine programa los contenedores en el clúster y los administra automáticamente en función de los requisitos definidos (como la CPU y la memoria). Está desarrollado en el sistema Kubernetes de código abierto, lo que permite aprovechar la infraestructura de la nube pública, híbrida o in situ.

Crear un clúster con GKE Lo primero que hay que hacer para poder crear un clúster es crear un proyecto; esto se hace a través de la página de Kubernetes Engine.

Google Cloud Platform

Nuevo proyecto

⚠ Te quedan 3 projects en la cuota. Solicita un aumento o elimina proyectos. [Más información](#)

[MANAGE QUOTAS](#)

Nombre de proyecto *

angelamorenoproyecto

ID del proyecto: angelamorenoproyecto. No se puede cambiar más adelante.

[EDITAR](#)

Ubicación *

Ninguna organización

[EXPLORAR](#)

Carpeta u organización principal

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

A continuación hay que elegir un shell, pudiendo usar la shell local o la shell proporcionada por Google Cloud, que es la que voy a usar yo. Esta shell que viene dada por Google Cloud viene preinstalada con las herramientas de línea de comandos `gcloud` y `kubectl`; proporcionando la primera la interfaz de línea de comandos principal de GCP y la segunda la interfaz de línea de comandos para ejecutar los comandos en los clústeres de Kubernetes. Para configurar el proyecto creado anteriormente hay que ejecutar las siguientes instrucciones en la shell:

```
gcloud config set project angelamorenoproyecto
gcloud config set compute/zone europe-west2
```

Una vez que se ha configurado el proyecto, puedo proceder a crear el clúster de

GKE:

```
gcloud container clusters create angelamorenocluster
```

Como se puede ver en la imagen que viene a continuación, al ejecutar el comando anterior, da error. El error es debido a que Kubernetes Engine API no tenía permiso para acceder al proyecto. Haciendo click sobre la URL que se muestra en la imagen que viene a continuación pude habilitarlo.

```
ERROR: (gcloud.container.clusters.create) ResponseError: code=403, message=Kubernetes Engine API is not enabled for this project. Please ensure it is enabled in Google Cloud Console and try again: visit https://console.cloud.google.com/apis/api/container.googleapis.com/overview?project=angelamorenoproject to do so.
```

Al ejecutar de nuevo la instrucción para crear el clúster volvió a dar error. Sin embargo, en esta ocasión, el error se debía a que la zona o localización elegida para el proyecto (europe-west2) no tenía la cuota necesaria como para permitir crear el clúster. Por tanto, cambié la localización del proyecto y, al volver a ejecutar la instrucción no hubo ningún error.

```
ERROR: (gcloud.container.clusters.create) ResponseError: code=403, message=
(1) Insufficient regional quota to satisfy request: resource 'gcp' request requires '9.0' and is about '1.0'. project has a quota of '8.0' with '8.0' available. View and manage quotas at https://console.cloud.google.com/iam-roles/quota?usage=0&tag=project=angelamorenoproject
(2) Insufficient regional quota to satisfy request: resource 'k8s.googleapis.com' request requires '9.0' and is about '1.0'. project has a quota of '8.0' with '8.0' available. View and manage quotas at https://console.cloud.google.com/iam-roles/quota?usage=0&tag=project=angelamorenoproject.
angelamorenoprado@cloudshell: (angelamorenoproject) gcloud config set compute/zone us-west1-a
Update property [compute/zone]:
```

NAME	LOCATION	MASTER VERSION	MASTER IP	MACHINE TYPE	NODE VERSION	NUM_NODES	STATUS
angelamorenocluster	us-west1-a	1.12.7-gke.10	35.199.155.124	n1-standard-1	1.12.7-gke.10	3	RUNNING

```
angelamorenoprado@cloudshell:~ (angelamorenoproject) $
```

A continuación, para autenticarme y poder interactuar con el clúster ya creado, ejecuté el siguiente comando:

```
gcloud container clusters get-credentials angelamorenocluster
```

Una vez que se ha creado el clúster se pueden implementar aplicaciones en contenedores en él. En esta ocasión voy a implementar una aplicación dada por Google Cloud llamada hello-app.

Implementar una aplicación en el clúster Para ejecutar la aplicación hello-app en el clúster que hemos creado, ejecuto el siguiente comando:

```
kubectl run hello-server --image gcr.io/google-samples/hello-app:1.0 --port 8080
```

Dicho comando, al ejecutarse, va a retornar un 'hello-server created'. A continuación, para exponerlo en Internet y que los usuarios puedan acceder a ella, se ejecuta el siguiente comando:

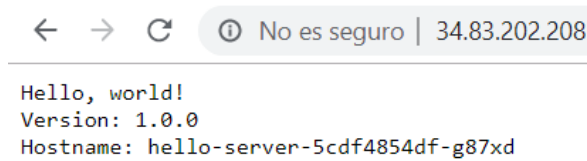
```
kubectl expose deployment hello-server --type LoadBalancer
--port 80 --target-port 8080
```

Inspeccionar y visualizar la aplicación Para inspeccionar el servicio hello-server y que así nos de la dirección IP externa del servicio hay que ejecutar el siguiente comando:

```
kubectl get service hello-server
```

```
angelamorenoprado@cloudshell:~ (angelamorenoproyecto)$ kubectl get service hello-server
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
hello-server   LoadBalancer 10.11.246.223  34.83.202.208  80:30601/TCP 89s
angelamorenoprado@cloudshell:~ (angelamorenoproyecto)$
```

Si copiamos la dirección IP externa en el navegador, nos aparecerá la aplicación.



The screenshot shows a web browser address bar with the URL `34.83.202.208`. Below the address bar, the page content displays:

```
Hello, world!
Version: 1.0.0
Hostname: hello-server-5cdf4854df-g87xd
```

Limpieza Para borrar tanto el servicio de la aplicación como el clúster creados hay que ejecutar los siguientes comandos:

```
angelamorenoprado@cloudshell:~ (angelamorenoproyecto)$ kubectl delete service hello-server
service "hello-server" deleted
angelamorenoprado@cloudshell:~ (angelamorenoproyecto)$ gcloud container clusters delete angelamorenocluster
The following clusters will be deleted.
- [angelamorenocluster] in [us-west1-a]
Do you want to continue (Y/n)? Y
Deleting cluster angelamorenocluster...done.
Deleted [https://container.googleapis.com/v1/projects/angelamorenoproyecto/zones/us-west1-a/clusters/angelamorenocluster].
angelamorenoprado@cloudshell:~ (angelamorenoproyecto)$
```

Conclusiones

Azure Con respecto a contenedores, Azure tiene cinco aplicaciones para poner trabajar con ellos.

1. Kubernetes Service o AKS: Es un servicio ofrecido por Azure que simplifica la implementación de clústeres. Como característica destacada encontramos que al implementar un clúster de AKS, el maestro y todos los nodos de Kubernetes se implementan y configuran automáticamente.

Pero, además de eso, AKS ofrece administración de identidades y seguridad a través de RBAC, que implementan las definiciones de 'rol' a un usuario o grupo para un ámbito determinado; también ofrece supervisión y registro integrados. AKS también ofrece escalado de pods y nodos de clúster: a medida que cambia la demanda de recursos, el número de pods o nodos de clúster que ejecutan sus servicios pueden escalarse vertical u horizontalmente; actualizaciones de nodos de clúster y nodos habilitados para GPU.

También es compatible con imágenes de Docker.

2. Service Fabric: Es una plataforma de sistemas distribuidos para implementar y administrar microservicios y contenedores escalables.

3. Azure Web App for Containers: Proporciona pilas de aplicaciones predefinidas en Linux con compatibilidad con lenguajes como .NET, PHP y Node.js entre otros. También puede usar una imagen personalizada de Docker para ejecutar la aplicación web.

4. Azure Container Registry: Servicio de registro de contenedores de Docker usado para almacenar imágenes de contenedor de docker privadas.

5. Azure Container Instances: Ejecutar contenedores de Docker sin servidor en Azure.

AWS Con respecto a contenedores, AWS tiene cinco servicios para trabajar con ellos:

1. Amazon ECR: Servicio de organización de contenedores para almacenar sus imágenes de contenedores. Facilita las tareas de almacenamiento, administración e implementación.

2. Amazon ECS: Dividir aplicaciones monolíticas en microservicios, migrar a la nube o ejecutar cargas de trabajo de procesamiento por lotes. También aplicaciones de aprendizaje automático.

3. Amazon EKS: Ejecutar Kubernetes en AWS, crear aplicaciones híbridas o implementar modelos de aprendizaje automático.

4. AWS Fargate: Contenedores sin servidor o crear una PaaS.

5. Amazon EC2: Obtener el máximo control sobre su tipo de lanzamiento.

Google Cloud Google Cloud tiene Container Engine como sistema de administración y organización de clústeres que ejecuta contenedores Docker.

Container Engine es compatible con Docker, tiene registro privado de contenedores para almacenar imágenes Docker privadas y acceder a ellas fácilmente, es escalable, tiene logging y monitoring, permite redes híbridas y administración de

identidades y acceso.

Como conclusión, de Azure destacaría la gran cantidad de tutoriales que tienen disponibles, pudiendo haber creado un clúster, una aplicación en un contenedor, crear una aplicación web, crear un grupo de recursos, crear un registro de Docker privado y añadirle una imagen.

De AWS destacaría lo intuitivo que es, además de un punto muy importante en temas de seguridad que es la generación de las dos claves, además de la generación de un ID y contraseña secretos cuando intentas acceder de forma remota. También destaco la cantidad de tutoriales y de ejemplos que hay en la web, la documentación, además del procesamiento en lotes y las aplicaciones de aprendizaje automático. Además, también destacaría AWS Cloud9, que permite escribir, ejecutar y debuggear código en el propio buscador. Como punto negativo, pondría que para poder acceder a tutoriales no sirve con la cuenta AWS student. Con respecto a Google Cloud, destaco el tener una única aplicación para trabajar con los contenedores, además de permitir redes híbridas, logging y monitoring. Sin embargo, ampliaría la documentación que ofrecen, al igual que tutoriales. Por todo lo anteriormente comentado e implementado, mi elección sería AWS.

Problemas encontrados

Durante el desarrollo de esta práctica he encontrado mayoritariamente un problema, y es que me ha sido imposible instalar Docker en el ordenador Windows que tengo, debido a que, como se ha comentado anteriormente, Docker pide tener como sistema operativo Windows 10 Pro, el cual yo no tengo.

Por tanto, para solucionar ese problema, hay apartados de la práctica que han sido resueltos con el sistema operativo Windows y otros (cuando daba algún tipo de error en Windows) con el sistema operativo macOS Sierra.

```
C:\Users\angel>docker login
Warning: failed to get default registry endpoint from daemon (error during connect: Get http://K2F3ZF.X2Fp3eK2F/docker_engine/v1.37/info: open //./pipe/docker_engine: The system cannot find th
file specified. In the default daemon configuration on Windows, the docker client must be run elevated to connect. This error may also indicate that the docker daemon is not running.). Using
system default: https://index.docker.io/v1/
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: westlandz
Password:
Error during connect: Post http://K2F3ZF.X2Fp3eK2F/docker_engine/v1.37/auth: open //./pipe/docker_engine: The system cannot find the file specified. In the default daemon configuration on Wind
ows, the docker client must be run elevated to connect. This error may also indicate that the docker daemon is not running.
```

Bibliografía containers

1. Shell de Azure: <https://shell.azure.com/>
2. Azure for Containers: <https://docs.microsoft.com/es-es/azure/containers/>
3. Instalación de la CLI de Azure en macOS: <https://docs.microsoft.com/es-es/cli/azure/install-azure-cli-macos?view=azure-cli-latest>
4. Introducción a los contenedores (AWS): <https://aws.amazon.com/es/containers/getting-started/>
5. Container Engine - Google Cloud: <https://cloud.google.com/container-engine/?hl=es>