



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Despliegue de un cluster Kubernetes altamente disponible
en IBM Cloud

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Jovaní Beltrán, Óscar

Tutor/a: Acebrón Linuesa, Floreal

CURSO ACADÉMICO: 2022/2023



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Oscar Jovaní Beltrán

Tutor: Floreal Acebrón Linuesa

Curso Académico 2022/2023

Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

Resumen

El objetivo principal es el despliegue de un cluster Kubernetes altamente disponible en IBM Cloud, explorando los diferentes aspectos involucrados en su configuración y gestión. Se busca proporcionar una infraestructura sólida y confiable para el despliegue de aplicaciones en contenedores, utilizando las mejores prácticas y metodologías actuales. A lo largo del proyecto, se analizarán aspectos clave como la configuración de nodos, el almacenamiento persistente, la gestión de redes y el despliegue de servicios, con el fin de brindar una visión completa y práctica, con el objetivo final de que cualquier persona pueda seguir los mismos pasos para aprender de Kubernetes y desplegar su propio cluster.

Palabras clave: Kubernetes, kubeadm, IBM Cloud, cluster, HPA, Load balancer.

Abstract

The main objective is the deployment of a highly available Kubernetes cluster on IBM Cloud, exploring the different aspects involved in its configuration and management. The aim is to provide a solid and reliable infrastructure for the deployment of containerized applications, using current best practices and methodologies. Throughout the project, key aspects such as node configuration, persistent storage, network management and service deployment will be discussed, in order to provide a complete and practical overview, with the ultimate goal that anyone can follow the same steps to learn about Kubernetes and deploy their own cluster.

Keywords : Kubernetes, kubeadm, IBM Cloud, cluster, HPA, Load balancer.



Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

Tabla de contenidos

1. Introducción.....	9
1.1. Objetivos	9
1.2. Metodología.....	9
2. Estado del arte	10
3. Análisis del problema	11
3.1. Introducción	11
3.2. Problemática	11
3.3. Justificación del Estudio.....	12
3.4. Objetivos del Estudio	12
3.5. Metodología	12
3.6. Limitaciones.....	13
4. Diseño de la solución	14
4.1. Desarrollo Local con Kubeadm	14
4.2. Implementación en IBM Cloud	14
4.3. Arquitectura del Sistema	15
4.4. Tecnología utilizada.....	17
5. Desarrollo de la solución en local	18
5.1. Despliegue con Minikube	18
5.1.1. Instalación de Minikube en Windows.....	18
5.1.2. Despliegue del cluster	19
5.1.3. Complementos del cluster	19
5.1.4. Despliegue de un servicio	20
5.2. Despliegue con Kubeadm.....	21
5.2.1. Preparación del entorno virtual.....	22
5.2.2. Configuración de las máquinas virtuales	23
5.3. Almacenamiento persistente (NFS SERVER)	29
5.3.1. Preparación de la máquina virtual	29
5.3.2. Persistent volumes y Persistent volumes claims.....	30
5.3.3. Test del almacenamiento persistente.....	31
5.4. Load balancer (MetalLB)	32
5.4.1. Implementación de MetalLB	33
5.4.2. Tests del balanceador	34



5.5. Horizontal Pod Autoscaling (HPA).....	35
5.5.1. Implementación del autoescalador.....	35
5.5.2. Tests del autoescalador	35
6. IBM Cloud.....	37
6.1. Registro y primeras impresiones	38
6.2. IBM Cloud shell	41
6.3. IBM Kubernetes Service (IKS).....	43
6.3.1. Free tier cluster.....	43
6.3.2. Standard tier cluster.....	44
6.4. Red Hat Openshift	46
6.5. Container registry	47
7. Despliegue del cluster en IBM Cloud.....	48
7.1. Despliegue de un servicio con kube dashboard.....	50
7.2. Load balancer en IBM Cloud	53
7.3. Persistent Storage en IBM Cloud	54
7.3.1. Diferencias entre storages	54
7.3.2. Tests con File Storage.....	55
7.4. Autoscalers en IBM Cloud	57
7.4.1. HPA en IBM Cloud	57
7.4.1. Cluster autoscaler	58
7.5. Ingress en IBM Cloud.....	60
7.5.1. Tests con Ingress.....	61
8. Observability en IBM Cloud	62
8.1. Logging integration	62
8.2. Monitoring integration	64
9. High availability	66
10. Conclusiones	69
11. Bibliografía	70
12. ANEXO.....	71

Tabla de figuras

Figura 1. Arquitectura de Kubernetes	16
Figura 2. Directorio minikube en Windows	18
Figura 3. Addons en Minikube	20
Figura 4. Servicio en Minikube	21
Figura 5. Clonado de la máquina virtual	26
Figura 6. Configuración de red master.....	27
Figura 7. Configuración de red worker 1	27
Figura 8. Configuración de red worker 2	27
Figura 9. kubectl get nodes en local	29
Figura 10. index.php en nfserver	31
Figura 11. HTML desde el navegador	31
Figura 12. Maqueta del Load Balancer	32
Figura 13. Pods activos de MetalLB.....	33
Figura 14. Servicio nginx con ip externa	34
Figura 15. Load Balancer en acción	34
Figura 16. Load generator en acción	36
Figura 17. HPA.....	36
Figura 18. IBM Cloud registro	38
Figura 19. IBM Cloud dashboard	39
Figura 20. IBM Cloud 200\$ credito	39
Figura 21. Gasto mensual en IBM Cloud.....	40
Figura 22. IBM Cloud shell web	41
Figura 23. IBM Cloud shell on premise.....	42
Figura 24. Summary free cluster	43
Figura 25. Dashboard free cluster	44
Figura 26. Selección tipo cluster IBM Cloud.....	45
Figura 27. Container registry en IBM Cloud	47
Figura 28. kubectl get nodes en IBM Cloud	48
Figura 29. Coste del cluster	49
Figura 30. Kubernetes dashboard button	50
Figura 31. Create en kube dashboard	51
Figura 32. Load Balancer kubectl get svc.....	51
Figura 33. Página web nginx IBM	52
Figura 34. Load balancer IBM Cloud	53
Figura 35. PVC en IBM Cloud	55
Figura 36. Deployment para PVC en IBM Cloud	55
Figura 37. Servicio para PVC en IBM Cloud	56
Figura 38. Test File Storage.....	56
Figura 39. Prueba HPA IBM Cloud	57
Figura 40. Escalado de pods en IBM Cloud	58
Figura 41. Cluster autoscaler addon	59
Figura 42. Configmap para cluster autoscaler	59
Figura 43. Esquema de ingress.....	60
Figura 44. kubectl get ing	61



Figura 45. Hello app 1 HTML.....	61
Figura 46. Hello app 2 HTML	61
Figura 47. Integrations en el cluster.....	62
Figura 48. Logging integration.....	63
Figura 49. Monitoring integration	65
Figura 50. Selección de Multiple location en cluster	67
Figura 51. Summary multi zone cluster.....	68

1. Introducción

El despliegue de aplicaciones en entornos de nube ha ganado una enorme relevancia en los últimos años debido a su capacidad para brindar flexibilidad, escalabilidad y confiabilidad a las organizaciones. Kubernetes, como una de las plataformas de orquestación de contenedores más populares, ha demostrado ser una herramienta fundamental para gestionar y desplegar aplicaciones de forma eficiente en entornos de nube.

1.1. Objetivos

El objetivo principal de este proyecto es explorar los diferentes aspectos involucrados en el despliegue de un cluster Kubernetes altamente disponible en IBM Cloud, incluyendo la configuración de nodos, la configuración de almacenamiento persistente, la gestión de redes y la alta disponibilidad de los componentes del clúster.

Básicamente se busca hacer un primer despliegue de Kubernetes utilizando kubeadm en máquinas virtuales locales para familiarizarse un poco con el sistema, aprender a usar los principales comandos y desplegar nuestro primer microservicio.

Para más tarde seguir con el despliegue de un cluster altamente disponible en IBM Cloud, que es una plataforma de nube que ofrece una amplia gama de servicios y soluciones, incluyendo soporte para Kubernetes y así estudiar todo lo que nos puede ofrecer esta plataforma. Al implementar este cluster, se busca proporcionar una infraestructura sólida y confiable para el despliegue y ejecución de aplicaciones en contenedores.

1.2. Metodología

El trabajo va a consistir principalmente en dos partes, en la primera se tratará de desplegar un entorno virtualizado en el que vamos a desplegar varias máquinas para instalar Kubernetes de forma manual y probar a desplegar ciertos servicios para familiarizarnos con Kubernetes.

En la segunda parte ya nos dirigiremos a IBM Cloud para ver qué herramientas y servicios nos brinda a la hora de querer montar nuestro cluster Kubernetes y poder comparar precios y facilidades que nos ofrece a la hora de abordar el proyecto.



2. Estado del arte

Kubernetes fue desarrollado originalmente por Google y se basó en sus experiencias internas con la administración de contenedores a escala. La compañía había estado utilizando contenedores durante muchos años como una forma eficiente de empaquetar y desplegar aplicaciones.

Antes de la aparición de Kubernetes, el despliegue y la administración de aplicaciones en contenedores eran tareas desafiantes y complejas. No había una solución estándar y consolidada para orquestar y gestionar contenedores a gran escala. Los equipos de operaciones tenían que realizar tareas manuales y personalizadas para administrar y escalar los contenedores, lo que resultaba en una carga de trabajo significativa y falta de eficiencia.

En 2014, Google lanzó públicamente Kubernetes como un proyecto de código abierto, bajo la Fundación de Software de Nube de la Fundación Linux (CNCF). Al abrirlo al público, Google permitió que la comunidad contribuyera y aprovechara las mismas técnicas y prácticas que Google había desarrollado para administrar sus propias aplicaciones basadas en contenedores.

Kubernetes proporcionó una solución elegante y poderosa para la orquestación de contenedores. Introdujo conceptos clave, como los pods (grupos de contenedores que comparten recursos), los servicios (abstracción de red para acceder a los contenedores) y los controladores de replicación (gestión de la escala y la disponibilidad de los contenedores). Kubernetes también ofreció una interfaz declarativa para describir el estado deseado de las aplicaciones y permitió que el sistema se encargara de llevar a cabo la orquestación y el mantenimiento.

Desde su lanzamiento, Kubernetes ha ganado una gran popularidad y se ha convertido en una de las principales plataformas de orquestación de contenedores. Ha fomentado un ecosistema vibrante con una amplia gama de herramientas, servicios y proveedores que admiten y se integran con Kubernetes.

En resumen, antes de la aparición de Kubernetes, no existía una solución estandarizada y de código abierto para la administración y orquestación de contenedores a gran escala. Kubernetes ha llenado ese vacío y ha revolucionado la forma en que se despliegan y gestionan las aplicaciones en contenedores, ofreciendo escalabilidad, disponibilidad y eficiencia operativa.

3. Análisis del problema

3.1. Introducción

El despliegue de aplicaciones en entornos de computación en la nube ha experimentado un crecimiento significativo en los últimos años. Sin embargo, garantizar la alta disponibilidad de estas aplicaciones sigue siendo un desafío para muchas organizaciones. En este contexto, el uso de Kubernetes como plataforma de orquestación de contenedores se ha vuelto cada vez más popular debido a sus capacidades para gestionar aplicaciones de manera escalable y confiable.

3.2. Problemática

El despliegue de un clúster Kubernetes altamente disponible implica superar una serie de desafíos técnicos y operativos. Algunos de los problemas clave que pueden surgir en este proceso incluyen:

Configuración del clúster: La configuración inicial de un clúster Kubernetes requiere atención especial para garantizar la alta disponibilidad. Esto implica abordar aspectos como la elección adecuada del número de nodos maestros y trabajadores, la configuración de almacenamiento persistente, el enrutamiento de red y la seguridad del clúster.

Escalabilidad: A medida que una aplicación crece y experimenta un mayor número de usuarios y carga de trabajo, el clúster Kubernetes debe ser capaz de escalar de manera eficiente y automática. Esto implica definir políticas de escalado automático basadas en métricas de rendimiento, como la utilización de CPU o la latencia de las aplicaciones.

Gestión de la carga de trabajo: El despliegue de múltiples aplicaciones y servicios en un clúster Kubernetes requiere una gestión adecuada de la carga de trabajo. Esto incluye la planificación de pods, el equilibrio de carga entre los nodos y la asignación de recursos adecuados a cada aplicación para evitar cuellos de botella y garantizar un rendimiento óptimo.

Resiliencia y recuperación de errores: Los fallos en los nodos o en los servicios de un clúster Kubernetes pueden afectar la disponibilidad de las aplicaciones. Por lo tanto, es fundamental implementar estrategias de resiliencia, como la replicación de pods y la detección automática de fallos para garantizar una rápida recuperación de errores y minimizar el impacto en los usuarios finales.



3.3. Justificación del Estudio

La alta disponibilidad es un requisito crítico para muchas aplicaciones empresariales y servicios en la nube. El despliegue de un clúster Kubernetes altamente disponible en IBM Cloud proporciona una solución potencial para abordar estos desafíos y garantizar la disponibilidad y confiabilidad de las aplicaciones en un entorno de producción. Al investigar y analizar estos problemas, se podrán identificar las mejores prácticas y estrategias para implementar y mantener un clúster Kubernetes altamente disponible en IBM Cloud.

3.4. Objetivos del Estudio

El objetivo principal de este trabajo es desplegar y configurar un clúster Kubernetes altamente disponible en IBM Cloud, teniendo en cuenta los desafíos mencionados anteriormente. Los objetivos específicos incluyen:

Investigar y analizar las características y funcionalidades de alta disponibilidad ofrecidas por Kubernetes en IBM Cloud.

Diseñar una arquitectura adecuada para un clúster, considerando los requisitos de disponibilidad y escalabilidad.

Implementar y configurar el clúster utilizando las mejores prácticas y herramientas recomendadas.

Realizar pruebas y evaluaciones para verificar la alta disponibilidad del clúster y la capacidad de escalar con la carga de trabajo.

Documentar las lecciones aprendidas, las limitaciones y las recomendaciones para realizar un despliegue exitoso.

3.5. Metodología

El estudio se llevará a cabo utilizando una metodología basada en la investigación, experimentación y análisis de los componentes y características necesarios para el despliegue de un clúster Kubernetes altamente disponible en IBM Cloud. Se realizarán pruebas y evaluaciones para validar la configuración del clúster y se recopilarán datos para analizar su rendimiento y disponibilidad.

Se utilizarán recursos y servicios disponibles en IBM Cloud, como IBM Kubernetes Service (IKS), para implementar y configurar el clúster Kubernetes. También se investigarán las mejores prácticas y recomendaciones de IBM y la comunidad de Kubernetes para abordar los desafíos mencionados anteriormente.

3.6. Limitaciones

Algunas limitaciones que podrían surgir durante el desarrollo de este estudio incluyen:

Restricciones de tiempo y recursos: Debido a la naturaleza de un TFG, el tiempo y los recursos disponibles pueden ser limitados para realizar una investigación exhaustiva y pruebas a gran escala. Se buscará maximizar la calidad y la relevancia de las pruebas y análisis dentro de las limitaciones existentes.

Cambios en las versiones y características de Kubernetes: Dado que Kubernetes es una tecnología en constante evolución, algunas características y funcionalidades pueden cambiar o actualizarse durante el desarrollo del estudio. Se deberá tener en cuenta cualquier cambio relevante y adaptar la configuración y las pruebas en consecuencia.

Complejidad de la configuración y la arquitectura: El despliegue de un clúster Kubernetes puede ser complejo y requerir un conocimiento profundo de los componentes y la arquitectura de Kubernetes. Se hará un esfuerzo por simplificar y documentar el proceso, pero algunas configuraciones avanzadas pueden quedar fuera del alcance de este estudio.

A pesar de estas limitaciones, se espera que el estudio proporcione una base sólida para comprender y abordar los desafíos del despliegue de un clúster Kubernetes altamente disponible en IBM Cloud.



4. Diseño de la solución

En este apartado, se presenta el diseño de la solución que, se dividirá en dos secciones: una parte enfocada en el desarrollo local utilizando Kubeadm y otra parte centrada en la implementación en IBM Cloud.

4.1. Desarrollo Local con Kubeadm

El desarrollo local utilizando Kubeadm permitirá crear y probar un clúster Kubernetes en un entorno controlado para así familiarizarse con el sistema antes de implementarlo en IBM Cloud.

Durante este despliegue aprenderemos a montar las máquinas virtuales junto a su configuración y puesta en marcha de todos los componentes para hacer funcionar Kubernetes.

También desplegaremos diversos servicios que nos ayudarán a dar redundancia y disponibilidad a nuestras aplicaciones.

4.2. Implementación en IBM Cloud

La implementación en IBM Cloud se centrará en utilizar los servicios y recursos proporcionados por la plataforma para desplegar un clúster Kubernetes altamente disponible en un entorno de producción.

Se utilizará el servicio administrado de IBM Cloud, IBM Kubernetes Service (IKS), para desplegar el clúster Kubernetes altamente disponible. IKS ofrece capacidades integradas de alta disponibilidad y escalabilidad, lo que facilita la administración del clúster y reduce la carga operativa.

Se distribuirán los nodos del clúster Kubernetes en diferentes zonas de disponibilidad dentro de IBM Cloud para garantizar la redundancia y la resistencia a fallos. Esto permitirá que la aplicación siga estando disponible incluso en caso de fallos en una zona específica lo que nos va a proporcionar la alta disponibilidad que buscamos.

Se utilizarán diferentes servicios proporcionados por IBM Cloud como el almacenamiento persistente, el balanceo de carga y el HPA, para hacer nuestro cluster lo más profesional posible.

4.3. Arquitectura del Sistema

La arquitectura del sistema puede seguir un enfoque de arquitectura en capas, que proporciona escalabilidad, redundancia y capacidad de recuperación frente a fallos. A continuación, se presenta una descripción general de la arquitectura del sistema:

Capa de Infraestructura: Esta capa proporciona la infraestructura en la nube necesaria para alojar el clúster Kubernetes en IBM Cloud ofreciendo recursos computacionales, almacenamiento y servicios de red que permiten la creación y administración de los nodos del clúster y también representa el entorno virtual que vamos a crear a la hora de hacer el diseño local con kubeadm.

Capa de Control del Clúster Kubernetes: En esta capa se encuentran los componentes de control del clúster Kubernetes, incluidos los nodos maestros. Los nodos maestros son responsables de la gestión del clúster, como el control de estado, la programación de tareas y la gestión del almacenamiento. Esta capa incluye componentes como etcd, kube-apiserver, kube-controller-manager y kube-scheduler, que son los componentes internos que hacen que funcione Kubernetes.

Capa de Nodos Trabajadores: Esta capa está compuesta por los nodos trabajadores que ejecutan las aplicaciones y cargas de trabajo desplegadas en el clúster. Los nodos trabajadores ejecutan los contenedores y proporcionan los recursos computacionales necesarios para las aplicaciones. Pueden escalar horizontalmente según la demanda de carga de trabajo.

Capa de Red y Comunicación: Esta capa se encarga de la comunicación entre los diferentes componentes del clúster Kubernetes y las aplicaciones desplegadas en él. Utiliza servicios de red definidos por software (SDN) para facilitar la conectividad y el enrutamiento entre los nodos del clúster.

Capa de Servicios Adicionales: Esta capa comprende los servicios adicionales que se pueden utilizar junto con el clúster Kubernetes para mejorar la funcionalidad y la gestión. Pueden incluir servicios de registro y monitorización, servicios de seguridad, servicios de almacenamiento en la nube, entre otros.

Capa de Aplicaciones: En esta capa se despliegan las aplicaciones y cargas de trabajo dentro del clúster Kubernetes. Las aplicaciones se empaquetan en contenedores y se administran mediante Kubernetes para aprovechar las características de escalabilidad, gestión de recursos y alta disponibilidad proporcionadas por el clúster.



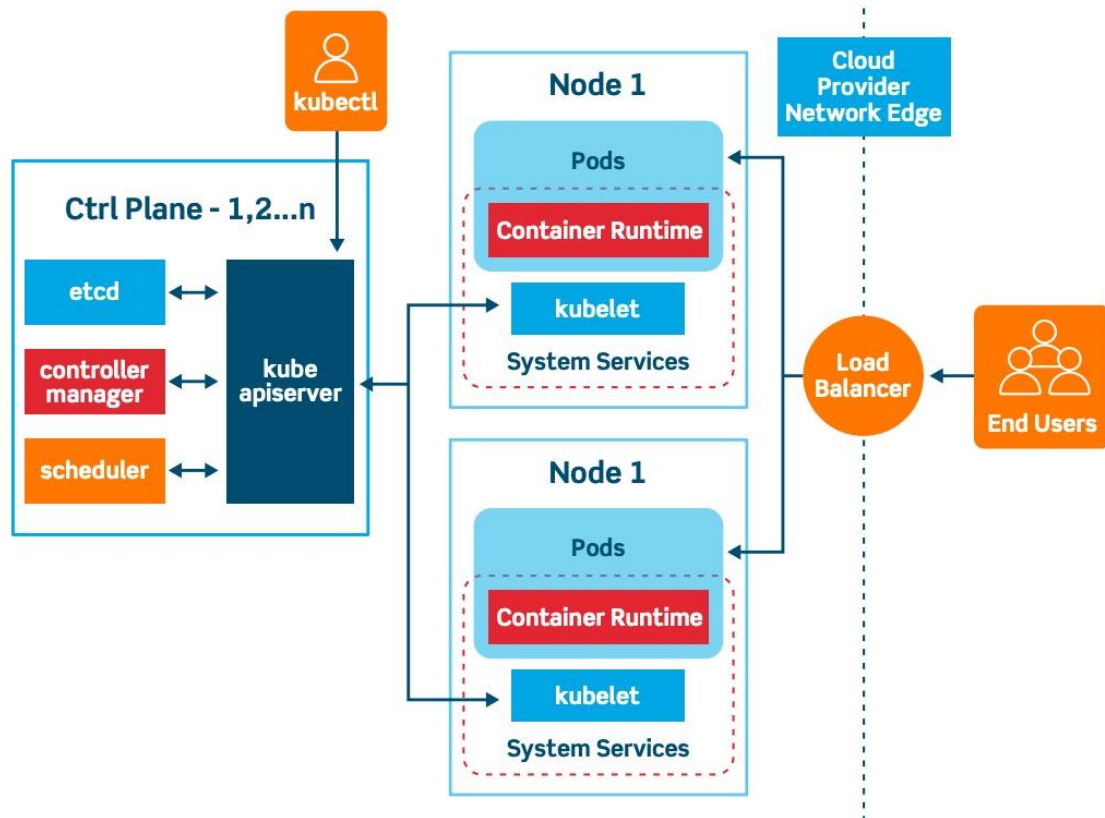


Figura 1. Arquitectura de Kubernetes

La arquitectura del sistema está diseñada para garantizar la alta disponibilidad, escalabilidad y resistencia a fallos del clúster Kubernetes. La distribución de los nodos en diferentes zonas de disponibilidad, el uso de servicios de balanceo de carga y la configuración de mecanismos de replicación y recuperación ante fallos contribuyen a lograr estos objetivos.

4.4. Tecnología utilizada

El despliegue de un clúster Kubernetes altamente disponible implica el uso de varias tecnologías y herramientas.

Kubernetes: Kubernetes es la tecnología principal utilizada para orquestar y gestionar el clúster. Proporciona una plataforma robusta y escalable para el despliegue y la administración de contenedores.

Kubeadm: Kubeadm es una herramienta oficial de Kubernetes utilizada para facilitar la creación y la configuración del clúster. Permite la inicialización y el mantenimiento de los componentes del clúster, como los nodos maestros y trabajadores.

IBM Cloud: La plataforma en la nube utilizada para el despliegue del clúster Kubernetes altamente disponible. Proporciona una infraestructura escalable y confiable, así como servicios adicionales que se pueden integrar con el clúster, como monitoreo, seguridad y almacenamiento.

Redes Definidas por Software: Las soluciones SDN, como Calico, Flannel o Weave, se utilizan para proporcionar la conectividad y el enrutamiento de red necesarios dentro del clúster Kubernetes. Estas soluciones permiten establecer comunicación entre los nodos maestros y trabajadores, así como entre las aplicaciones desplegadas.

Servicios de Balanceo de Carga: Se pueden utilizar servicios de balanceo de carga externos, como MetalLB, para distribuir el tráfico de red de manera equitativa entre los nodos del clúster. Estos servicios garantizan una distribución eficiente de las solicitudes y evitan la sobrecarga en un solo nodo.

Sistemas de Almacenamiento: Se pueden emplear diferentes sistemas de almacenamiento para garantizar la persistencia de los datos en el clúster Kubernetes. Esto puede incluir volúmenes en el host, sistemas de archivos distribuidos o servicios de almacenamiento en la nube.

Estas tecnologías son fundamentales para el diseño y la implementación de un clúster Kubernetes altamente disponible. Su uso conjunto permite la creación de un entorno robusto, escalable y confiable para la ejecución de aplicaciones en contenedores.



5. Desarrollo de la solución en local

Vamos a comenzar con el desarrollo de un cluster kubernetes utilizando Minikube ya que es una herramienta muy sencilla de utilizar que nos ayudará a familiarizarnos con los comandos de kubernetes y aprender a manejar bien el cluster antes de hacer la implementación en IBM Cloud.

5.1. Despliegue con Minikube

He decidido añadir este apartado por la facilidad y rapidez que ofrece minikube a la hora de montar un cluster en kubernetes y hacer pruebas con él.

5.1.1. Instalación de Minikube en Windows

Para instalar Minikube, es tan facil como abrir una terminal con permisos de administrador y ejecutar: **winget install minikube**.

Después de ello se nos habrá instalado Minikube y solo tendremos que cerrar la terminal y volverla a abrir para empezar a ejecutar comandos. (Si despues de esto, no nos funciona aun, solo hay que dirigirse a la carpeta de Minikube mediante el comando cd de windows y luego ya podremos usar los comandos de minikube).

C:\Program Files\Kubernetes\Minikube

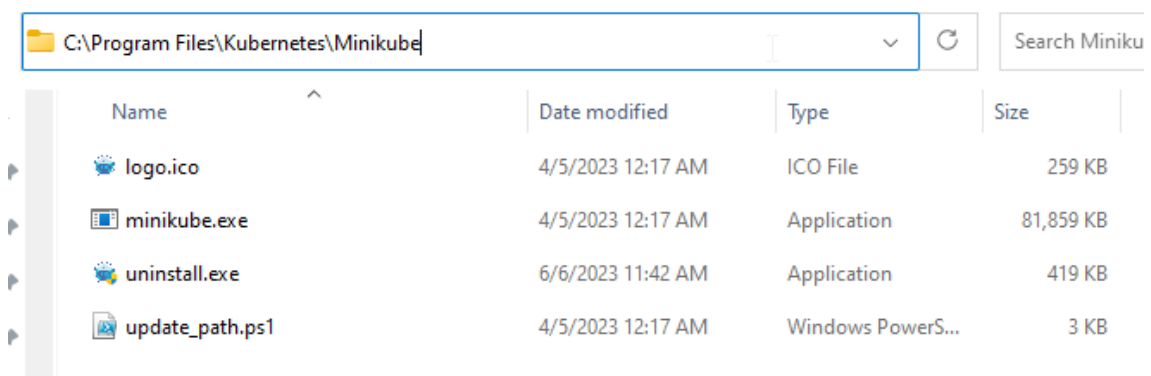


Figura 2. Directorio minikube en Windows

5.1.2. Despliegue del cluster

Minikube nos permite tener un cluster kubernetes totalmente operativo en menos de 5 minutos con una simple sentencia. Para ello solo debe ejecutar un comando: `minikube start`. Esta sentencia nos crea un cluster con un master y un worker. Pero si queremos algo mas concreto podemos añadir parametros a este comando. Por ejemplo:

```
minikube start --driver=virtualbox -n 3 --container-runtime=containerd  
--cni=calico
```

Con estos parametros le estamos diciendo que nos cree 3 máquinas virtuales en virtualbox(2 workers y 1 master) usando containerd y utilizando calico como red de contenedores.

5.1.3. Complementos del cluster

En el contexto de Kubernetes y Minikube, existen los addons que son componentes adicionales que se pueden habilitar en un clúster de Minikube para extender su funcionalidad y facilitar tareas comunes. Estos addons proporcionan características y herramientas útiles para el desarrollo, la depuración y la administración de aplicaciones.

Los addons se pueden habilitar usando el comando `minikube addons enable <nombre_del_addon>`.

En la siguiente imagen podemos observar un listado de todos los addons que nos ofrece minikube.



```
PS C:\Windows\system32> minikube addons list
```

ADDON NAME	PROFILE	STATUS	MAINTAINER
ambassador	minikube	disabled	3rd party (Ambassador)
auto-pause	minikube	disabled	Google
cloud-spanner	minikube	disabled	Google
csi-hostpath-driver	minikube	disabled	Kubernetes
dashboard	minikube	disabled	Kubernetes
default-storageclass	minikube	disabled	Kubernetes
efk	minikube	disabled	3rd party (Elastic)
freshpod	minikube	disabled	Google
gcp-auth	minikube	disabled	Google
gvisor	minikube	disabled	Google
headlamp	minikube	disabled	3rd party (kinvolk.io)
helm-tiller	minikube	disabled	3rd party (Helm)
inacel	minikube	disabled	3rd party (InAccel [info@inacel.com])
ingress	minikube	disabled	Kubernetes
ingress-dns	minikube	disabled	Google
istio	minikube	disabled	3rd party (Istio)
istio-provisioner	minikube	disabled	3rd party (Istio)
kong	minikube	disabled	3rd party (Kong HQ)
kubevirt	minikube	disabled	3rd party (KubeVirt)
logviewer	minikube	disabled	3rd party (unknown)
metallb	minikube	disabled	3rd party (MetalLB)
metrics-server	minikube	disabled	Kubernetes
nvidia-driver-installer	minikube	disabled	Google
nvidia-gpu-device-plugin	minikube	disabled	3rd party (Nvidia)
olm	minikube	disabled	3rd party (Operator Framework)
pod-security-policy	minikube	disabled	3rd party (unknown)
portainer	minikube	disabled	3rd party (Portainer.io)
registry	minikube	disabled	Google
registry-aliases	minikube	disabled	3rd party (unknown)
registry-creds	minikube	disabled	3rd party (UPMC Enterprises)
storage-provisioner	minikube	disabled	Google
storage-provisioner-gluster	minikube	disabled	3rd party (Gluster)
volumesnapshots	minikube	disabled	Kubernetes

Figura 3. Addons en Minikube

5.1.4. Despliegue de un servicio

En Kubernetes, un servicio es un objeto que define una forma abstracta de exponer aplicaciones que se ejecutan en un conjunto de pods a través de la red. Los servicios proporcionan una abstracción que permite a las aplicaciones comunicarse entre sí o con componentes externos sin tener que preocuparse por la dirección IP de los pods individuales o su ubicación exacta en el clúster.

Para desplegar nuestro primer servicio hemos utilizado el comando: **kubectl create deployment test -image=nginx -port=80** para crear un deployment con la imagen de nginx, para posteriormente crear el servicio de este mediante: **kubectl expose deployment test -port=80 -type=NodePort**.

En la siguiente imagen podemos observar los pasos que hemos realizado y como accedemos al servicio utilizando curl.

```
PS C:\Windows\system32> kubectl create deployment test --image=nginx --port=80
deployment.apps/test created
PS C:\Windows\system32> kubectl expose deployment test --port=80 --type=NodePort
service/test exposed
PS C:\Windows\system32> kubectl get svc
NAME         TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes   ClusterIP     10.96.0.1     <none>       443/TCP          77d
miweb        NodePort      10.96.19.96   <none>       80:32359/TCP     24m
test         NodePort      10.107.122.158 <none>       80:30928/TCP     5s
web-dash     LoadBalancer 10.108.23.56  <pending>    8080:32018/TCP   63d
PS C:\Windows\system32> curl 192.168.59.100:30928

statusCode      : 200
statusCodeDescription : OK
content         : <!DOCTYPE html>
                <html>
                <head>
                <title>Welcome to nginx!</title>
                <style>
                html { color:scheme: light dark; }
                body { width: 35em; margin: 0 auto;
                font-family: Tahoma, Verdana, Arial, sans-serif; }
                </style...
RawContent      : HTTP/1.1 200 OK
                Connection: keep-alive
                Accept-Ranges: bytes
                Content-Length: 615
                Content-Type: text/html
                Date: Fri, 18 Aug 2023 10:01:53 GMT
                ETag: "64dbafc8-267"
                Last-Modified: Tue, 15 Aug 2023 ...
Forms           : {}
Headers         : {[Connection, keep-alive], [Accept-Ranges, bytes], [Content-Length, 615], [Content-Type,
                text/html]...}
Images          : {}
InputFields     : {}
Links           : {@{innerHTML=nginx.org; innerText=nginx.org; outerHTML=<A href="http://nginx.org/">nginx.org</A>;
                outerText=nginx.org; tagName=A; href=http://nginx.org/}, @{innerHTML=nginx.com;
                innerText=nginx.com; outerHTML=<A href="http://nginx.com/">nginx.com</A>; outerText=nginx.com;
                tagName=A; href=http://nginx.com/}}
```

Figura 4. Servicio en Minikube

5.2. Despliegue con Kubeadm

Kubeadm es una herramienta de línea de comandos que se utiliza para facilitar la instalación, configuración y administración de clústeres Kubernetes. Está diseñada para simplificar el proceso de creación de un clúster, permitiendo a los usuarios configurar fácilmente los nodos maestros y trabajadores de Kubernetes.

Con Kubeadm, se pueden seguir una serie de pasos predefinidos para establecer un clúster Kubernetes de manera rápida y consistente. La herramienta automatiza gran parte del trabajo requerido, lo que facilita la configuración inicial del clúster y garantiza la coherencia entre los diferentes nodos.



5.2.1. Preparación del entorno virtual

Para implementar kubernetes mediante kubeadm necesitamos de una infraestructura para poder trabajar que puede ser un entorno con servidores reales donde puedes usar uno como master y los otros workers. Como esta sección va a ser solo para hacer pruebas, he elegido utilizar máquinas virtuales para simplificar el proceso.

Hemos decido crear 4 máquinas virtuales, teniendo una arquitectura de 1 master junto con 2 workers y servidor nfs, para contar con almacenamiento persistente, con las siguientes especificaciones:

Nombre	Hilos	RAM	Storage	Sistema operativo	RED
master	2	4	20GB	Ubuntu Server	NatNetwork y HostOnlyAdapter
worker1	2	4	20GB	Ubuntu Server	NatNetwork y HostOnlyAdapter
worker2	2	4	20GB	Ubuntu Server	NatNetwork y HostOnlyAdapter
nfsserver	2	4	25GB	Ubuntu Server	NatNetwork y HostOnlyAdapter

Nombre	IP
master	10.0.2.10 y 192.168.59.116
worker1	10.0.2.11 y 192.168.59.117
worker2	10.0.2.12 y 192.168.59.118
nfsserver	10.0.2.13 y 192.168.59.119

5.2.2. Configuración de las máquinas virtuales

Para empezar a configurar las máquinas virtuales he decidido realizar todos los pasos comunes que necesitan las 3 máquinas que formarán parte del cluster kubernetes en una sola máquina para más tarde clonarla y así agilizar el procedimiento.

ACTUALIZAR SISTEMA:

```
sudo apt update
```

```
sudo apt upgrade
```

Estos comandos nos permiten actualizar el listado de paquetes disponibles de nuestro sistema y actualizarlos.

PERMITIR PUERTOS FIREWALL:

```
sudo ufw allow 6443/tcp
```

```
sudo ufw allow 2379:2380/tcp
```

```
sudo ufw allow 10250/tcp
```

```
sudo ufw allow 10259/tcp
```

```
sudo ufw allow 10257/tc
```

Estos serían los comandos a realizar si queremos permitir el tráfico de kubernetes dentro de nuestro cluster, en mi caso directamente he hecho un:

```
sudo ufw disable
```

Este comando desactiva el firewall ya que esto es un entorno de pruebas y no hay ningún problema en hacerlo. En caso de que estemos configurando esto en un entorno de producción deberíamos tener el firewall activado y deshabilitar los puertos manualmente mediante los comandos mencionados anteriormente.

HABILITAR MÓDULOS DEL KERNEL

Activamos estos módulos del kernel mediante:

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```



Para que al reiniciar el equipo no se deshabiliten los módulos, creamos este fichero de configuración:

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf  
overlay  
br_netfilter  
EOF
```

Añadimos estas líneas para permitir el tráfico de kubernetes entre nodos:

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-iptables = 1  
net.bridge.bridge-nf-call-ip6tables = 1  
net.ipv4.ip_forward = 1  
EOF
```

Aplicamos toda la configuración anterior con este último comando:

```
sudo sysctl -system
```

DESHABILITAR SWAP

Kubernetes no funciona bien con la memoria swap, así que hay que editar el fichero `/etc/fstab` y comentar la línea donde se monta la partición swap para evitar que se monte cada vez que iniciemos el sistema, después de ello como aun tenemos activa la memoria swap ya que no hemos reiniciado, utilizamos el siguiente comando para desactivarla:

```
sudo swapoff -a
```

INSTALAR CONTAINERD

Nos traemos el repositorio donde se haya containerd:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --  
dearmor -o /usr/share/keyrings/docker.gpg
```

```
echo \
```



```
"deb [arch=$(dpkg --print-architecture) signed-  
by=/usr/share/keyrings/docker.gpg]  
https://download.docker.com/linux/ubuntu \
```

```
$(lsb_release -cs) stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null
```

Actualizamos la lista de repositorios para que aparezca el paquete de containerd y podamos instalarlo:

```
sudo apt update
```

Instalamos containerd:

```
sudo apt install containerd.io
```

Modificamos el archivo de configuración(*/etc/containerd/config.toml*) de containerd para añadir *SystemdCgroup=true*. Y paramos y iniciamos el servicio de containerd para que se aplique el cambio:

```
sudo systemctl stop containerd
```

```
sudo systemctl start containerd
```

INSTALAR KUBERNETES

Instalamos las dependencias de kubernetes:

```
sudo apt install apt-transport-https ca-certificates curl -y
```

Añadimos el repositorio de kubernetes junto con su key y actualizamos la lista de repositorios para poder instalarlo:

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg  
https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-  
keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo  
tee /etc/apt/sources.list.d/kubernetes.list
```

```
sudo apt update
```



Instalamos kubeadm:

```
sudo apt install kubelet kubeadm kubectl
```

INSTALAMOS FLANNEL

Flannel es un CNI(Container Network Interface) que es una herramienta de red que nos permite facilitar la comunicacion entre contenedores.

Creamos el directorio y descargamos el binario de flannel y le damos permisos de ejecucion para que cuando añadamos el pod de flannel pueda ejecutarlo:

```
mkdir -p /opt/bin/
```

```
sudo curl -fsSLo /opt/bin/flanneld https://github.com/flannel-io/flannel/releases/download/v0.19.0/flanneld-amd64
```

```
sudo chmod +x /opt/bin/flanneld
```

CLONAMOS LA MÁQUINA

Clonamos nuestra máquina para tener nuestro master y los 2 workers y configuramos el nombre y la red de todas las máquinas para que puedan comunicarse entre ellas.

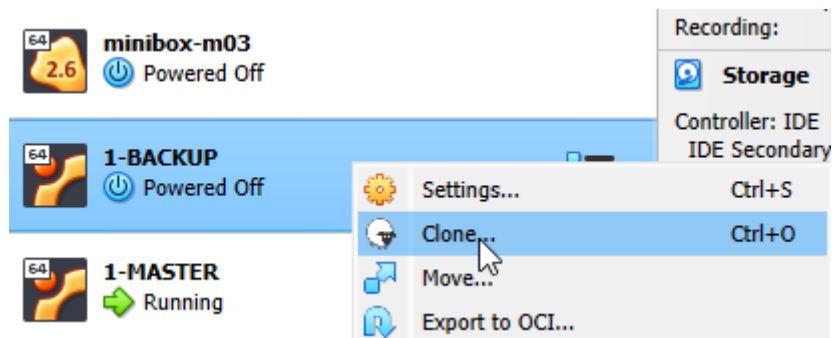


Figura 5. Clonado de la máquina virtual

```

user@master:~$ cat /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [10.0.2.10/24]
      routes:
        - to: default
          via: 10.0.2.1
      nameservers:
        addresses: [1.1.1.1,1.0.0.1]
    enp0s8:
      dhcp4: true
      dhcp-identifier: mac
  version: 2
user@master:~$

```

Figura 6. Configuración de red master

```

user@worker1:~$ cat /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [10.0.2.11/24]
      routes:
        - to: default
          via: 10.0.2.1
      nameservers:
        addresses: [1.1.1.1,1.0.0.1]
    enp0s8:
      dhcp4: true
      dhcp-identifier: mac
  version: 2
user@worker1:~$ _

```

Figura 7. Configuración de red worker 1

```

user@worker2:~$ cat /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [10.0.2.12/24]
      routes:
        - to: default
          via: 10.0.2.1
      nameservers:
        addresses: [1.1.1.1,1.0.0.1]
    enp0s8:
      dhcp4: true
      dhcp-identifier: mac
  version: 2

```

Figura 8. Configuración de red worker 2

INSTALAR CONTROL PLANE

Ahora con las máquinas ya preparadas podemos empezar a montar el panel de control en el nodo maestro, para ello descargamos las imagenes de las partes que necesita kubernetes para funcionar:

```
sudo kubeadm config images pull
```

Iniciamos el cluster:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 \  
--apiserver-advertise-address=10.0.2.10 \  
--cri-socket=unix:///run/containerd/containerd.sock
```

Ejecutamos los siguientes comandos para poder ejecutar comandos en nuestro cluster:

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Por último aplicamos el yml de flannel para instalar el pod de flannel y poder tener comunicación entre nuestro nodos:

```
kubect1 apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml
```

AÑADIR NODOS WORKERS AL CLUSTER

Después de iniciar el control plane, nos devolverá por pantalla un comando que nos servirá para meter los nodos workers dentro del cluster:

```
kubeadm join 10.0.2.10:6443 --token po3sb1.oux4z76nwb0veuna \  
--discovery-token-ca-cert-hash  
sha256:f5068150fabaf85f3d04e19a395c60d19298ba441e2d9391e20df3267ea6cd2  
8
```

Después de utilizar este comando en nuestros nodos workers, formaran ambos parte del cluster y podemos comprobarlo mediante el comando: *kubect1 get nodes*.



```

root@master:~# kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
master       Ready    control-plane   16d   v1.27.3
worker1      Ready    <none>         16d   v1.27.3
worker2      Ready    <none>         16d   v1.27.3
root@master:~#

```

Figura 9. kubectl get nodes en local

5.3. Almacenamiento persistente (NFS SERVER)

En un clúster de Kubernetes, el almacenamiento persistente es esencial para garantizar que los datos críticos de las aplicaciones se mantengan disponibles incluso cuando los pods se reinician, escalan o se mueven entre nodos. Un sistema de almacenamiento persistente permite a las aplicaciones retener y acceder a los datos a largo plazo, evitando la pérdida de información y asegurando una continuidad operativa adecuada.

Aquí es donde entra en juego el servidor NFS (Network File System) en Kubernetes. NFS proporciona una solución confiable y centralizada para el almacenamiento persistente, permitiendo a los pods acceder a un sistema de archivos compartido a través de la red.

Para desplegar el servidor NFS se ha decidido crear otra máquina virtual con Ubuntu Server con 2 CPUs, 4 GB de RAM y un disco de 25GB de almacenamiento y con la misma configuración de red que el cluster para poder tener conexión entre sí.

5.3.1. Preparación de la máquina virtual

Lo primero de todo después de haber configurado nuestro sistema operativo y verificar que tenemos conectividad con el resto de máquinas del cluster.

COMANDOS

```
sudo lvcreate -n k8s_data -L 12G ubuntu-vg
```

```
sudo mkdir -p /data/k8s
```

```
sudo vim /etc/fstab
```

```
#K8s data mount point
```

```
/dev/data/k8s_data /data/k8s xfs defaults o o
```

```
sudo mount /data/k8s/
```

```
sudo apt install nfs-kernel-server
```

```
sudo vim /etc/exports  
  
/data/k8s/ 10.0.2.0/24(rw,no_root_squash)  
  
sudo systemctl restart nfs-server
```

5.3.2. Persistent volumes y Persistent volumes claims

En Kubernetes, los términos PV (Persistent Volume) y PVC (Persistent Volume Claim) están relacionados con la persistencia de datos y el almacenamiento en la orquestación de contenedores. Estos conceptos permiten que las aplicaciones tengan acceso a almacenamiento persistente y mantengan sus datos incluso cuando los contenedores se reinician o se mueven a diferentes nodos dentro del clúster.

Un Persistent Volume (PV) es un recurso de Kubernetes que representa una unidad de almacenamiento física o virtual, independiente del ciclo de vida del pod (unidades básicas de ejecución en Kubernetes). Es decir, un PV es una abstracción del almacenamiento que puede ser provisionado en cualquier lugar del clúster y tiene una existencia independiente.

Un Persistent Volume Claim (PVC) es una solicitud realizada por un pod para obtener almacenamiento persistente de un PV. En lugar de trabajar directamente con un PV, los pods interactúan con PVCs, solicitando un determinado tamaño y clase de almacenamiento. Cuando se crea un PVC, Kubernetes encuentra un PV compatible y lo vincula al PVC. Así, los pods pueden utilizar el almacenamiento solicitado a través del PVC sin preocuparse por los detalles de la infraestructura subyacente.

En resumen, los PVs son recursos de Kubernetes que representan almacenamiento persistente, mientras que los PVCs son las solicitudes que hacen los pods para obtener acceso a ese almacenamiento.

En el Anexo 1 podemos ver el fichero yaml con la configuración del PV y PVC que vamos a utilizar.

En este caso hemos creado un PV de 10Gb apuntando a nuestro servidor nfs con ip: 10.0.2.13 y apuntando al path que hemos compartido anteriormente.

Mientras que con el PVC estamos reclamando la totalidad de espacio que hemos reservado con el PV anterior.

5.3.3. Test del almacenamiento persistente

Para comprobar el correcto funcionamiento de nuestro servidor NFS, hemos creado un deployment nginx indicando el PVC que queremos utilizar en los apartados de volumes y volumeMounts como podemos ver en el Anexo 2.

Dentro del directorio que estamos compartiendo mediante el servidor nfs está ubicado el archivo php que se va a ver al acceder a nuestro servicio nginx.

```
root@nfsserver:/data/k8s# cat index.php
<h1>HTML5 Test Page</h1>
<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>
<?php
echo "Served from server " . $_SERVER['SERVER_ADDR'] . "<br>";
?>
```

Figura 10. index.php en nfsserver

Al acceder al servicio mediante http, podemos ver por pantalla que en efecto estamos accediendo a través de nuestro servidor NFS, por lo tanto, hemos conseguido el objetivo de tener almacenamiento persistente.

En el siguiente punto sobre el balanceador de carga veremos por qué estamos dando la IP del servidor del cual se sirve la página web.

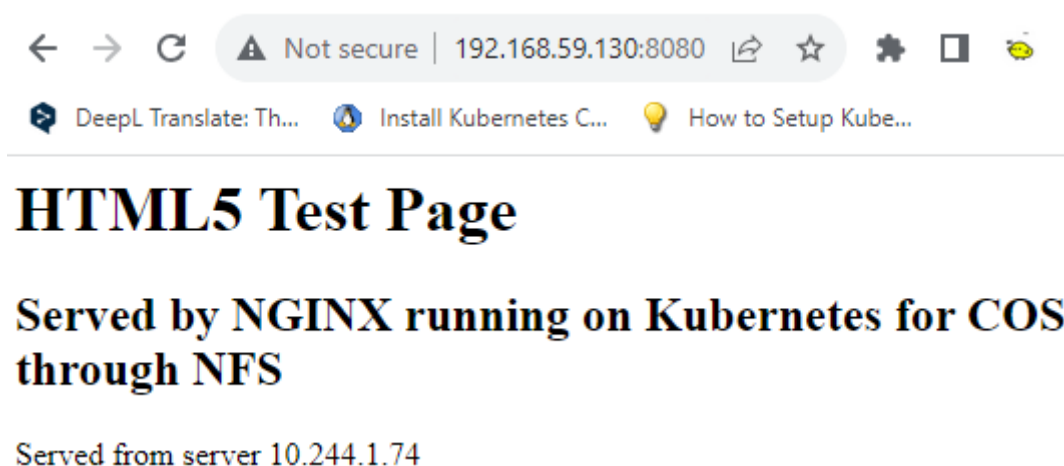


Figura 11. HTML desde el navegador

5.4. Load balancer (MetalLB)

MetalLB es una implementación de equilibrio de carga para Kubernetes que permite exponer servicios a través de direcciones IP externas y brinda flexibilidad y configurabilidad en entornos de clústeres de Kubernetes.

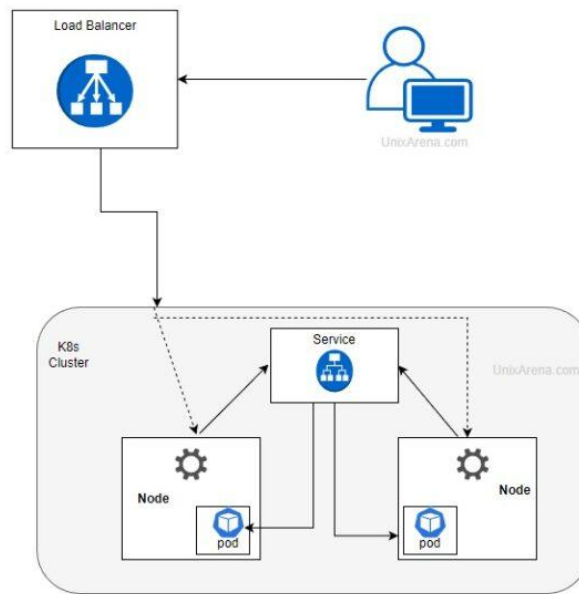


Figura 12. Maqueta del Load Balancer

Esto nos permite poder acceder a un servicio de nuestro cluster Kubernetes mediante una única ip y se encarga de repartir las solicitudes entre los distintos pods para repartir la carga.

5.4.1. Implementación de MetalLB

A continuación, se van a mostrar todos los comandos necesarios para dar de alta nuestro balanceador de carga:

```
MetalLB_RTAG=$(curl -s
https://api.github.com/repos/metallb/metallb/releases/latest|grep
tag_name|cut -d '"' -f 4|sed 's/v//')

echo $MetalLB_RTAG

mkdir ~/metallb

cd ~/metallb

wget
https://raw.githubusercontent.com/metallb/metallb/v\$MetalLB\_RTAG/config/manifests/metallb-native.yaml

kubectl apply -f metallb-native.yaml
```

Como último paso hay que aplicar el yaml del Anexo 3 en el que podemos ver las IPs que va a usar nuestro balanceador de carga y para que sean visibles al cluster mediante el L2Advertisement.

```
root@master:~# kubectl get pods -n metallb-system
NAME                                READY   STATUS    RESTARTS   AGE
controller-595f88d88f-16zzk         1/1     Running   4 (109m ago)  13d
speaker-5dpmz                        1/1     Running   12 (109m ago)  15d
speaker-dlgl2                        1/1     Running   11 (109m ago)  15d
speaker-pjdnk                       1/1     Running   11 (109m ago)  15d
```

Figura 13. Pods activos de MetalLB

5.4.2. Tests del balanceador

Como podemos ver en las siguientes imágenes, para utilizar el load balancer hay que utilizar la ip externa que tiene asignada nuestro servicio nginx y si nos damos cuenta pertenece al pool de ips que hemos configurado anteriormente.

```
user@master:~$ kubectl get svc
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes          ClusterIP     10.96.0.1       <none>           443/TCP          26d
nginx-service       LoadBalancer 10.104.246.150  192.168.59.130  8080:32733/TCP  14d
php-apache          ClusterIP     10.105.185.210  <none>           80/TCP           19d
```

Figura 14. Servicio nginx con ip externa

Para hacer las pruebas hemos desplegado un nginx con un php que nos dice la ip del servidor en el que está corriendo, en este caso nos dice la ip del pod donde está corriendo la instancia de nginx. Al tener 4 pods vemos como cada petición curl va a un pod distinto gracias a MetalLB.

```
root@master:~# kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE
nginx-deployment-85fc498d86-5cgkg   1/1    Running   0          44s   10.244.1.30    worker1
nginx-deployment-85fc498d86-9njnd   1/1    Running   2 (12m ago)  24h   10.244.2.34    worker2
nginx-deployment-85fc498d86-fxgsl   1/1    Running   1 (12m ago)  18h   10.244.2.35    worker2
nginx-deployment-85fc498d86-g5lkw   1/1    Running   0          44s   10.244.1.31    worker1
php-apache-5bdbb8dbf8-djl5q        1/1    Running   2 (12m ago)  18h   10.244.2.32    worker2
root@master:~# curl 192.168.59.130:8080
<h1>HTML5 Test Page</h1>
<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>
Served from server 10.244.1.31<br>
root@master:~# curl 192.168.59.130:8080
<h1>HTML5 Test Page</h1>
<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>
Served from server 10.244.2.35<br>
root@master:~# curl 192.168.59.130:8080
<h1>HTML5 Test Page</h1>
<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>
Served from server 10.244.1.30<br>
root@master:~# curl 192.168.59.130:8080
<h1>HTML5 Test Page</h1>
<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>
Served from server 10.244.2.34<br>
```

Figura 15. Load Balancer en acción

5.5. Horizontal Pod Autoscaling (HPA)

HPA (Horizontal Pod Autoscaler) es una característica de Kubernetes que permite escalar automáticamente el número de réplicas de un conjunto de pods basado en la utilización de recursos. También monitorea constantemente la carga de trabajo y ajusta dinámicamente el número de réplicas para mantener un equilibrio entre la capacidad de procesamiento y la demanda.

El HPA utiliza métricas como el uso de CPU o memoria para determinar cuándo escalar el número de pods hacia arriba o hacia abajo. Cuando la carga de trabajo aumenta y supera ciertos umbrales definidos, se aumenta el número de réplicas de los pods para asegurar que la aplicación pueda manejar la carga adicional. Por otro lado, cuando la carga disminuye, se reduce el número de réplicas para ahorrar recursos.

Es una herramienta muy útil para mantener una alta disponibilidad y eficiencia en las aplicaciones desplegadas. Permite que los recursos sean utilizados de manera óptima, escalando automáticamente la infraestructura en función de la demanda de la aplicación. Esto asegura que la aplicación pueda manejar picos de carga sin interrupciones y que los recursos no sean subutilizados durante períodos de baja demanda.

5.5.1. Implementación del autoescalador

Como el HPA ya es una herramienta nativa de Kubernetes simplemente necesitamos instalar un servidor de métricas para que el HPA pueda leer los valores que necesita para poder trabajar.

Para ello solo necesitamos instalar metrics server con el siguiente comando:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

5.5.2. Tests del autoescalador

Para comprobar el correcto funcionamiento del autoescalador hemos utilizado el yaml que podemos observar en el Anexo 4 que se encarga de hacer un deployment de un servidor apache con php junto con un servicio para poder acceder a él.

Después de aplicar el anterior yaml, hemos utilizado el siguiente comando que básicamente le dice al deployment que autoescale a partir del porcentaje consumido de CPU, en este caso al llegar al 50% debería autoescalar y que tenga como mínimo 1 pod y como máximo 10 pods.:



6. IBM Cloud

IBM Cloud es una plataforma de servicios en la nube ofrecida por International Business Machines Corporation (IBM), una de las compañías de tecnología más grandes y reconocidas del mundo. Esta plataforma brinda un amplio abanico de soluciones en la nube que permiten a empresas y desarrolladores crear, desplegar y gestionar aplicaciones y servicios de forma flexible, escalable y segura.

Con la visión de ayudar a las organizaciones a aprovechar el poder de la nube, IBM Cloud ofrece un conjunto diverso de servicios que abarcan desde la infraestructura básica hasta tecnologías de vanguardia, como inteligencia artificial, blockchain, Internet de las cosas (IoT) y análisis de datos avanzados.

La plataforma de IBM Cloud se adapta tanto a pequeñas startups como a grandes empresas, facilitando el desarrollo ágil de aplicaciones, la integración de sistemas y la adopción de nuevas tecnologías. Sus servicios en la nube permiten a los usuarios centrarse en la innovación y el crecimiento de sus negocios, sin preocuparse por la complejidad de la infraestructura subyacente.

Además, IBM Cloud ofrece una serie de ventajas y características destacadas, como una seguridad sólida y una arquitectura de red global, lo que garantiza la protección de los datos y la disponibilidad de los servicios en todo momento.



6.1. Registro y primeras impresiones

En el proceso de registro en IBM Cloud, lo primero que nos encontramos es un apartado donde debemos introducir nuestro correo electrónico junto a la contraseña que queramos utilizar para acceder al servicio.

Los siguientes pasos ya consisten en verificar la cuenta y aportar los datos de la tarjeta de crédito para poder acreditar que la cuenta es única y poder optar al crédito de 200\$ que nos ofrece IBM durante un mes para poder probar sus servicios.

Por último, también nos hace dar de alta un doble factor de autenticación (MFA) para así tener una capa extra de seguridad a la hora de acceder a nuestra cuenta.



The screenshot shows the 'Create an IBM Cloud account' registration page. At the top left is the IBM logo. Below it, the text reads 'Create an IBM Cloud account' and 'Already have an IBM Cloud account? [Log in](#)'. The main form area is titled 'Account information' and contains two input fields: 'Email' with a red border and a blue arrow icon, and 'Password' with a blue arrow icon. Below the email field is the text 'Enter an email address.' and below the password field is a 'Next' button with a downward arrow. To the right of the form, there are two promotional sections: 'Get started with a USD 200 credit' with the subtext 'Receive a credit for your first USD 200 of apps and services on us.' and 'Build your journey to public cloud' with the subtext 'Build, deploy, and manage solutions in IBM's public cloud.' At the bottom of the form is a 'Continue' button with a rightward arrow.

Figura 18. IBM Cloud registro

Lo siguiente que nos encontramos es un dashboard que nos muestra todo lo que podemos hacer en IBM Cloud.

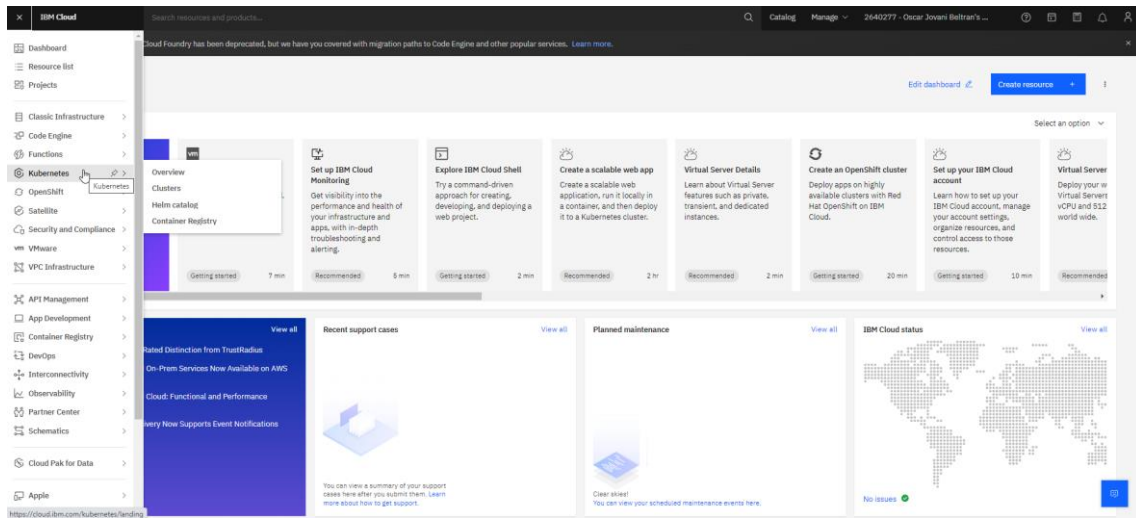


Figura 19. IBM Cloud dashboard

Si navegamos hasta la sección de “Billing and usage” y accedemos a las promociones activas podemos observar que al registrarnos por primera vez nos dan un crédito de 200\$ que usaremos para desplegar nuestro cluster y hacer todas las pruebas que necesitemos.

Promotion	Remaining Credit	Starting Credit
Manual credit ⓘ	\$191.37	\$200.00

Items per page: 10 ▾ 1-1 of 1 items

Figura 20. IBM Cloud 200\$ credito

Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

Y si volvemos a la página anterior podemos observar lo que llevamos consumido durante el último periodo de cobro y también podemos asignar un límite de gasto para evitar algún susto.

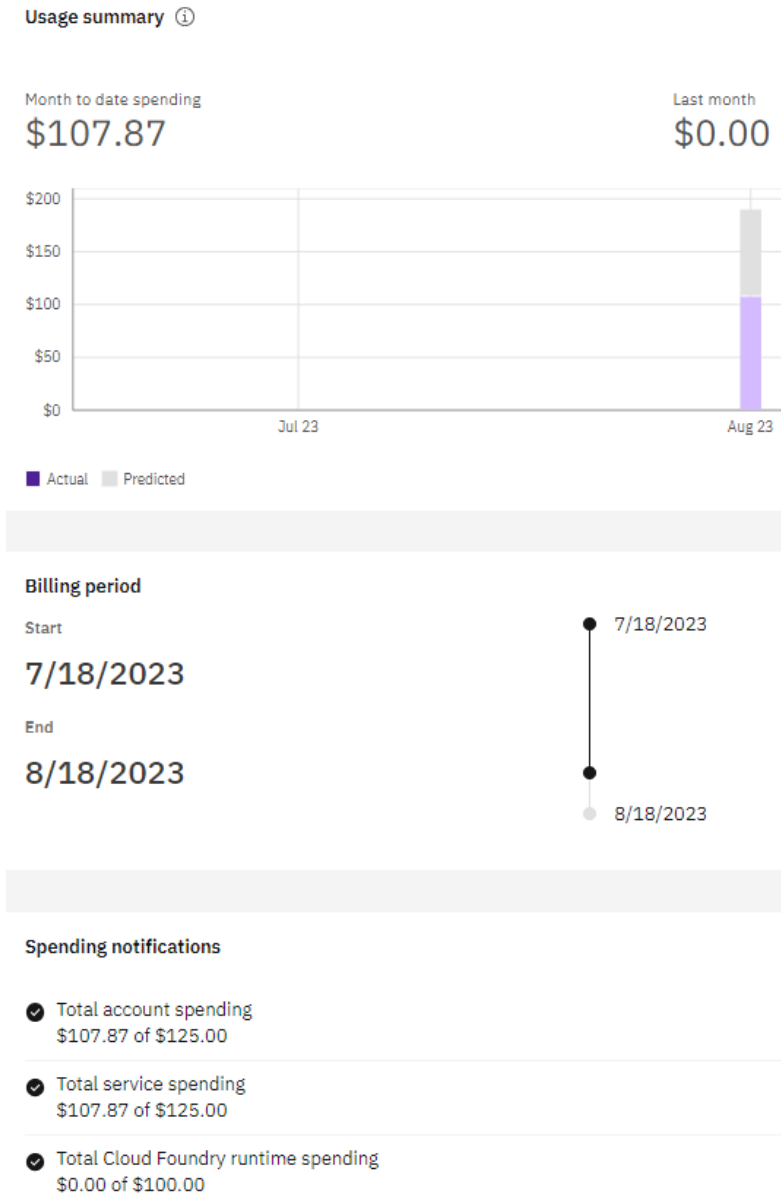
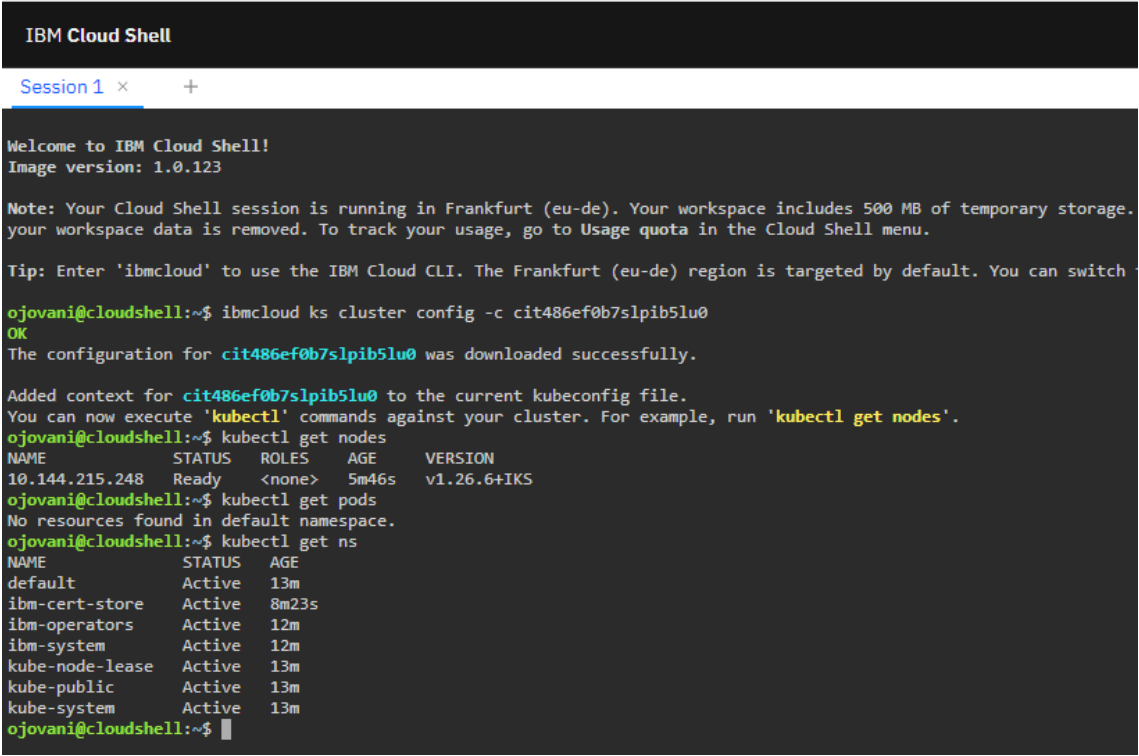


Figura 21. Gasto mensual en IBM Cloud

6.2. IBM Cloud shell

La "IBM Cloud Shell" es una herramienta que ofrece una interfaz de línea de comandos interactiva basada en el navegador web para interactuar con los servicios y recursos de IBM Cloud. Esta herramienta proporciona un entorno de línea de comandos dentro del navegador, lo que permite a los usuarios ejecutar comandos y administrar sus recursos en la nube sin necesidad de instalar ninguna herramienta adicional en su computadora local. Aunque también se puede instalar en linux o windows como si fuera una aplicación y utilizarla así desde la línea de comandos.



```
IBM Cloud Shell
Session 1 x +
Welcome to IBM Cloud Shell!
Image version: 1.0.123
Note: Your Cloud Shell session is running in Frankfurt (eu-de). Your workspace includes 500 MB of temporary storage.
your workspace data is removed. To track your usage, go to Usage quota in the Cloud Shell menu.
Tip: Enter 'ibmcloud' to use the IBM Cloud CLI. The Frankfurt (eu-de) region is targeted by default. You can switch
ojovani@cloudshell:~$ ibmcloud ks cluster config -c cit486ef0b7slpib5lu0
OK
The configuration for cit486ef0b7slpib5lu0 was downloaded successfully.
Added context for cit486ef0b7slpib5lu0 to the current kubeconfig file.
You can now execute 'kubectl' commands against your cluster. For example, run 'kubectl get nodes'.
ojovani@cloudshell:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE      VERSION
10.144.215.248      Ready    <none>   5m46s   v1.26.6+IKS
ojovani@cloudshell:~$ kubectl get pods
No resources found in default namespace.
ojovani@cloudshell:~$ kubectl get ns
NAME                STATUS    AGE
default             Active   13m
ibm-cert-store      Active   8m23s
ibm-operators       Active   12m
ibm-system          Active   12m
kube-node-lease     Active   13m
kube-public         Active   13m
kube-system         Active   13m
ojovani@cloudshell:~$
```

Figura 22. IBM Cloud shell web

Como vemos en la imagen, esta es la CLI con interfaz web que nos ofrece IBM Cloud, una vez dentro de ella, para poder tener acceso a nuestro cluster Kubernetes solamente hay que utilizar el siguiente comando:

```
ibmcloud ks cluster config -c [IDdelCluster]
```



Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

También podemos usar la CLI de IBM Cloud desde nuestro sistema operativo y así no tener que depender de la interfaz web, en el caso de tener un dispositivo windows, solamente debemos ejecutar el siguiente comando para instalar la CLI en local:

```
iex (New-Object  
Net.WebClient).DownloadString('https://clis.cloud.ibm.com/install/powe  
rshell')
```

Y será necesario instalar el container service para poder acceder al cluster:

```
ibmcloud plugin install ks
```

Finalmente debemos instalar kubectl en nuestra máquina local para poder ejecutar comandos en nuestro cluster y con ello ya habíamos acabado la configuración:

```
curl.exe -LO  
"https://dl.k8s.io/release/v1.27.4/bin/windows/amd64/kubectl.exe"
```

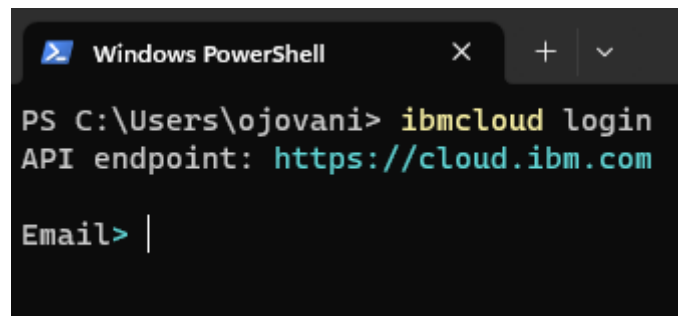


Figura 23. IBM Cloud shell on premise

6.3. IBM Kubernetes Service (IKS)

IBM Cloud Kubernetes Service es un servicio administrado de Kubernetes proporcionado por IBM Cloud. Kubernetes es una plataforma de código abierto para la orquestación y administración de contenedores, y IKS es la implementación de Kubernetes ofrecida por IBM para que los usuarios puedan desplegar, gestionar y escalar sus aplicaciones en clústeres de Kubernetes de forma sencilla y eficiente.

Con IKS, IBM Cloud simplifica la creación y gestión de clústeres de Kubernetes, lo que permite a los desarrolladores y equipos de operaciones enfocarse en desarrollar y desplegar aplicaciones en contenedores sin preocuparse por la complejidad subyacente de la infraestructura.

6.3.1. Free tier cluster

El Free Tier Cluster es una oferta en la plataforma de Kubernetes en IBM que permite a los usuarios acceder a un clúster de Kubernetes sin costo alguno dentro de ciertas limitaciones. Algunos aspectos clave de los clústeres de nivel gratuito incluyen su gratuidad inicial, recursos limitados, aplicaciones básicas, tiempo limitado y soporte limitado.

Este cluster consiste básicamente en un nodo worker donde puedes hacer todas las pruebas que quieras durante 30 días, esto viene muy bien cuando queremos probar la plataforma y estar seguros antes de recurrir a un plan de Pay as you Go, donde pagas por lo que utilizas.

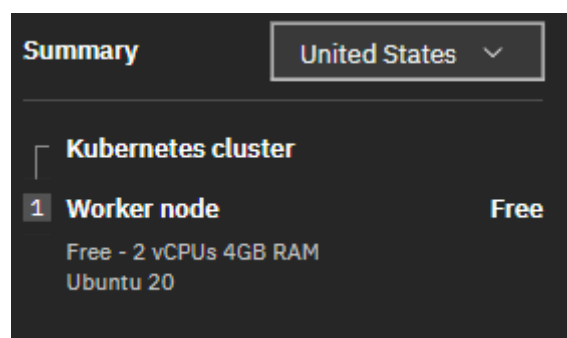


Figura 24. Summary free cluster

Despliegue de un cluster Kubernetes altamente disponible en IBM Cloud

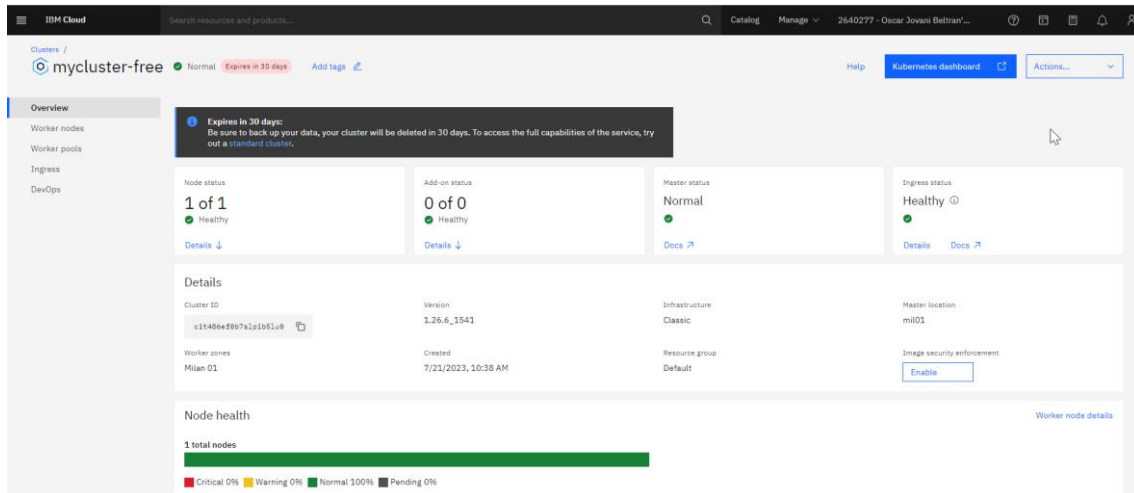


Figura 25. Dashboard free cluster

6.3.2. Standard tier cluster

La opción estándar para desplegar un cluster Kubernetes en IBM Cloud nos brinda una oferta que proporciona a los usuarios un paso adelante en términos de características y recursos en comparación con los clústeres de nivel gratuito. Estos clusters representan un compromiso con la escalabilidad, el rendimiento y la integración en el entorno de contenedores administrados de IBM Cloud.

Los clústeres de Kubernetes de nivel estándar ofrecen características más avanzadas en comparación con las ofertas gratuitas. Estas características incluyen opciones de escalabilidad mejorada, seguridad más robusta, integración con otros servicios de IBM Cloud y capacidades de gestión más avanzadas.

También permiten a los usuarios configurar recursos de manera más flexible. Puedes seleccionar diferentes tamaños de clúster y asignaciones de recursos según las necesidades de tu carga de trabajo.

Aunque los clústeres de nivel estándar brindan beneficios adicionales, también conllevan costos. Los precios varían según el tamaño del clúster, los recursos asignados y las características específicas.

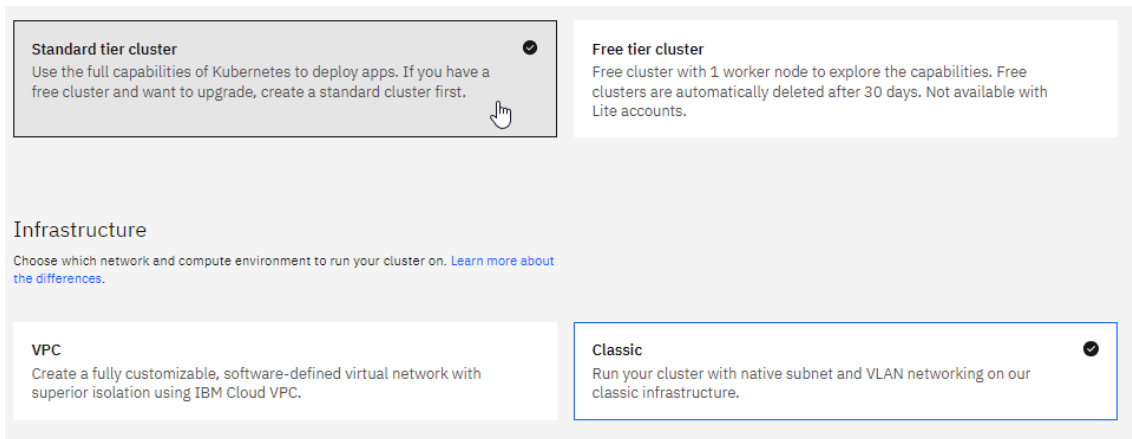


Figura 26. Selección tipo cluster IBM Cloud

Como hemos visto en la imagen anterior, después de seleccionar la opción de Standard tier cluster, la página nos pide seleccionar que tipo de infraestructura queremos tener para nuestro cluster Kubernetes:

En IBM Cloud, la diferencia principal entre un clúster de Kubernetes clásico y un clúster de Kubernetes con Virtual Private Cloud (VPC) radica en cómo están configuradas las redes y la seguridad para el clúster.

Clúster de Kubernetes clásico:

En un clúster de Kubernetes clásico en IBM Cloud, el clúster se implementa en una red compartida y gestionada por IBM Cloud. Esto significa que el clúster comparte la infraestructura de red con otros recursos de IBM Cloud y no tiene un control granular sobre la configuración de la red. La seguridad en el clúster se basa en políticas de acceso y permisos proporcionadas por IBM Cloud IAM (Identity and Access Management).

Clúster de Kubernetes con Virtual Private Cloud (VPC):

Un clúster de Kubernetes con VPC en IBM Cloud ofrece una mayor personalización y control sobre la red y la seguridad. En lugar de utilizar una red compartida, este tipo de clúster se despliega dentro de una VPC dedicada, lo que permite crear y configurar sus propias subredes, gateways, reglas de firewall y otros componentes de red. Esto proporciona a los usuarios un aislamiento más sólido y mayor seguridad para el clúster y sus aplicaciones.

Ventajas del clúster con VPC:

Mayor aislamiento y seguridad: Al estar en una VPC dedicada, el clúster no comparte la infraestructura de red con otros recursos, lo que reduce el riesgo de interferencia entre diferentes entornos.

Personalización de redes: Permite configurar subredes y ajustar las reglas de firewall para adaptarse a los requisitos específicos de la aplicación.

Conectividad privada: Permite una conectividad privada más segura con otros recursos de la VPC, lo que es beneficioso para aplicaciones que requieren comunicación interna.

En resumen, un clúster de Kubernetes con VPC en IBM Cloud proporciona un mayor control sobre la red y la seguridad, lo que lo hace más adecuado para aplicaciones que requieren niveles más altos de aislamiento y personalización. Sin embargo, también es importante tener en cuenta que, al tener más control y personalización, también implica una mayor responsabilidad en la configuración y gestión de la red y la seguridad del clúster.

6.4. Red Hat Openshift

OpenShift es una plataforma de aplicaciones basada en Kubernetes desarrollada por Red Hat y que ahora forma parte IBM ya que esta adquirió Red Hat en 2019. Esta plataforma simplifica el proceso de implementación, administración y escala de aplicaciones en contenedores. Proporciona un entorno unificado para desarrolladores y operadores.

Agrega funcionalidades y capacidades adicionales a Kubernetes, como automatización de implementación, integración continua, entrega continua (CI/CD), monitoreo avanzado, escalado automático y capacidades de red avanzadas.

OpenShift en IBM Cloud se centra en la seguridad y el cumplimiento. Proporciona características de seguridad robustas, como aislamiento de recursos, políticas de acceso granulares y herramientas de escaneo de vulnerabilidades.

Como producto de Red Hat, OpenShift ofrece soporte empresarial confiable y opciones de suscripción que pueden adaptarse a las necesidades de diferentes organizaciones. Esto incluye acceso a actualizaciones y parches, así como asistencia técnica.

También se integra con otros servicios y ofertas de IBM Cloud, lo que permite a los desarrolladores construir aplicaciones más completas y conectadas. También es compatible con tecnologías de nube híbrida y MultiCloud.

6.5. Container registry

El Container Registry en IBM Cloud es un servicio que almacena imágenes de contenedores Docker. Estas imágenes son esenciales para construir y desplegar aplicaciones en entornos de contenedores y Kubernetes. El registro ofrece almacenamiento seguro y gestión de versiones para las imágenes. También integra características de seguridad, como autenticación y autorización, y se integra con flujos de trabajo de CI/CD. Permite la colaboración, el control de acceso y la visibilidad sobre el uso de imágenes. Con múltiples ubicaciones globales, garantiza disponibilidad y baja latencia.

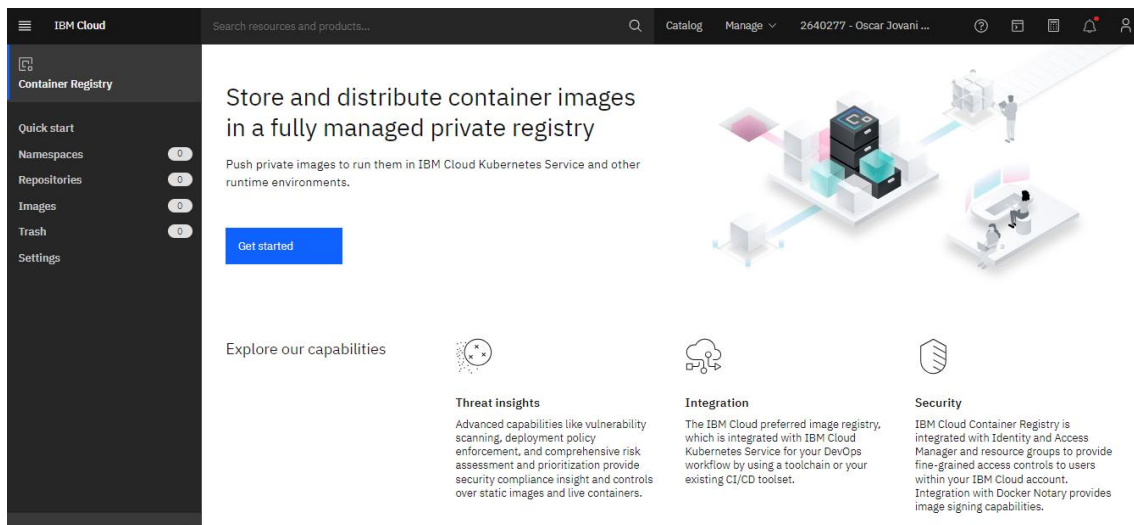


Figura 27. Container registry en IBM Cloud

7. Despliegue del cluster en IBM Cloud

IBM nos ofrece múltiples opciones a la hora de desplegar un cluster Kubernetes como hemos visto en el apartado anterior. Para llevar a cabo este proyecto hemos decidido por desplegar un cluster Kubernetes clásico con la siguiente configuración de nodos:

- 1 nodo maestro con una configuración desconocida, ya que IBM Cloud no nos da acceso al nodo maestro
- 2 nodos trabajadores con 2 vCPUs 4GB RAM cada uno
-

```
ojovani@cloudshell:~$ kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
10.127.70.14        Ready    <none>   65m   v1.26.6+IKS
10.127.70.3         Ready    <none>   67m   v1.26.6+IKS
ojovani@cloudshell:~$ kubectl get pods
No resources found in default namespace.
ojovani@cloudshell:~$ kubectl get ns
NAME                STATUS    AGE
default             Active    75m
ibm-cert-store      Active    68m
ibm-observe         Active    70m
ibm-operators       Active    74m
ibm-system          Active    74m
kube-node-lease     Active    75m
kube-public         Active    75m
kube-system         Active    75m
ojovani@cloudshell:~$
```

Figura 28. kubectl get nodes en IBM Cloud

Esta configuración siendo la más barata posible teniendo 2 nodos trabajadores nos da un gasto mensual de 0.21€/hora, lo que se traduce a 155.79€/mes. Hay que resaltar que esta configuración no nos da alta disponibilidad, ya que se nos dispararía el precio para hacer las pruebas.

Summary Spain

Kubernetes cluster

2 Worker nodes €0.21/hr

u3c.2x4 - 2 vCPUs 4GB RAM
Virtual - shared
x86-64
Ubuntu 20

Usage-based charges

The charges for these services are based on actual usage after provisioning.

Activity tracker [View pricing](#)

Service: IBM Cloud Activity Tracker
Name: activity-frankfurt-NN
Location: Frankfurt
Plan: 7 day Event Search

Logging [View pricing](#)

Service: IBM Log Analysis
Name: logs-frankfurt-Dx
Location: Frankfurt
Plan: 7 day Log Search

Monitoring [View pricing](#)

Service: IBM Cloud Monitoring
Name: metrics-frankfurt-QQ
Location: Frankfurt
Plan: Graduated Tier

Total estimated cost €155.79/mo

Figura 29. Coste del cluster

7.1. Despliegue de un servicio con kube dashboard

El Kubernetes Dashboard es una herramienta de interfaz gráfica de usuario que facilita la gestión y supervisión de clústeres de Kubernetes de manera intuitiva y visual. Proporciona una interfaz gráfica fácil de usar que permite a los desarrolladores, administradores y operadores de clústeres Kubernetes realizar tareas de administración, monitoreo y depuración sin necesidad de escribir comandos de línea.

Los usuarios pueden realizar acciones como la creación y escalado de pods, la monitorización del estado de los recursos, la visualización de registros de contenedores y el análisis de métricas de rendimiento. Además, el Dashboard proporciona una vista detallada del estado del clúster y sus componentes, lo que facilita la identificación y solución de problemas de manera más rápida y efectiva.

IBM Cloud nos ofrece esta herramienta directamente sin tener que realizar ninguna configuración adicional en nuestro clúster. Para ello solo tenemos que ir a la página de nuestro cluster y arriba a la derecha nos aparecerá el botón para acceder a nuestro Kubernetes Dashboard.

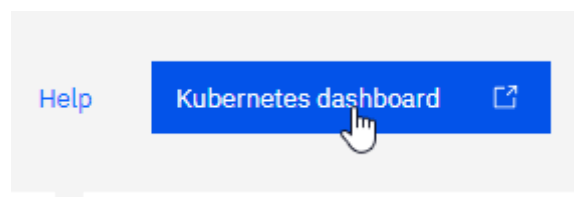


Figura 30. Kubernetes dashboard button

Dentro de este tenemos la posibilidad de crear nuestros servicios de una forma muy sencilla, ya que solamente tenemos que escribir el nombre de nuestra aplicación, que imagen vamos a utilizar, número de pods que queremos y finalmente decidir si queremos que sea una aplicación interna o externa junto con los puertos que queramos utilizar.

Figura 31. Create en kube dashboard

Después de crear el despliegue, podemos chequear que todo haya funcionado bien haciendo un **kubectl get pods** y **kubectl get svc**, donde encontraremos un svc tipo load balancer con una ip externa que se nos habrá asignado, con la cual podremos acceder a nuestro servicio.

```

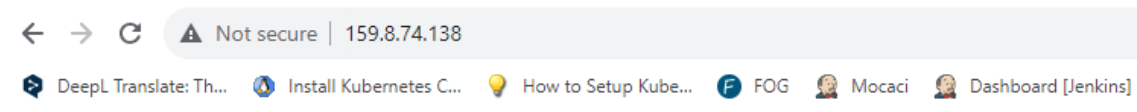
ojovani@cloudshell:~$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
web-6f786cbbdb-692p6  1/1     Running   0           5d3h
web-6f786cbbdb-wfv4w  1/1     Running   0           5d3h
ojovani@cloudshell:~$ kubectl get svc
NAME                TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes          ClusterIP      172.21.0.1    <none>         443/TCP          5d5h
web                  LoadBalancer   172.21.207.30 159.8.74.138  80:300892/TCP   5d3h

```

Figura 32. Load Balancer kubectl get svc



Al acceder a esta ip mediante un navegador web podremos ver la página de bienvenida de nuestro servidor web nginx.



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Figura 33. Página web nginx IBM

7.2. Load balancer en IBM Cloud

Un load balancer es un recurso que permite distribuir el tráfico de red entre los nodos de un clúster para garantizar una alta disponibilidad y mejorar el rendimiento de las aplicaciones. En IBM Cloud, se pueden utilizar servicios de Load Balancer para exponer nuestras aplicaciones a través de una dirección IP pública y gestionar la distribución del tráfico.

A diferencia del despliegue en local donde hemos tenido que instalar MetalLB para poder tener balanceo de carga, IBM nos proporciona directamente un balanceador de carga junto a varias ips públicas para poder utilizarlo.

Test del load balancer

Para poder hacer una prueba de este, hemos utilizado el despliegue del servidor nginx anterior y le hemos añadido un pod más para observar mejor el comportamiento del balanceador.

Para ello también hemos utilizado el comando **kubectl exec -it [nombredelpod] bash** para abrir una shell en cada pod y así modificar el fichero html de cada pod que se encuentra en el path **/usr/share/nginx/html/index.html** y así poder diferenciarlos correctamente durante la prueba.

Como vemos en la siguiente imagen, a medida que vamos haciendo curls, nos ofrece la página web un pod diferente cada vez, por lo tanto podemos confirmar que el balanceador de carga está funcionando correctamente.

```
ojovani@cloudshell:~$ curl 159.8.74.138
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body>
<h1>Welcome to POD 3!
</body>
</html>
ojovani@cloudshell:~$ curl 159.8.74.138
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body>
<h1>Welcome to POD 1</h1>
</body>
</html>
ojovani@cloudshell:~$ curl 159.8.74.138
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
</head>
<body>
<h1>Welcome to POD 2!
</body>
</html>
```

Figura 34. Load balancer IBM Cloud

7.3. Persistent Storage en IBM Cloud

En IBM Cloud, el concepto de "persistent storage" se refiere a la capacidad de mantener y conservar datos más allá del ciclo de vida de una instancia de aplicación o contenedor. Esto es esencial para garantizar que los datos críticos se conserven incluso cuando las instancias de aplicaciones o contenedores se crean, se eliminan o se actualizan.

La implementación de almacenamiento persistente en IBM Cloud te permite asegurarte de que tus datos se mantengan intactos y accesibles a pesar de los cambios en las instancias de aplicaciones o contenedores.

7.3.1. Diferencias entre storages

En IBM Cloud tenemos diferentes formas de almacenar nuestros datos de manera persistente, pero las 2 principales formas de hacerlo son mediante:

El **File Storage** es un tipo de servicio de almacenamiento en el que los datos se organizan y almacenan en forma de archivos y directorios, como en un sistema de archivos tradicional. Se utiliza para almacenar y acceder a archivos, como documentos, imágenes, videos y otros tipos de contenido. En un entorno de File Storage, los datos se pueden compartir y acceder desde múltiples máquinas y dispositivos a través de protocolos de red como NFS o SMB. Esto es particularmente útil en situaciones en las que se necesita acceso compartido a los mismos archivos desde diferentes ubicaciones o aplicaciones. El almacenamiento de archivos se ve como una jerarquía de carpetas y archivos, lo que lo hace adecuado para situaciones donde la estructura de los datos es importante.

El **Block Storage**, por otro lado, es un tipo de almacenamiento en el que los datos se almacenan en bloques individuales, como si fueran sectores de un disco duro. Cada bloque se trata como una unidad de almacenamiento independiente y se accede a través de identificadores únicos. A diferencia del File Storage, el Block Storage no maneja archivos ni estructuras de directorios; en su lugar, proporciona "bloques" que los sistemas operativos o aplicaciones pueden utilizar como si fueran dispositivos de almacenamiento. El Block Storage es altamente eficiente para aplicaciones que requieren un acceso rápido y directo a los datos y para escenarios donde se necesita un control preciso sobre el formato y la estructura de almacenamiento.

La principal diferencia entre File Storage y Block Storage radica en cómo se almacenan y acceden los datos. File Storage se centra en organizar archivos en una jerarquía de carpetas, permitiendo el acceso compartido y eficiente a través de protocolos de red. Por otro lado, Block Storage se enfoca en el almacenamiento basado en bloques, ofreciendo un alto rendimiento y control, pero sin la organización de archivos y carpetas propia del File Storage.

También existe el **Object Storage** y el **Cloud Backup** que están más dirigidos a grandes empresas con necesidades muy grandes de almacenamiento, por ello no vamos a explicar en qué consisten.

7.3.2. Tests con File Storage

En nuestro cluster hemos elegido utilizar el File Storage ya que es más simple que el Block Storage y está más enfocado a sistemas que no requieren unas necesidades de almacenamiento muy elevadas.

Lo primero que debemos hacer es crear el PVC de 20GB, ya que es lo mínimo que nos permite IBM Cloud contratar, para ello hemos utilizado un fichero yaml que se puede encontrar en el (Anexo 5), donde indicamos el tamaño del PVC y el storage class que queremos utilizar, esto es básicamente para elegir qué velocidad de lectura y escritura queremos para nuestro storage, IBM nos proporciona un listado con los storage class junto a su velocidad de escritura que podemos utilizar.

```
ojovani@cloudshell:~$ kubectl get pvc
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
pvconce      Bound   pvc-4adc81ad-81a0-41e5-9fc3-845dba4f39df   20Gi       RWO            ibmc-file-silver 46h
```

Figura 35. PVC en IBM Cloud

Lo siguiente es crear un Deployment apuntando a nuestro PVC añadiendo el path predeterminado donde Nginx guarda el fichero html y posteriormente hay que crear el servicio para poder acceder a él.

```
ojovani@cloudshell:~$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
nginxpers    1/1     1             1           25h
php-apache   1/1     1             1           23h
web          3/3     3             3           8d
```

Figura 36. Deployment para PVC en IBM Cloud




```

ojovani@cloudshell:~$ kubectl get svc
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes          ClusterIP     172.21.0.1      <none>           443/TCP          9d
nginx-service       LoadBalancer 172.21.17.119   159.8.74.140    80:31853/TCP    25h
php-apache          ClusterIP     172.21.111.72   <none>           80/TCP           23h
web                 LoadBalancer 172.21.207.30   159.8.74.138    80:30892/TCP    8d

```

Figura 37. Servicio para PVC en IBM Cloud

Finalmente, lo que hemos hecho para utilizar el almacenamiento persistente es acceder al pod mediante `kubectl exec -it nginxpers-5bc6f86f97-z6k8j` y modificar el archivo `index.html` escribiendo algo diferente a lo predeterminado para asegurarnos de que los cambios que hemos realizado no se pierden más tarde.

En la siguiente imagen debemos fijarnos en el pod de `nginxpers`, para comprobar que el almacenamiento es realmente persistente, hemos hecho un `curl` a la ip pública del servicio para ver qué es lo que nos imprime por pantalla. Posteriormente borramos el pod de `nginxpers` para forzar a nuestro Deployment a satisfacer la necesidad de que siempre tenga que haber un pod activo como le hemos indicado anteriormente.

Con esto debería haber desaparecido lo que habíamos escrito anteriormente en el `index.html` ya que los pods son volátiles y no guardan datos, pero como hemos creado un PVC y lo hemos apuntado a ese path, por ello no se borra. Lo podemos comprobar en el siguiente `curl`, ya que imprime lo mismo por pantalla teniendo el pod una terminación diferente ya que se ha tenido que crear de nuevo después de haberlo borrado.

```

ojovani@cloudshell:~$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
nginxpers-5bc6f86f97-z6k8j   1/1     Running   0           24h
php-apache-7495ff8f5b-xcq52  1/1     Running   0           22h
web-6f786cbddb-692p6        1/1     Running   0           8d
web-6f786cbddb-8q8jh        1/1     Running   0           45h
web-6f786cbddb-wfv4w        1/1     Running   0           8d
ojovani@cloudshell:~$ curl 159.8.74.140
<html>
<head>
  <title>Persistent Storage</title>
</head>
<body>
  <h1>Persistent storage test</h1>
  <h2>Despliegue de un cluster kubernetes altamente disponible en IBM Cloud</h2>
  <p>Volumen persistente en Kubernetes mediante file storage en IBM CLOUD</p>
</body>
</html>
ojovani@cloudshell:~$ kubectl delete pod nginxpers-5bc6f86f97-z6k8j
pod "nginxpers-5bc6f86f97-z6k8j" deleted
ojovani@cloudshell:~$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
nginxpers-5bc6f86f97-2bvmmw   1/1     Running   0           28s
php-apache-7495ff8f5b-xcq52  1/1     Running   0           22h
web-6f786cbddb-692p6        1/1     Running   0           8d
web-6f786cbddb-8q8jh        1/1     Running   0           45h
web-6f786cbddb-wfv4w        1/1     Running   0           8d
ojovani@cloudshell:~$ curl 159.8.74.140
<html>
<head>
  <title>Persistent Storage</title>
</head>
<body>
  <h1>Persistent storage test</h1>
  <h2>Despliegue de un cluster kubernetes altamente disponible en IBM Cloud</h2>
  <p>Volumen persistente en Kubernetes mediante file storage en IBM CLOUD</p>
</body>
</html>

```

Figura 38. Test File Storage

7.4. Autoscalers en IBM Cloud

Un autoscaler es una herramienta que se utiliza en sistemas informáticos para ajustar automáticamente la cantidad de recursos como CPU, memoria, instancias y nodos, asignados a una aplicación o servicio en función de la carga de trabajo actual. El objetivo principal de un autoscaler es garantizar un rendimiento óptimo y una utilización eficiente de los recursos, evitando tanto la infrutilización como la sobreutilización.

En el contexto de la administración de clústeres y aplicaciones en Kubernetes, como mencioné anteriormente, hay dos tipos principales de autoscalers que vamos a ver a continuación.

7.4.1. HPA en IBM Cloud

El HPA como vimos anteriormente en el despliegue en local es una característica de Kubernetes que escala horizontalmente, agregando o eliminando réplicas de pods según sea necesario para mantener un equilibrio entre la carga de trabajo y los recursos disponibles.

Para probarlo en IBM, hemos hecho exactamente lo que hicimos en la prueba en local, ya que el HPA al ser una característica que ya viene directamente con Kubernetes, no es necesario cambiar ninguna configuración (Anexo 4). Solamente hay que asegurarse de tener instalado el metrics server, podemos hacerlo mediante el siguiente comando:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

En las pruebas como podemos ver, hemos vuelto a utilizar un pod con una imagen que se encarga de generar carga al pod con el php-apache.

```
ojovani@cloudshell:~$ kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Figura 39. Prueba HPA IBM Cloud



Y aquí vemos como según aumenta la carga, también se van aumentando las réplicas y más tarde al haber parado el load generator, vemos como decrecen de nuevo las réplicas para volver al estado inicial de 1 pod que es lo que marca el deployment que hemos realizado.

```

ojovani@cloudshell:~$ kubectl get hpa --watch

```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	1%/50%	1	10	1	24h
php-apache	Deployment/php-apache	165%/50%	1	10	1	24h
php-apache	Deployment/php-apache	165%/50%	1	10	4	24h
php-apache	Deployment/php-apache	173%/50%	1	10	4	24h
php-apache	Deployment/php-apache	84%/50%	1	10	4	24h
php-apache	Deployment/php-apache	84%/50%	1	10	6	24h
php-apache	Deployment/php-apache	86%/50%	1	10	6	24h
php-apache	Deployment/php-apache	84%/50%	1	10	6	24h
php-apache	Deployment/php-apache	85%/50%	1	10	6	24h
php-apache	Deployment/php-apache	81%/50%	1	10	6	24h
php-apache	Deployment/php-apache	85%/50%	1	10	6	24h
php-apache	Deployment/php-apache	50%/50%	1	10	6	24h
php-apache	Deployment/php-apache	0%/50%	1	10	6	24h
php-apache	Deployment/php-apache	0%/50%	1	10	6	24h
php-apache	Deployment/php-apache	1%/50%	1	10	1	24h

Figura 40. Escalado de pods en IBM Cloud

7.4.1. Cluster autoscaler

El cluster autoscaler en el contexto de IBM Cloud Kubernetes Service (IKS) se refiere a una característica que permite que nuestro clúster Kubernetes ajuste automáticamente el número de nodos en función de la carga de trabajo. Esta característica es esencial para optimizar los recursos y garantizar un rendimiento óptimo para tus aplicaciones en un entorno Kubernetes.

Se pueden configurar umbrales y políticas para el autoescalamiento en función de métricas como la utilización de CPU y memoria. Estas configuraciones determinan cuándo se deben agregar o eliminar nodos.

El sistema monitoriza constantemente las métricas de uso de recursos en los pods y el clúster. Si la demanda de recursos supera ciertos umbrales configurados, el autoescalador toma la decisión de agregar nodos al clúster para proporcionar más capacidad de procesamiento. Si la demanda disminuye y los nodos ya no son necesarios, el autoescalador puede eliminar nodos del clúster para ahorrar recursos.

El cluster autoscaler en IKS contribuye a la eficiencia operativa y a garantizar un rendimiento óptimo para nuestras aplicaciones, ya que asegura que siempre haya suficiente capacidad disponible para manejar las cargas de trabajo cambiantes.

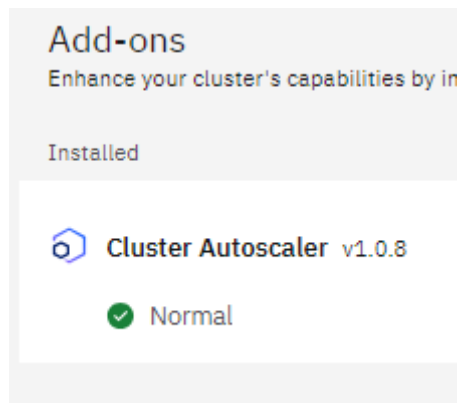


Figura 41. Cluster autoscaler addon

```
kubectl edit cm iks-ca-configmap -n kube-system -o yaml
```

Al utilizar el comando anterior podemos editar el yaml que se encarga de decidir cuál es el número mínimo y máximo de nodos disponibles en función de la carga

```
workerPoolsConfig.json: |
[
  {"name": "default","minSize": 1,"maxSize": 2,"enabled":false}
]
```

Figura 42. Configmap para cluster autoscaler

Esta herramienta hay que usarla con precaución ya que puede suponer costes adicionales a la hora de escalar más nodos.

7.5. Ingress en IBM Cloud

Ingress es un recurso de Kubernetes que nos permite exponer servicios HTTP y HTTPS en nuestro cluster al mundo exterior. Básicamente, un Ingress actúa como una capa de entrada que administra el enrutamiento del tráfico de entrada hacia los servicios correctos dentro del clúster, lo que simplifica la configuración y la administración de rutas y reglas de tráfico.

El recurso Ingress se compone de reglas que definen cómo se debe manejar el tráfico entrante. Estas reglas especifican patrones de rutas y hostnames para redirigir las solicitudes a los servicios correspondientes. Además, Ingress también puede manejar la configuración de SSL/TLS para conexiones seguras.

IBM Cloud nos ofrece su propio controlador de Ingress para facilitar la administración del enrutamiento. Se puede utilizar este controlador para crear reglas de Ingress y gestionar cómo se enruta el tráfico hacia nuestros servicios.

Para configurar el enrutamiento, se necesita crear recursos Ingress en nuestro clúster. Estos recursos contienen información sobre cómo se deben dirigir las solicitudes de entrada, como rutas, reglas de host, configuración SSL, etc..



Figura 43. Esquema de ingress

7.5.1. Tests con Ingress

Para probar el funcionamiento del controlador Ingress que nos ofrece IBM Cloud, hemos utilizado 2 deployments que imprimen por pantalla un texto y el nombre del pod que lo ejecuta, junto a ellos hemos creado 2 servicios para poder exponerlos al exterior. Todo esto lo podemos encontrar en el ANEXO.

Lo siguiente es crear un yaml ingress indicando que servicios queremos que apunten a nuestro punto de entrada ingress y el nombre que queremos usar para acceder a ellos junto al subdominio ingress que nos proporciona IBM Cloud, en nuestro caso sería:

clusterbueno-d83692433bb57a5097d1ceba0408cf98-0000.eu-de.containers.appdomain.cloud

Si hacemos un `kubectl get ing`, podemos observar las reglas que hemos creado anteriormente.

```
ojovani@cloudshell:~$ kubectl get ing
NAME          CLASS          HOSTS
ingress      public-iks-k8s-nginx  v1.clusterbueno-d83692433bb57a5097d1ceba0408cf98-0000.eu-de.containers.appdomain.cloud
,v2.clusterbueno-d83692433bb57a5097d1ceba0408cf98-0000.eu-de.containers.appdomain.cloud  159.8.74.139  80  28m
```

Figura 44. `kubectl get ing`

Podemos acceder a ambas URLs mediante un navegador para comprobar que funciona como vemos en las siguientes imagenes.

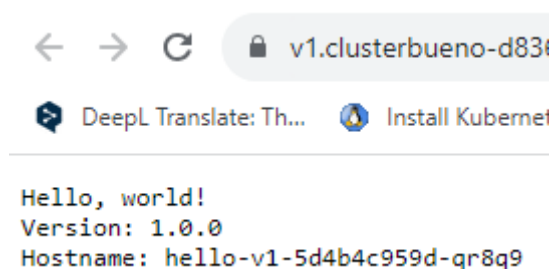


Figura 45. Hello app 1 HTML

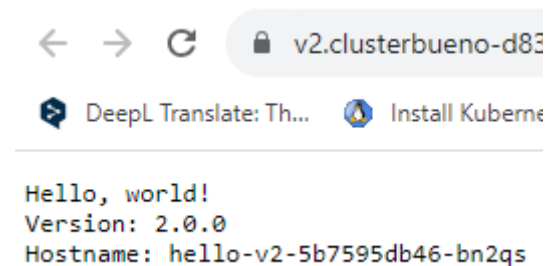


Figura 46. Hello app 2 HTML



8. Observability en IBM Cloud

La observabilidad se refiere a la capacidad de obtener información detallada y comprensible sobre el comportamiento, el rendimiento y el estado de nuestras aplicaciones y sistemas desplegados en la nube. Más allá de simplemente monitorear y recolectar datos, se trata de obtener una comprensión profunda de cómo funcionan nuestros sistemas en tiempo real y cómo se comportan en diferentes circunstancias.

En un entorno tan dinámico y complejo como la nube de IBM, la observabilidad se convierte en un aspecto esencial para garantizar la disponibilidad, la confiabilidad y el rendimiento óptimo de nuestras aplicaciones. Permite a los equipos de desarrollo, operaciones y mantenimiento identificar y resolver problemas rápidamente, tomar decisiones informadas y anticipar posibles desafíos.

También involucra la recopilación, el análisis y la interpretación de diferentes tipos de datos, como registros de aplicaciones, métricas de rendimiento, trazas de solicitudes, eventos y más.



Figura 47. Integrations en el cluster

8.1. Logging integration

El logging integration es una herramienta que nos proporciona IBM Cloud que nos permite integrar y gestionar logs generados por nuestros recursos y aplicaciones en el nuestro cluster. Los logs son información detallada sobre eventos, acciones y comportamientos de nuestras aplicaciones y sistemas, y son esenciales para el diagnóstico, seguimiento y solución de problemas.

La integración de logs en IKS permite capturar, almacenar y analizar los registros generados por los contenedores, las aplicaciones y los servicios en nuestros clústeres de Kubernetes. Esto es crucial para monitorear el rendimiento, detectar problemas y asegurarse de que nuestras aplicaciones estén funcionando de manera óptima. IBM Cloud ofrece una solución de gestión de registros llamada "IBM Cloud Monitoring with Sysdig", que puede utilizarse para este propósito.

Algunas características clave de la integración de registro en IKS incluyen:

Recolección centralizada: Los registros generados por los contenedores y las aplicaciones en múltiples nodos del clúster se recopilan y almacenan en un único lugar para una fácil accesibilidad.

Búsqueda y análisis avanzados: Puedes buscar y analizar los registros utilizando consultas avanzadas para identificar patrones, tendencias y problemas específicos.

Alertas y notificaciones: Puedes configurar alertas basadas en ciertos patrones de registro o condiciones específicas para recibir notificaciones cuando ocurran eventos importantes o problemas.

Visualización de datos: Las herramientas de gestión de registros te permiten visualizar los datos en forma de gráficos, tablas y paneles para comprender mejor el rendimiento y el comportamiento de tus aplicaciones.

Diagnóstico y resolución de problemas: Los registros son una fuente invaluable para el diagnóstico de problemas y la resolución de incidencias, ya que proporcionan información detallada sobre lo que está sucediendo dentro de tus aplicaciones y sistemas.

La integración de logs en IKS generalmente implica la instalación y configuración de agentes o herramientas específicas en nuestros clústeres Kubernetes para capturar y enviar los registros a una plataforma de gestión centralizada. IBM Cloud Monitoring with Sysdig es una solución popular para esta tarea, ya que proporciona capacidades avanzadas de monitoreo y análisis de registros.

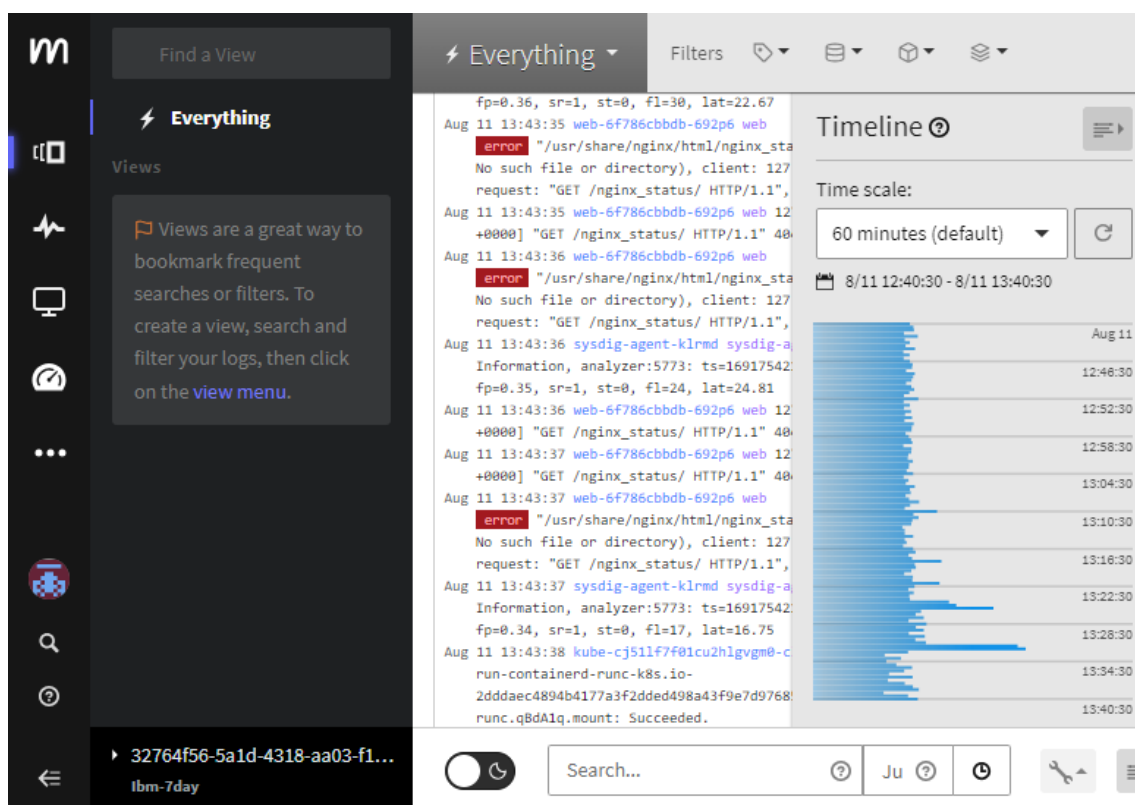


Figura 48. Logging integration



8.2. Monitoring integration

IBM Cloud Monitoring with Sysdig es una solución de monitoreo avanzada que se integra perfectamente con IKS. Esta herramienta está diseñada para proporcionar una visibilidad completa y profunda de nuestro cluster Kubernetes y las aplicaciones que se ejecutan en ellos. Nos permite monitorear y analizar métricas de rendimiento, registros de aplicaciones y trazas de solicitudes para garantizar la disponibilidad, el rendimiento y la confiabilidad de nuestros servicios en entornos de contenedores.

Esta herramienta nos permite principalmente:

Recopilación Integral de Datos: La herramienta recopila automáticamente métricas detalladas del sistema operativo, métricas de Kubernetes y métricas de aplicaciones desde todos los nodos y contenedores en tu clúster de IKS. Esto proporciona una imagen completa de la salud y el rendimiento de tus aplicaciones.

Análisis en Tiempo Real: Puedes analizar los datos de monitoreo en tiempo real para identificar patrones, tendencias y anomalías. Esto ayuda a detectar problemas y tomar medidas proactivas antes de que afecten la disponibilidad o el rendimiento.

Visualización Personalizada: IBM Cloud Monitoring with Sysdig ofrece un tablero intuitivo y personalizable con gráficos, paneles y visualizaciones para presentar los datos de manera clara y comprensible. Puedes crear paneles adaptados a tus necesidades para tener una vista rápida del estado de tus clústeres y aplicaciones.

Alertas Configurables: Puedes configurar alertas basadas en umbrales específicos o condiciones personalizadas. Cuando se alcanzan estos umbrales, la herramienta puede enviar notificaciones a través de diversos canales para informarte sobre eventos críticos.

Diagnóstico y Resolución de Problemas: La función de búsqueda avanzada te permite investigar incidentes específicos al navegar por los registros y las trazas de eventos. Esto facilita la identificación de las causas raíz y la resolución de problemas.

Tracing: La herramienta permite el seguimiento y análisis de las solicitudes a medida que fluyen a través de tus aplicaciones. Esto ayuda a identificar cuellos de botella y optimizar el rendimiento de tus microservicios.

Integración con Kubernetes: IBM Cloud Monitoring with Sysdig está especialmente diseñado para trabajar con entornos de Kubernetes.

En resumen, IBM Cloud Monitoring with Sysdig es una solución poderosa para monitorear y garantizar el rendimiento de nuestras aplicaciones en entornos Kubernetes, como IBM Kubernetes Service en la plataforma IBM Cloud. La herramienta nos ayuda a mantener la visibilidad, identificar problemas rápidamente y tomar decisiones informadas para garantizar una experiencia confiable para tus usuarios finales.

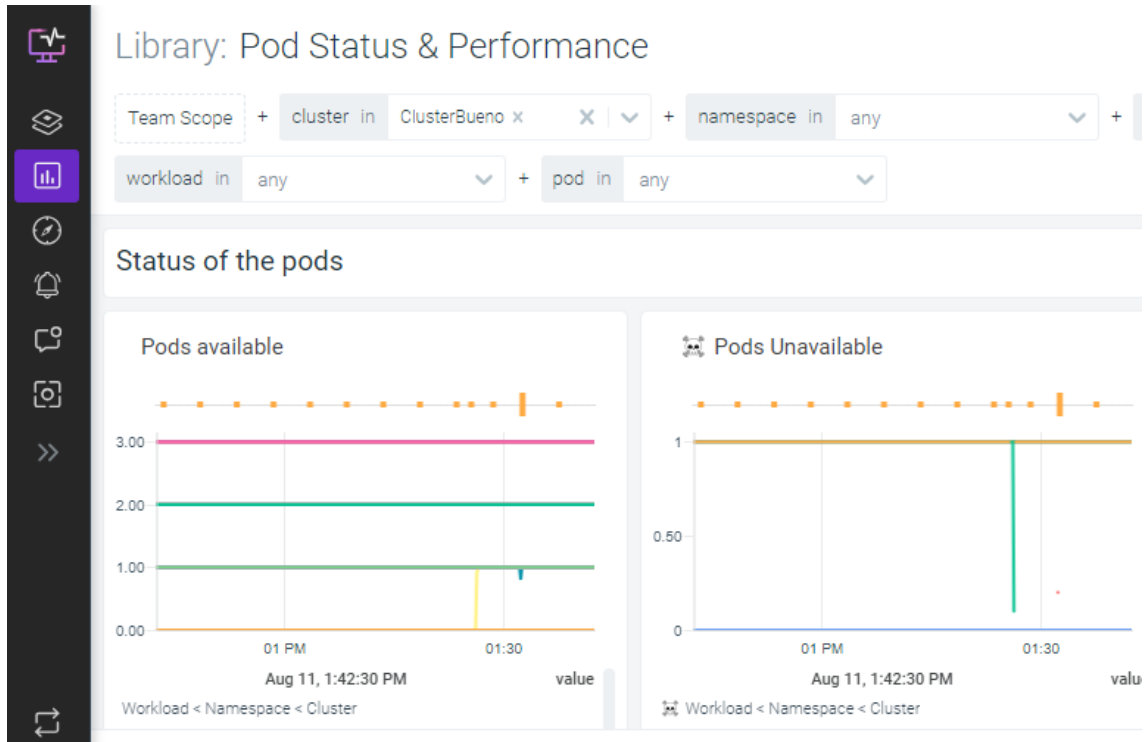


Figura 49. Monitoring integration

9. High availability

Para lograr alta disponibilidad en un clúster de Kubernetes, es necesario configurar el clúster de manera que esté diseñado para resistir fallos y mantener la disponibilidad de las aplicaciones en ejecución

IBM Cloud proporciona herramientas y servicios adicionales que nos pueden ayudar a lograr una alta disponibilidad, como Load Balancer, Multi-Zone Clusters y servicios de almacenamiento altamente disponibles.

Como ya hemos visto anteriormente el despliegue y funcionamiento de un Load Balancer y del almacenamiento persistente, vamos a centrar este apartado en hablar sobre los Multi-Zone Clusters.

Los clústeres de varias zonas son una solución avanzada para garantizar la alta disponibilidad y la resiliencia de las aplicaciones alojadas en la plataforma Kubernetes. Estos clústeres están diseñados para abordar los desafíos de la interrupción y los fallos en una sola zona de disponibilidad al distribuir los nodos del clúster en múltiples zonas geográficas dentro de una región de IBM Cloud.

En esencia, un clúster de varias zonas utiliza la capacidad de múltiples zonas de disponibilidad dentro de una región específica de IBM Cloud para garantizar que, en caso de que una zona experimente problemas o fallas, el clúster pueda seguir funcionando sin interrupciones en las zonas restantes.

La distribución de nodos del clúster en múltiples zonas geográficas asegura que incluso si una zona experimenta una falla o interrupción, la aplicación y el clúster en su conjunto puedan continuar funcionando en otras zonas, manteniendo así la disponibilidad. Al estar dispersos en diferentes zonas, los nodos del clúster son menos susceptibles a las fallas que podrían afectar una sola zona de disponibilidad.

La capacidad de distribuir el tráfico entre diferentes zonas puede mejorar el rendimiento de las aplicaciones al permitir que los usuarios se conecten al nodo más cercano en términos de latencia.

Al utilizar la capacidad de múltiples zonas, los clústeres de varias zonas pueden permitir actualizaciones de clúster y nodos sin interrumpir la disponibilidad de la aplicación.

Se pueden configurar balanceadores de carga para distribuir el tráfico entre las diferentes zonas, lo que contribuye a un uso más eficiente de los recursos y una experiencia del usuario más uniforme.

Todo esto se puede configurar de forma muy sencilla a la hora de desplegar nuestro clúster en la sección de Location.

Location

Choose your location and configure your VLANs. [Learn more about this.](#)

Resource group

Default

Geography: Europe

Availability: Multizone (High availability)

Metro: Frankfurt

Worker zones and VLANs ⓘ

- Frankfurt 02
VLANs will be created
- Frankfurt 04
VLANs will be created
- Frankfurt 05
VLANs will be created

Figura 50. Selección de Multiple location en cluster



También debemos recalcar que esta configuración nos va a suponer un incremento notable en el presupuesto de nuestro cluster Kubernetes, ya que al final lo que estamos haciendo es replicar nuestro cluster en 3 zonas distintas, por lo tanto disponemos del triple de nodos que con una configuración normal.

Por ello como vemos en la siguiente imagen tenemos 6 workers distribuidos en 3 zonas distintas, lo que nos supone un precio de 481.06\$/mes.

Summary Spain

Kubernetes cluster

- 6 Worker nodes** €0.64/hr
 - u3c.2x4 - 2 vCPUs 4GB RAM
 - Virtual - shared
 - x86-64
 - Ubuntu 20
- 1 Multizone load balancer** €0.02/hr
 - Multizone clusters require a cross-zone load balancer.

Usage-based charges
The charges for these services are based on actual usage after provisioning.

- Activity tracker** [View pricing](#)
 - Service: IBM Cloud Activity Tracker
 - Name: activity-frankfurt-NN
 - Location: Frankfurt
 - Plan: 7 day Event Search
- Logging** [View pricing](#)
 - Service: IBM Log Analysis
 - Name: logs-frankfurt-mx
 - Location: Frankfurt
 - Plan: 7 day Log Search
- Monitoring** [View pricing](#)
 - Service: IBM Cloud Monitoring
 - Name: metrics-frankfurt-N6
 - Location: Frankfurt
 - Plan: Graduated Tier

Total estimated cost €481.06/mo

Figura 51. Summary multi zone cluster

10. Conclusiones

En el transcurso de este trabajo de investigación, se ha explorado en profundidad la implementación de un clúster Kubernetes tanto en un entorno local utilizando máquinas virtuales como en un entorno cloud como viene a ser IBM Cloud que nos ha proporcionado todas las herramientas necesarias para abordar este proyecto.

El objetivo principal fue comprender cómo diseñar y desplegar una infraestructura que garantice la resiliencia, la escalabilidad y la disponibilidad continua de las aplicaciones alojadas en un entorno de contenedores.

Hemos aprendido a desplegar servicios en Kubernetes y a configurar y utilizar recursos que nos ayudan a conseguir la alta disponibilidad que buscamos en nuestro cluster. También hemos explorado todas las posibilidades que nos ofrece IBM Cloud en cuanto a Kubernetes y observability.

A lo largo del proyecto nos hemos enfrentado a múltiples problemas como las complicaciones que surgieron durante el primer despliegue con kubeadm, ya que había que aprender todo el entorno de Kubernetes desde cero, o los problemas que nos causó IBM Cloud a la hora de registrarse ya que solo acepta cuentas de empresa. Pero pese a todo ello se ha conseguido llevar el proyecto adelante.



11. Bibliografía

1. Install Kubernetes Cluster on Ubuntu 22.04 using kubectl. Disponible en: <https://computingforgeeks.com/install-kubernetes-cluster-ubuntu-jammy/>. Consultado en 05/23.
2. How to Setup Kubernetes Cluster with Kubectl on Ubuntu 22.04. Disponible en: <https://www.howtoforge.com/how-to-setup-kubernetes-cluster-with-kubectl-on-ubuntu-22-04/>. Consultado en 05/23.
3. Kubernetes Documentation. Disponible en: <https://kubernetes.io/docs/concepts/>. Consultado en 05/23.
4. Introduction to Kubernetes. Disponible en: <https://www.edx.org/learn/kubernetes/the-linux-foundation-introduction-to-kubernetes>. Consultado en 05/23.
5. Listado de videos de Kubernetes. Disponible en: https://www.youtube.com/watch?v=oTfOKxK1QNo&list=PLqRCtmokbeHA5ME_Anwu-vh4NWlgrOY. Consultado en: 06/23
6. Configure NFS as Kubernetes Persistent Volume Storage. Disponible en: https://computingforgeeks.com/configure-nfs-as-kubernetes-persistent-volume-storage/#google_vignette. Consultado en: 06/23.
7. Moreira Flors, Pablo. (2021). Metallb LB PV PVC NFS K8S ON PREMISES. Consultado en 06/23.
8. How To Install Metrics Server on a Kubernetes Cluster. Disponible en: <https://computingforgeeks.com/how-to-deploy-metrics-server-to-kubernetes-cluster/>. Consultado en: 07/23/
9. How To Deploy MetalLB Load Balancer on Kubernetes Cluster. Disponible en: <https://computingforgeeks.com/deploy-metallb-load-balancer-on-kubernetes/>. Consultado en: 07/23.
10. HorizontalPodAutoscaler Walkthrough. Disponible en: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>. Consultado en: 07/23.
11. IBM Cloud File Storage. Disponible en: <https://cloud.ibm.com/docs/FileStorage>. Consultado en 08/23.
12. Ingress in IBM Cloud. Disponible en: <https://cloud.ibm.com/docs/containers?topic=containers-managed-ingress-about>. Consultado en: 08/23.

12. ANEXO

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: kube-nfs-pv
spec:
  storageClassName: storage-nfs
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 10.0.2.13
    path: "/data/k8s"
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: kube-nfs-pvc
spec:
  storageClassName: storage-nfs
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

Anexo 1. PV y PVC local

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: trafex/php-nginx:latest
          ports:
            - containerPort: 8080
          volumeMounts:
            - name: kube-nfs-pvc
              mountPath: /var/www/html
              readOnly: true
      volumes:
        - name: kube-nfs-pvc
          persistentVolumeClaim:
            claimName: kube-nfs-pvc
```

Anexo 2. Deployment php-nginx persistente




```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: production
  namespace: metallb-system
spec:
  addresses:
  - 192.168.59.130-192.168.59.150
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2-advert
  namespace: metallb-system
```

Anexo 3. IPAddressPool y L2Advertisement

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

Anexo 4. Deployment y Service Php-Apache para el HPA

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvconce
  labels:
    billingType: hourly
    region:
    zone:
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 20Gi
  storageClassName: ibmc-file-silver
```

Anexo 5. PVC en IBM Cloud

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-v1
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-v1
  template:
    metadata:
      labels:
        app: hello-v1
    spec:
      containers:
      - name: hello
        image: gcr.io/google-samples/hello-app:1.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
```

Anexo 6. Hello App 1 deployment



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-v2
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hello-v2
  template:
    metadata:
      labels:
        app: hello-v2
    spec:
      containers:
        - name: hello
          image: gcr.io/google-samples/hello-app:2.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
```

Anexo 7. Hello App 2 deployment

```
kind: Service
apiVersion: v1
metadata:
  name: hello-v1-svc
spec:
  selector:
    app: hello-v1
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Anexo 8. Hello App 1 service

```
kind: Service
apiVersion: v1
metadata:
  name: hello-v2-svc
spec:
  selector:
    app: hello-v2
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Anexo 9. Hello App 2 service

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: public-iks-k8s-nginx
  rules:
    - host: v1.clusterbueno-d83692433bb57a5097d1ceba0408cf98-0000.eu-
      de.containers.appdomain.cloud
      http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: hello-v1-svc
                port:
                  number: 80

    - host: v2.clusterbueno-d83692433bb57a5097d1ceba0408cf98-0000.eu-
      de.containers.appdomain.cloud
      http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: hello-v2-svc
                port:
                  number: 80
```

Anexo 10. Ingress yaml



Glosario de términos

- **Cloud:** datos y programas que son accedidos a través de Internet en lugar de hacerlo directamente al disco duro de nuestro ordenador.
- **Cluster:** conjunto o agrupación de computadoras interconectadas que trabajan juntas como si fueran un único sistema.
- **Deployment:** es un objeto que puede representar una aplicación de nuestro cluster.
- **IBM Cloud:** plataforma cloud que ofrece al usuario infraestructura como servicio (IaaS).
- **Observability:** capacidad de comprender lo que sucede dentro de un sistema examinando los datos de salida.
- **Pod:** objetos más pequeños y básicos que se pueden implementar en Kubernetes.
- **Kubeadm:** utilidad que se encarga de la creación y el despliegue de clústeres de Kubernetes de una manera muy sencilla para el usuario.
- **Kubernetes:** popular sistema para orquestar contenedores.
- **Servicio:** es una abstracción que define un conjunto lógico de Pods y una política por la cual acceder a ellos.
- **Yaml:** es un lenguaje de serialización de datos que suele utilizarse en el diseño de archivos de configuración.



ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre cero.				X
ODS 3. Salud y bienestar.				X
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.	X			
ODS 9. Industria, innovación e infraestructuras.	X			
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.				X
ODS 12. Producción y consumo responsables.		X		
ODS 13. Acción por el clima.		X		
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones sólidas.				X
ODS 17. Alianzas para lograr objetivos.				X



Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Kubernetes es la plataforma de orquestación de contenedores por excelencia, ha desempeñado un papel fundamental en la transformación de la industria de la tecnología y en la forma en que las empresas gestionan sus aplicaciones y servicios, por ello podemos decir que ha tenido un impacto significativo en diferentes puntos de los ODS, principalmente relacionados con la eficiencia energética y la sostenibilidad. Entre ellos están los siguientes:

Trabajo Decente y Crecimiento Económico (ODS 8):

Kubernetes ha impulsado la creación de empleos relacionados con la gestión de contenedores y la infraestructura de nube. La demanda de profesionales de Kubernetes ha aumentado, lo que ha contribuido al crecimiento del empleo en el sector de la tecnología. Todo esto ha ayudado a impulsar la productividad, automatizando el despliegue y la gestión de aplicaciones, lo que reduce los tiempos de inactividad y aumenta la productividad de los trabajadores, permitiendo a las empresas ser más ágiles y eficientes en la entrega de aplicaciones, lo que a su vez ha contribuido al crecimiento económico y mejorar la competitividad.

Industria, Innovación e Infraestructuras (ODS 9):

Kubernetes es una parte integral de la infraestructura de tecnología de la información de muchas organizaciones. Ha fomentado la innovación al simplificar la implementación y escalabilidad de aplicaciones, lo que permite a las empresas adoptar nuevas tecnologías y enfoques de desarrollo más rápidamente. La facilidad de administración de Kubernetes ha preparado el camino para la adopción de la nube y ha permitido a las empresas modernizar sus infraestructuras.

Producción y Consumo Responsables (ODS 12):

Kubernetes permite una asignación más eficiente de recursos de computación, lo que reduce el desperdicio y promueve el consumo responsable de recursos en la nube y en centros de datos.

Además, al facilitar la implementación y escalabilidad de aplicaciones en contenedores, Kubernetes puede ayudar a las empresas a diseñar sus aplicaciones de manera que sean más eficientes en cuanto al consumo de recursos, lo que reduce su impacto ambiental



Acción por el Clima (ODS 13):

Promueve la eficiencia energética, al permitir una mejor gestión de la infraestructura de TI, lo que puede resultar en un uso más eficiente de la energía. Además, al facilitar la implementación y escalabilidad de aplicaciones en contenedores, Kubernetes puede ayudar a las empresas a diseñar sus aplicaciones de manera que sean más eficientes en cuanto al consumo de recursos, lo que reduce su impacto ambiental.

En resumen, Kubernetes desempeña un papel importante en el logro de varios de los Objetivos de Desarrollo Sostenible, ya que esta tecnología no solo impulsa la eficiencia empresarial y la innovación, sino que también puede contribuir a un mundo más sostenible al optimizar el uso de recursos y reducir el impacto ambiental de las operaciones empresariales.

