**APUNTES DE LA PRESENTACIÓN KUBEMAGIG**

Javier Ramírez Moral

ÍNDICE

[DIAPO 1: Presentación. 2](#_Toc137711481)

[DIAPO 2: Caso práctico y Objetivos. 2](#_Toc137711482)

[DIAPO 3: Explicación de las partes del proyecto. 3](#_Toc137711483)

[DIAPO 4: Primera Parte. 3](#_Toc137711484)

[DIAPO 5: **Componentes de Kubernetes** 4](#_Toc137711485)

[DIAPO 6: Componentes de Kubernetes 4](#_Toc137711486)

[DIAPO 7: **Herramientas de Kubernetes** 5](#_Toc137711487)

[DIAPO 8: Segunda Parte. 6](#_Toc137711488)

[DIAPO 9: MariaDB Deployment 6](#_Toc137711489)

[DIAPO 10: MariaDB Secret 6](#_Toc137711490)

[DIAPO 11: MariaDB Internal Service 7](#_Toc137711491)

[DIAPO 13: **Wordpress ConfigMap:** 7](#_Toc137711492)

[DIAPO 14: **Wordpress External Service** 7](#_Toc137711493)

[DIAPO 16: **Namespaces** 8](#_Toc137711494)

[DIAPO 17: **Volúmenes persistentes** 9](#_Toc137711495)

[DIAPO 18: **Gestión de recursos** 9](#_Toc137711496)

[DIAPO 19: Tercera Parte Azure Kubernetes Service. 10](#_Toc137711497)

[DIAPO 20: Integrar el Clúster1 10](#_Toc137711498)

[DIAPO 21: Integrar el Clúster1 10](#_Toc137711499)

[DIAPO 22: **Crear un nuevo clúster** 10](#_Toc137711500)

[DIAPO 23: **HELM** 11](#_Toc137711501)

[DIAPO 24: **Instalar HELM** 11](#_Toc137711502)

[DIAPO 25: **Desplegar HELM** 11](#_Toc137711503)

[DIAPO 26: **Desplegar HELM** 12](#_Toc137711504)

[DIAPO 28: **Cuarta parte** 12](#_Toc137711505)

[DIAPO 29: **Azure Monitor** 12](#_Toc137711506)

[DIAPO 30: **Prometheus** 12](#_Toc137711507)

[DIAPO 31: **Grafana & Prometheus** 13](#_Toc137711508)

[Aquí podemos ver el grupo de recursos que se ha creado para la administración de Prometheus. Ahora añadimos Azure Managed Grafana en el mismo grupo de recursos donde tenemos nuestro clúster. 13](#_Toc137711509)

[DIAPO 32: **Grafana & Prometheus** 13](#_Toc137711510)

[DIAPO 33: **Prometheus** 13](#_Toc137711511)

[DIAPO 34: **Grafana Inicio y Fuentes de datos** 13](#_Toc137711512)

[DIAPO 35: **Grafana Dashboards** 13](#_Toc137711513)

[DIAPO 36: **Grafana Dashboards** 14](#_Toc137711514)

[DIAPO 37: **Grafana alertas** 14](#_Toc137711515)

[DIAPO 38: **Grafana alertas** 14](#_Toc137711516)

[DIAPO 39: **Grafana alertas** 14](#_Toc137711517)

[DIAPO 40: **Grafana alertas** 14](#_Toc137711518)

[DIAPO 41: **Grafana alertas** 14](#_Toc137711519)

[DIAPO 42: **Grafana alertas** 15](#_Toc137711520)

[DIAPO 43: **Grafana alertas** 15](#_Toc137711521)

[DIAPO 44: **Container Insights** 15](#_Toc137711522)

[DIAPO 45: **Container Insights Reglas de alertas** 15](#_Toc137711523)

[DIAPO 46: **Container Insights Métricas** 15](#_Toc137711524)

[DIAPO 47: **Container Insights Registros** 15](#_Toc137711525)

# DIAPO 1: Presentación.

Buenos días, soy Javier Ramírez Moral desarrollador de la empresa ELFHEIN y vengo a explicar en qué consiste el proyecto Kube-Magic desarrollado para NeoSoul.

Debido al tiempo que tengo para exponer, no podré abarcar todo lo que he hecho en el proyecto, en caso de que alguien quiera saber un poco más en profundidad de cualquier cosa del proyecto, al final de este con mucho gusto lo trataremos. Os he compartido al correo de todo un link a la página web del proyecto para que el que quiera pueda seguir la presentación y consultar los documentos del proyecto.

# DIAPO 2: Caso práctico y Objetivos.

**Caso práctico:**

Neo Soul es una empresa de entretenimiento y ha contactado conmigo explicándome lo siguiente. Ellos quieren crear su página web para su tienda en WordPress y a la vez quieren almacenar todos los pedidos que se realizan, la gente que se ha registrado, qué tipo de productos se han comprado etc. En una base de datos. Para ello me han pedido que les haga el despliegue tanto de la aplicación web en línea como de la base de datos y que las conecte entre sí. Esta página web esta más orientada para sus productos de ropa y merchandising.

Posteriormente, quieren también otra página y con otra base de datos y que esta sea la página global de la empresa, donde usuarios de todo el mundo accedan y de esta se deriven al resto de páginas enfocadas a los productos o servicios concretos que ellos ofrecen, como por ejemplo la del merchandising que mencioné antes y que me han encargado desplegar también.

**Objetivos:**

* **1**
* **2**
* **3**
* **4**

# DIAPO 3: Explicación de las partes del proyecto.

Ahora os voy a mostrar una visión general de las distintas fases del proyecto y según vayamos llegando a ellas las explicaré más en profundidad y en detalle. De momento os las expongo para ya tener una primera idea de cómo se ha llevado a cabo el proyecto.

* En la primera parte pretendo explicar que es Kubernetes, sus componentes elementales, las herramientas que he usado para desollar el proyecto y el funcionamiento de su arquitectura para entender las bases básicas necesarias del proyecto.
* En la segunda parte pretendo llevar a cabo el primer set up que necesitamos en el que desplegaré dos aplicaciones, nuestro sitio web y su base de datos dentro de nuestro primer clúster. Veremos que componentes he usado para ello y su funcionamiento interno.
* En la tercera parte nos serviremos de Azure Kubernetes Service para llevar a cabo la integración del primer clúster además de implantar el segundo escenario requerido, creando el segundo clúster desde el portal de Azure y desplegando en las otras dos aplicaciones necesarias, esta vez usando una herramienta llamada HELM.
* En la cuarta y última parte implementaremos las aplicaciones de monitoreo Prometheus y Grafana y las combinaremos para monitorear, establecer alertas y métricas a nuestros clústeres, así como también usaremos Azure Monitor.

# DIAPO 4: Primera Parte.

**¿Qué es Kubernetes?**

Kubernetes es una plataforma de código abierto que se utiliza para administrar y orquestar aplicaciones dentro de contenedores. Kubernetes te ayuda a gestionar y organizar estos contenedores como puede ser Docker de manera eficiente y que pueden estar en distintos entornos ya sea en físico, virtual o en la nube.

Piensa en Kubernetes como un controlador de tráfico para tus aplicaciones en contenedores.

**Características de Kubernetes:**

* Kubernetes facilita la escalabilidad automática de las aplicaciones en función de la demanda. \*ESCALABILIDAD DE KUBERNETES\*
* Kubernetes se asegura de que tus aplicaciones estén siempre disponibles. Si un contenedor falla, Kubernetes reiniciará automáticamente el contenedor o lo reemplazará con uno nuevo para que tu aplicación siga funcionando sin interrupciones. \*REPARACIÓN AUTOMÁTICA DE POD\*
* Puedes realizar actualizaciones en tus aplicaciones sin detenerlas.
* Se puede mover y desplegar nuestras aplicaciones en diversas infraestructuras como en on-premises, híbrida o cualquier tipo de nube.

**Utilidad de Kubernetes:**

He decidió usar Kubernetes, por tanto, debido a que es fundamental tener un entorno escalable, adaptativo y dinámico para poder soportar el tráfico de ambas aplicaciones. un clúster robusto, monitoreo del estado de nuestras aplicaciones, así como implementar todo en la nube lo que hace que la experiencia del usuario sea fluida que al final es lo que importa.

# DIAPO 5: **Componentes de Kubernetes**

**Clúster**: Un clúster de Kubernetes es un conjunto de uno o más nodos que ejecutan el software de Kubernetes y trabajan juntos como una sola entidad para ejecutar y administrar aplicaciones en contenedores.

**Nodo**: Los nodos son responsables de ejecutar y administrar los contenedores y los recursos asociados en el clúster de Kubernetes. El controlador maestro de Kubernetes programa los pods en los nodos disponibles según los requisitos de recursos y la política de programación definida. Dentro de un nodo puede haber uno o varios pods, cada nodo en el clúster es una máquina virtual o física que ejecuta un sistema operativo y el software de Kubernetes, y está conectado a una red que permite la comunicación entre los nodos.

**Pod**: En Kubernetes, un pod es la unidad básica de despliegue y ejecución de contenedores. Puede tener una o varias aplicaciones en contenedores que se ejecutan en el clúster estrechamente acoplados que comparten el mismo espacio de red, el mismo espacio de almacenamiento y otros recursos. Estos contenedores dentro del pod se ejecutan en el mismo nodo y se pueden comunicar entre sí a través de la dirección IP de cada pod. \*QUÉ PASA CUANDO CREAMOS UN POD\*

**Contenedor**: En Kubernetes, un contenedor es una unidad de ejecución aislada y ligera que encapsula una aplicación y todas sus dependencias, incluidas las bibliotecas, los binarios y los archivos de configuración necesarios para su funcionamiento. Los contenedores son creados a partir de imágenes, que son plantillas estáticas que contienen todo lo necesario para ejecutar una aplicación.

# DIAPO 6: Componentes de Kubernetes

**Control Plane:** El plano de control es responsable de la orquestación de contenedores y de mantener el estado deseado del clúster. Tiene los siguientes componentes.

* Kube-apiserver: El servidor de la API es el componente del plano de control de Kubernetes que expone la API de Kubernetes. Se trata del único punto de entrada a nuestro clúster desde el exterior, recibe las peticiones de configuración declaradas que hagamos y actualiza acordemente el estado en etcd. Además, también es el responsable del mecanismo de autentificación y autorización cuando queremos interactuar con el clúster, Todos los componentes del clúster se comunican con él. Coordina todos los procesos entre el plano de control y los Worker nodes.
* Control manager: observa el estado de nuestro clúster a través de la API y cuando se recibe la notificación de algún cambio se encarga de realizarlos siempre buscando que el estado actual de nuestras aplicaciones sea la misma que nosotros le especificamos (estado actual= estado deseado).
* Cloud controller mánager: Su función principal, es integrar y exponer las características específicas de la nube en la que se ejecuta el clúster de Kubernetes.
* Etcd: es una base de datos que almacena todas las configuraciones, estados e información de los objetos de Kubernetes en todo momento.
* Kube-scheduler: Componente que está pendiente de los Pods que no tienen ningún nodo trabajador asignado y selecciona uno donde ejecutarlo en los nodos trabajadores en función de los recursos disponibles y las políticas de asignación.

**Worker nodes:** Los nodos de trabajo son responsables de ejecutar aplicaciones en contenedores. El control plane de Kubernetes es responsable de orquestar la ejecución de los pods en los nodos de trabajo.

* Kubelet: Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un pod.
* Kube-proxy: Es responsable de la gestión del enrutamiento de red dentro del clúster. Kube-proxy proporciona servicios de red para los pods y realiza el balanceo de carga entre los servicios dentro del clúster.
* Container runtime: es el software responsable de ejecutar los contenedores que en mi caso será Docker. se ejecuta dentro de un pod y que se encapsula una aplicación y todas sus dependencias y configuraciones en un entorno aislado y portátil.

<https://devopscube.com/kubernetes-architecture-explained/>

# DIAPO 7: **Herramientas de Kubernetes**

* Minikube: Nos permite usar Kubernetes en nuestra máquina local creando clústeres de Kubernetes en un entorno aislado.
* Kubectl: Usa la herramienta de línea de comandos de Kubernetes, con ella vamos a interactuar con el clúster, inspeccionando recursos del clúster; crear, eliminar, y actualizar componentes; explorar tu nuevo clúster y arrancar aplicaciones.
* Kubeconfig: es utilizado por la herramienta de línea de comandos de Kubernetes, kubectl, para autenticar y autorizar los usuarios que realizan operaciones en los clústeres. contiene información sobre el clúster como las credenciales o certificado del servidor.
* Docker: Ambas son tecnologías contenedoras y de código abierto. Su principal diferencia es en el papel que juega cada una a la hora de distribuir las aplicaciones en contenedores. Las desarrolladoras usan Docker para crear y administrar imágenes de contenedores. Y Kubernetes es usado administrar a gran escala y coordinar estos contenedores.

# DIAPO 8: Segunda Parte.

Lo que pretendemos conseguir es lo siguiente en esta parte. Voy a desplegar dos aplicaciones que serán la aplicación de Wordpress y su base de datos MariaDB.

Sería algo así:

1. La petición llega del navegador.
2. Llega al External service del Wordpress.
3. Se envía así al pod de Wordpress.
4. El pod de WordPress se conecta al internal service de MariaDB.
5. Y llega al pod de MariaDB donde se autentifica con las credenciales.

# DIAPO 9: MariaDB Deployment

Un Deployment en Kubernetes es la herramienta declarativa principal para crear y configurar componentes en nuestro clúster define el estado deseado de la aplicación.

* Si seguimos bajando tenemos esta parte que es la de especificación. Es la sección donde se especifica el comportamiento deseado del Deployment. ReplicaSet es un objeto que se utiliza para garantizar el número especificado de réplicas de pods que deben de estar en funcionamiento. Una réplica es una copia exacta de un conjunto de Pods que están configurados y administrados de manera idéntica. El propósito de usar replicas es proporcionar alta disponibilidad y capacidad de escalado horizontal a las aplicaciones que se ejecutan en Kubernetes. Si por alguna razón un Pod falla o se elimina, Kubernetes automáticamente iniciará un nuevo Pod para mantener el número de réplicas deseado. Además, si se desea aumentar la capacidad de una aplicación, tener mayor tolerancia a fallos, capacidad procesamiento se puede aumentar el número de réplicas en el Deployment, y Kubernetes creará automáticamente los Pods adicionales

# DIAPO 10: MariaDB Secret

Los secrets se utilizan para garantizar la seguridad y protección de la información sensible que necesita ser utilizada por las aplicaciones desplegadas en el clúster de Kubernetes.

Por tanto, yo lo he usado para almacenar las credenciales de la base de datos y he codificado los registros. De tal forma que me permite tener más seguridad a la hora de controlar quien puede acceder y quien no mediante la autentificación.

Los datos de los secrets se pueden exponer como variables de entorno en los pods. De esta manera, las aplicaciones pueden acceder a los valores de los secrets a través estas y permiten configurar la aplicación de manera dinámica y personalizada en función del entorno en el que se esté ejecutando la aplicación.

# DIAPO 11: MariaDB Internal Service

Un Internal Service en Kubernetes es un tipo de servicio que se utiliza para exponer los pods dentro del clúster de Kubernetes. los servicios internos solo son accesibles desde dentro del clúster. Los servicios internos se crean con una dirección IP virtual (ClusterIP), que se asigna a un conjunto de pods que tienen las mismas etiquetas.

Que viene a ser lo que hace posible la comunicación entre MariaDB y Wordpress en nuestro clúster. ¿Por qué uso un service?, debido a que como ya mencionamos los pods son efímeros y fácilmente mueren, esto haría que Kubernetes al reemplazarlo por otro nuevo también le asignara una nueva dirección IP, lo que haría que tuviésemos que estar reconfigurando cada vez que suceda. En cambio, al usar un service, este asigna una dirección IP permanente a nuestro pod a pesar de que este muera. Esto es debido a que el ciclo de vida del service no está relacionado con el del pod. Evitando el problema a la hora de comunicar nuestras aplicaciones.

# DIAPO 13: **Wordpress ConfigMap:**

ConfigMap no proporciona encriptación. Si los datos que quieres almacenar son confidenciales, utiliza un-Secret en lugar de un ConfigMap, los ConfigMap pueden contener información para que otros elementos del sistema utilicen para su configuración.

Imaginemos que service de MariaDB cambiase, tendríamos que recargar la URL de la propia aplicación, teniendo que reconstruir la aplicación con una nueva versión volver a ejecutar una nueva imagen en el pod todo ello sería demasiado tedioso por este pequeño cambio. De esta manera me ahorro todo esto y conecto la información que contiene el ConfigMap con el Deployment a través de las variables de entorno y que así pueda leer a información y configuración definida que está en un fichero de configuración distinto. Esto hace que la aplicación sea más flexible y fácil de mantener, ya que no hay necesidad de actualizar el archivo de configuración de la aplicación cada vez que se cambia la información de conexión a la base de datos.

# DIAPO 14: **Wordpress External Service**

Es un tipo de servicio que expone una aplicación o servicio que se ejecuta dentro del clúster de Kubernetes a través de una dirección IP externa y un puerto. A diferencia de los servicios internos (Internal Services), los servicios externos son accesibles desde fuera del clúster de Kubernetes, lo que permite el acceso desde otras redes o sistemas externos.

A través de una dirección IP externa y el puerto asignado

Imagina que tu sitio web tiene miles de clientes diferentes por minuto. No es fácil para un solo servidor mantenerse al día con la demanda de solicitudes. Esto crea una mala experiencia de usuario con tiempos de inactividad impredecibles.

El servicio Loadbalancer proporciona capacidades de equilibrio de carga para que el tráfico entrante se distribuya uniformemente entre varios pods para un mejor rendimiento y una alta disponibilidad

* ContainerPort: Es el puerto de del contenedor dónde está Wordpress.
* Port: Es el puerto del Service que se expone internamente dentro del clúster, es decir, otros recursos de Kubernetes dentro del mismo clúster lo usarán para acceder al servicio
* TargetPort: Es el puerto de los pods donde están escuchando para recibir el tráfico.

Conviene que sea el mismo que el del container port.

* NodePort: Es el puerto por donde vamos a buscar junto con la IP de nuestra aplicación en el navegador y por donde van a llegar las peticiones y el tráfico externo. Es el puerto por el que exponemos el servicio y redirige este tráfico al target port.

# DIAPO 16: **Namespaces**

Los Namespaces en Kubernetes son una forma de dividir un clúster en Kubernetes de manera lógica en múltiples clústeres virtuales más pequeños y aislados lógicamente. Cada namespace actúa como entorno de trabajo separado en el que los objetos de Kubernetes, como los pods, los servicios y los volúmenes, pueden ser creados y gestionados.

Nos proporciona aislamiento y seguridad, escalabilidad de los recursos de forma independiente u organizar.

Estos son el espacio de nombres predeterminados:

* Default: el espacio de nombres para los objetos que no tienen otro espacio de nombres.
* Kube-node-lease: de nodos permiten que el kubelet envíe latidos al plano de control, lo que le permite detectar fallas en los nodos.
* Kube-public: este espacio de nombres se genera automáticamente y todos los usuarios (incluidos los que no están autenticados) pueden acceder a él. Este espacio de nombres se reserva principalmente para el uso del clúster, en caso de que algunos recursos deban ser visibles públicamente.
* Kube-system: el espacio de nombres para los objetos creados por Kubernetes

\*RBAC\*

# DIAPO 17: **Volúmenes persistentes**

Debido a que nuestros Pods, son efímeros, hay que otorgarles algún tipo de almacenamiento, para que la información que se despliegue no se pierda. Para ello vamos a hacer uso de los volúmenes, con el objetivo principal de mantener nuestra información incluso cuando nuestros pods desaparezcan.

**PersistentVolume:**

* Un PersistentVolume es un recurso de Kubernetes que representa una unidad de almacenamiento persistente en el clúster.
* Los PV se definen en el nivel de clúster y se pueden utilizar por diferentes pods en diferentes espacios de nombres.

**PersistentVolumeClaim:**

* Un PersistentVolumeClaim es una petición de almacenaje por un usuario.
* El PVC es creado por el usuario y especifica los requisitos de almacenamiento necesarios, como tamaño, modo de acceso y clase de almacenamiento.
* El PVC actúa como un intermediario entre el pod y el PV, solicitando un volumen persistente que cumpla con los requisitos definidos.
* Un PVC solo puede estar vinculado a un único PV a la vez.

**Pod:**

* Un pod puede solicitar un volumen persistente a través de un PVC para almacenar y acceder a los datos de manera persistente.
* Cuando un pod incluye una especificación de volumen (por ejemplo, a través de un PVC), Kubernetes se asegura de que el volumen esté montado en los contenedores dentro del pod.
* Cuando el pod solicita un volumen persistente, Kubernetes monta el volumen en el contenedor del pod como un sistema de archivos normal, y el pod puede leer y escribir datos en el volumen como si fuera un disco local.

En resumen, un volumen persistente (PV) es el recurso de almacenamiento persistente en el clúster, el PersistentVolumeClaim (PVC) es una solicitud para utilizar un volumen persistente y el pod es la unidad de despliegue que puede solicitar y utilizar un volumen persistente a través de un PVC. El PVC actúa como una capa de abstracción entre el pod y el PV, permitiendo solicitar y utilizar el almacenamiento persistente de manera independiente del pod. Esto proporciona una forma flexible de gestionar el almacenamiento persistente para los pods en Kubernetes.

# DIAPO 18: **Gestión de recursos**

En Kubernetes, la gestión de recursos se refiere a la administración y asignación de los recursos computacionales, como CPU y memoria, dentro de un clúster de Kubernetes.

Cada pod requiere una cierta cantidad de recursos para su correcto funcionamiento, como CPU y memoria.

Existen dos aspectos principales en la gestión de recursos en Kubernetes:

* Asignación de recursos: Kubernetes se encarga de asignar los recursos necesarios a cada pod, asegurándose de que los nodos del clúster tengan suficiente capacidad para ejecutarlos.
* Límites de recursos: también es posible establecer límites máximos de recursos para cada pod. Estos límites garantizan que un pod no consuma más recursos de los asignados, lo que evita que una aplicación acapare los recursos del clúster y afecte el rendimiento de otras aplicaciones. Si un pod supera los límites establecidos, Kubernetes puede tomar acciones como reiniciar o terminar el pod.

# DIAPO 19: Tercera Parte Azure Kubernetes Service.

Azure Kubernetes Service (AKS) es un servicio de orquestación de contenedores completamente administrado que se ejecuta en Microsoft Azure.

En esta parte lo que pretendo es integrar el clúster donde tenemos el primer escenario ya creado para poder administrarlo de manera más eficiente desde Azure. Y posteriormente montar el segundo set up creando un nuevo clúster con dos aplicaciones desplegadas en el también para la segunda página web con su correspondiente base de datos. Esta vez usando la herramienta de HELM.

# DIAPO 21: Integrar el Clúster1

Kubernetes habilitado para Azure Arc permite asociar clústeres de Kubernetes que se ejecutan en cualquier lugar para que pueda administrarlos y configurarlos en Azure.

Los agentes de Azure Arc se instalan en los recursos que deseas administrar desde Azure, y actúan como intermediarios entre Azure y los recursos. Al instalar el agente de Azure Arc en un recurso, se habilita la conexión y la comunicación bidireccional entre Azure y ese recurso, lo que permite que Azure proporcione servicios de administración, monitoreo, seguridad y cumplimiento normativo para el recurso. Esto permite que los equipos de desarrollo se enfoquen en crear aplicaciones en lugar de preocuparse por la infraestructura subyacente.

Una vez conectado a Azure, el clúster se representará como su propio recurso en Azure Resource Manager y se puede organizar mediante grupos de recursos y etiquetado.

Token de sesión para ver los recursos.

# DIAPO 22: **Crear un nuevo clúster**

Azure se va a encargar de administrar el nodo maestro de forma predeterminada. Lo que nos va a quitar carga administrativa en principio. Nosotros solo tenemos que aprovisionar la cantidad de nodos y especificaciones físicas.

\*ACTUALIZAR Y ESCALAR UN CLUSTER\*

# DIAPO 23: **HELM**

Helm es una herramienta de administración de paquetes para Kubernetes. Proporciona una forma fácil de definir, instalar y actualizar aplicaciones y recursos en un clúster de Kubernetes además de reutilizar las configuraciones. lo que simplifica el proceso de implementación y mantenimiento de aplicaciones en Kubernetes.

A través de Helm, puedes crear y distribuir "charts" (gráficos) que contienen los recursos y configuraciones necesarios para desplegar una aplicación en Kubernetes, vienen empaquetados en ellos. Un chart es una colección de archivos YAML que define los pods, servicios, volúmenes, configuraciones y otros recursos necesarios para una aplicación específica. También puede incluir plantillas que permiten la personalización y la configuración flexible de la aplicación.

* **Charts:** podemos agregar la estructura de otro gráfico dentro de este directorio si nuestros gráficos principales tienen alguna dependencia de otros. Por defecto este directorio está vacío.
* **Charts.yaml:** contiene información sobre el gráfico de timón como versión, nombre, descripción, etc.
* **Templates:** Este directorio contiene los archivos de plantilla de Kubernetes que definen los recursos que se desplegarán.
* **Values.yaml:** este archivo contiene los valores predeterminados de las variables de configuración de la aplicación. <https://helm.sh/docs/chart_template_guide/getting_started/>

# DIAPO 28: **Cuarta parte**

En esta parte vamos a añadir y configurar el monitoreo de nuestro escenario usando las siguientes herramientas que iremos viendo a continuación. Para asegurarnos de que todo funciona correctamente en nuestros escenarios.

<https://learn.microsoft.com/es-es/azure/azure-monitor/essentials/azure-monitor-workspace-manage?tabs=azure-portal#link-a-grafana-workspace>

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

# DIAPO 29: **Azure Monitor**

Azure Monitor es un servicio de monitoreo y diagnóstico de aplicaciones y recursos en la nube de Microsoft Azure. En el contexto de Azure Kubernetes Service (AKS), Azure Monitor es una solución que nos permite monitorear los clústeres de Kubernetes administrados en AKS.

Prometheus es un sistema de monitoreo y alerta de código abierto que se utiliza para recopilar y almacenar métricas de diferentes componentes de un clúster de Kubernetes, como pods, contenedores, nodos, servicios y redes.

Grafana, por su parte, es una herramienta de visualización y análisis de datos que permite a los usuarios crear paneles y gráficos personalizados a partir de las métricas recopiladas por Prometheus.

Juntos, Prometheus y Grafana ofrecen una solución completa de monitoreo y visualización de métricas para clústeres de Kubernetes. Podemos configurar Prometheus para recopilar y almacenar las métricas, y luego utilizar Grafana para crear paneles y gráficos personalizados para visualizar esas métricas.

# DIAPO 30: **Prometheus**

El único requisito para habilitar el servicio administrativo de Azure monitor para Prometheus es crear un área de trabajo de Azure Monitor que será donde se almacenen las métricas de Prometheus.

# DIAPO 31: **Grafana & Prometheus**

# Aquí podemos ver el grupo de recursos que se ha creado para la administración de Prometheus. Ahora añadimos Azure Managed Grafana en el mismo grupo de recursos donde tenemos nuestro clúster.

# DIAPO 32: **Grafana & Prometheus**

Ahora vamos a conectar Grafana con Prometheus

Podemos ver las áreas vinculas de Grafana, al vincular un área de trabajo de Azure Monitor permitimos que Grafana use los datos de esa área de trabajo con sus paneles

Aquí vemos que clústeres se pueden supervisar y le damos a configurar el clúster2

# DIAPO 33: **Prometheus**

Habilitamos así Prometheus y también el área de Grafana donde podremos ver las métricas de manera visual

# DIAPO 34: **Grafana Inicio y Fuentes de datos**

En Grafana, un origen de es una fuente de datos que se utiliza para conectarse a un sistema de almacenamiento de datos y recuperar los datos que se desean visualizar en los paneles y tableros.

Una vez configurado el origen de datos, se puede crear paneles y tableros que utilicen ese origen de datos para visualizar los datos recuperados del sistema de almacenamiento de datos.

Como vemos tenemos a Azure monitor y a Prometheus. Al tener a Grafana integrado ya en Azure Monitor nos permite maximizar la disponibilidad y el redimiendo de nuestras aplicaciones y servicios.

# DIAPO 35: **Grafana Dashboards**

Un tablero en Grafana es una visualización personalizada de datos en tiempo real y de series temporales que se pueden configurar y compartir con otros usuarios.

Los tableros en Grafana pueden mostrar datos de varias fuentes, como bases de datos, servicios de monitorización, sistemas de métricas y otros orígenes de datos

Grafana ofrece una amplia gama de opciones de personalización para los tableros, lo que permite a los usuarios ajustar el diseño, los colores, la frecuencia de actualización y otros aspectos para que se ajusten a sus necesidades específicas. Además, los tableros pueden compartirse con otros usuarios y pueden incrustarse en otras aplicaciones y sitios web para facilitar el acceso y la visualización de los datos.

Podemos por ejemplo también ver la información de nuestros nodos y pods dentro del clúster.

# DIAPO 36: **Grafana Dashboards**

Podemos crear nuestros propios panales, editar los que ya existen, importar otros desde la página oficial

# DIAPO 37: **Grafana alertas**

Vamos ahora con el tema de las alertas y reglas que es una de las cosas más interesantes que hay en Grafana y que van a ser esenciales en nuestro objetivo de monitorear.

Las alertas nos van a permitir identificar problemas, es un conjunto de criterios que cuentan con una o más expresiones y consultas, condiciones, la frecuencia de avaluación entre otros parámetros.

Esta es la sección de alertas de Grafana donde podremos crear las alertas que queramos y estas son sus diferentes secciones que tenemos.

En esta sección de puntos de contactos tenemos los destinatarios de las alarmas que se van generando por Grafana. Se usan para enviar notificaciones a nosotros como usuarios u otros sistemas cuando se dan las condiciones de las alertas.

# DIAPO 38: **Grafana alertas**

Lo primero que vamos a hacer es irnos a la sección de alertas y nos vamos a crear un canal de notificación para las alertas que iremos creando. Será la vía por la que estaremos enviando nuestra alerta. Nosotros lo haremos por medio de Teams. Entonces nos vamos a Teams y tenemos que crear un canal y dentro otro canal por donde Grafana va a estar hablando.

He decidido hacerlo así ya que todo nuestro equipo usa Teams para comunicarse. Por lo que este grupo tendremos acceso solo los administradores y podremos ver todos en cualquier momento cuando ocurre una alerta y atenderla de manera eficaz y rápida.

Crear un conector WEBHOOK.

# DIAPO 39: **Grafana alertas**

Volvemos a Grafana y creamos el canal de comunicación. Le asignamos nombre y tipo y pegamos en la sección de URL la que nos dio Teams.

# DIAPO 40: **Grafana alertas**

Para sacarle realmente el jugo necesitamos tener las alertas asociada a paneles porque las alertas van atadas a los paneles de cada Dashboards y las alertas necesitan tener consultas que no estén atadas a ninguna variable de otros panales, nosotros la hemos creado en el panel de uso de CPU.

Por ejemplo, para este panel le doy a editar y en la zona de alertas le creo una.

# DIAPO 41: **Grafana alertas**

Aquí le ponemos un nombre y a continuación elegimos un tipo de regla. Las reglas administradas por Grafana son el tipo de regla de alerta más flexible. Nos permiten crear alertas que pueden actuar sobre los datos de cualquiera de nuestras fuentes de datos admitidas. Este es el único tipo de regla que permite alertar desde múltiples fuentes de datos en una sola definición de regla. Por eso lo he elegido.

Ahora elegimos el comportamiento de la regla, como el tiempo de evaluación, el recurso y grupo y del comportamiento que debe tener en determinados casos.

*Yo voy a establecer una regla para que me vigile el uso de CPU de mi clúster de cada namespace que tiene. Todo ello lo recogerá de la fuente de datos de Prometheus*

# DIAPO 42: **Grafana alertas**

Ahora está en un estado de pendiente, eso es que está evaluando a ver si tiene que enviar la alerta y a lo que se representa con el color amarillo. Finalmente se dispara y podemos verla más en detalle.

# DIAPO 43: **Grafana alertas**

Y el correspondiente mensaje en Teams.

# DIAPO 44: **Container Insights**

Con container Insights podemos supervisar el rendimiento de nuestras cargas de trabajo, el rendimiento recopilando métricas. Una vez habilitada la supervisión se empiezan a recopilar y las métricas se nos mandan a Azure Monitor y los registros al área de trabajo de los Log Analystic.

# DIAPO 45: **Container Insights Reglas de alertas**

Las alertas nos van a informar de una forma proactiva cuando se detecten las condiciones que nosotros definamos en ellas sobre los datos que supervisarán. Nos van a permitir identificar y solucionar diversos problemas antes incluso de que los usuarios del sistema puedan verlos. En nuestro caso vamos a usar las de tipo métrica que nos van a decir cuando el valor de una métrica está superando el umbral que nosotros pongamos.

# DIAPO 46: **Container Insights Métricas**

Las métricas son una serie de valores numéricos que se recopilan automáticamente y describen algún aspecto de un recurso determinado en un momento determinado. Con las métricas podremos crear gráficos a partir de sus valores, todo ello por medio del explorador de métricas de Azure Monitor.

# DIAPO 47: **Container Insights Registros**

En esta sección, nos serviremos de Log Analytics para recopilar y analizar los datos de nuestro clúster a través de consultas, creando un área de trabajo.

Si seleccionamos los Registros en el menú del recurso. Log Analytics se abre con la ventana Consultas que harán solicitudes de lectura para procesar datos y devolver los resultados, que incluye consultas creadas previamente para el tipo de recurso, aunque nosotros también podemos escribir las nuestra si queremos. Vamos a ejecutar una de prueba para saber el ciclo de vida de todos los contenedores.