UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Evaluación de una infraestructura basada en Kubernetes para la mejora de la disponibilidad de servicios

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Javier Riera Chirivella

Tutor: Ignacio Blanquer Espert
Tutor externo: Eduardo Izquierdo Martínez
Director Experimental: Sergio López Huguet

Curso 2019-2020

Resum

Amb l'arribada dels contenidors i la seua popularització per tots els avantatges que ofereixen, va aparéixer la necessitat d'ampliar el seu ús a més de només un recurs. Per això, Kubernetes va ser concebut i ara és una ferramenta de gran capacitat utilitzada per grans sectors de la indústria gràcies a la facilitat que atorga als usuaris que volen desplegar una gran quantitat de contenidors en un elevat nombre de recursos.

Aquest treball intenta projectar la inclusió d'aquesta tecnologia en un flux de treball ya existent dins de l'equip de Qualitat del *Software* de la companyia MaxLinear Inc., millorant el sistema d'execució automàtica de proves de productes. Per això, s'estudiaran les tecnologies disponibles, es compararan i decidirà quines utilitzar, i després es crearà una fulla de ruta que seguir per a incloure eixes tecnologies al sistema ja existent.

A més a més, s'estudiarà com funciona eixe sistema i com s'integraran les noves tecnologies, i després es farà una anàlisi intentant explicar els diferents sots, problemes i solucions que se n'han anat trobat pel camí.

Finalment, es realitzarà una valoració sobre l'estat de la tecnologia, els diferents casos d'ús i una sèrie de recomanacions si es decideix utilitzar Kubernetes per a realitzar un projecte similar.

Paraules clau: Kubernetes, OKD, Openshift, contenidors, Docker, Jenkins, clúster, microservicis, disponibilitat

---

Resumen

Con la llegada de los contenedores y su popularización por todas las ventajas que ofrecen, apareció la necesidad de desplegar contenedores interconectados en múltiples recursos. Entre las alternativas que aparecieron, destaca Kubernetes, una herramienta de gran capacidad utilizada por grandes sectores de la industria debido a la facilidad que otorga a los usuarios que quieren desplegar aplicaciones con múltiples servicios en diferentes contenedores desplegados sobre un número elevado de recursos.

En este trabajo se pretende proyectar la inclusión de esta tecnología en un flujo de trabajo ya existente dentro del equipo de Calidad de Software de la compañía Maxlinear Inc., mejorando su sistema de ejecución automática de pruebas de productos. Para ello, se estudiarán las tecnologías disponibles, se compararán y decidirán cuáles utilizar, y después se creará una hoja de ruta que seguir para incluir dichas tecnologías en el sistema ya existente.

Además, se estudiará el funcionamiento de dicho sistema y cómo se integrarán en los sistemas de la empresa, además de hacer un análisis intentando explicar los diferentes problemas y soluciones que se han ido encontrando por el camino.

Finalmente, se realizará una valoración sobre el estado de la tecnología, los diferentes casos de uso y una serie de recomendaciones si se decide utilizar Kubernetes para realizar un proyecto similar.

Palabras clave: Kubernetes, OKD, Openshift, contenedores, Docker, Jenkins, clúster, microservicios, disponibilidad

Abstract

With the arrival of containers and their popularization thanks to all the advantages they offer, the need to widen their usage to more than one resource appeared. That is why Kubernetes was created and it is now a highly capable tool used by big sectors in the industry, thanks to the ease it brings to users who want to deploy a huge amount of containers across an elevated number of resources.

This paper aims to project the inclusion of this technology in an already existing workflow inside the Quality Assurance group of the company MaxLinear Inc., improving their automatic execution of products tests system. For that, the new available technologies will be studied, compared and finally chosen accordingly, and afterwards a roadmap will be created to be followed in order to include such technologies in the already existing system.

Also, the functioning of that system will be studied, and how to include these new technologies too. After that, an analysis will be provided trying to explain the different setbacks, roadblocks and solutions found along the way.

Finally, an assessment of the current state of the technology will be issued, as well as of the different use cases and a list of recommendations in case Kubernetes is chosen to create a similar project.

Key words: Kubernetes, OKD, Openshift, containers, Docker, Jenkins, cluster, microservices, availability

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Contexto

En la actualidad, el concepto de aplicación software se define muchas veces como un ecosistema de servicios que se relacionan e interactúan entre sí para lograr un resultado deseado. Esto, por tanto, define una serie de nuevos problemas que los mantenedores y desarrolladores de dichas aplicaciones tienen que abordar: despliegue, verificación, actualización, y otras tantas problemáticas que aparecen al utilizar más y más recursos externos. Para solucionar o hacer más llevaderos y sencillos estos problemas, se yerguen los contenedores y la serie de ventajas que acarrea el utilizarlos. Los contenedores proporcionan una gran cantidad de beneficios: Eliminación de efectos secundarios o laterales no deseados, mayor portabilidad y compatibilidad, coexistencia de versiones, empaquetado de recursos y sencillez de despliegue, entre otros.

Y por supuesto, con los contenedores también se facilitó la desmonolitización de los productos software. La posibilidad de crear una aplicación basándose en los beneficios que ofrecen los microservicios ayudó a que muchos proyectos nuevos se decantasen hacia esta nueva metodología. Con este cambio de paradigma, llegó una mejora transversal: la capacidad de la escalabilidad horizontal y vertical.

La escalabilidad horizontal consiste en añadir más recursos que sean capaces de ejecutar servicios del *software* en cuestión, mientras que la escalabilidad vertical consiste en incrementar la capacidad de los recursos ya existentes que están ejecutándolo. Muchas veces realmente es necesario hacer uso de la segunda, pero en otras muchas ocasiones, el uso de la primera puede ser más efectivo o económico. Es por eso que, a la hora de querer dividir las unidades ejecutables de un mismo proyecto en diferentes recursos distribuidos, surge la necesidad de un controlador que las maneje y las mantenga conectadas y funcionando.

Es en este punto en el que empiezan a aparecer los diferentes manejadores de contenedores en clústers de ordenadores, como Docker Swarm y similares. Actualmente, la tecnología más popular que se usa en el mercado es Kubernetes, así como todas sus variaciones y giros proporcionados por diferentes empresas y sectores de la comunidad. Kubernetes se ha llegado a denominar como un “Sistema Operativo de Contenedores en Sistemas Distribuidos“. Esto es así porque

es capaz de abstraer la lógica de manejar contenedores en diferentes puntos del clúster y añadir transparencia para el encargado de ejecutar el software.

1.2 Motivación

Este proyecto fue propuesto por la empresa MaxLinear Inc. como proyecto individual y paralelo al departamento de Calidad de Software. Consiste en investigar y construir tanto como sea posible de un sistema basado en Kubernetes para integrarlo en el flujo de trabajo del equipo y el actual sistema existente de Nube de Pruebas (TestCloud). Esta Nube de Pruebas se halla *on-premise*, es decir, es un sistema que se ejecuta en hardware local. La razón de existir de este proyecto es dependiente de muchos motivos: MaxLinear, al ser una empresa puntera en el campo de la tecnología, tiene que estar siempre atenta de las nuevas alternativas que se ofrecen en la comunidad para poder mejorar su funcionamiento, e intentar adaptar aquellas que crea que le puedan convenir. Además, debido al tipo de *testeo* que se realiza por parte del departamento, Kubernetes era el siguiente paso lógico. Los tests eran enviados desde un servidor de Jenkins a diferentes bancos de trabajo, ya encapsulados como contenedores que incluían todas las dependencias posibles.

No solo eso, sino que el hecho de utilizar Jenkins como una única puerta de entrada a las tareas de todo el TestCloud hacía que fuese un punto crítico. Al introducir Kubernetes como tecnología para controlar la vida y ejecución de los contenedores, se podría robustecer enormemente este servicio, evitando caídas innecesarias. De la misma forma, podría dividirse muy cómodamente la carga de trabajo de una sola instancia de Jenkins en varias más especializadas, gracias al enfoque de Kubernetes de reutilización de proyectos y clonación de los mismos para introducir pequeños cambios de manera rápida y eficaz.

Entre las ventajas que se ofrecen al implementar esta tecnología está, además, el poder utilizar recursos que ya no estén en uso para los proyectos específicos para los que se obtuvieron.

Aunque el objetivo de este proyecto no era sustituir el actual sistema de TestCloud, el hecho de adaptar las mejoras que Kubernetes podía introducir al flujo de trabajo era una idea que se dejaba abierta para el futuro, pues los cambios a la infraestructura y el modelo de trabajo del departamento se hacen de manera paralela, y con menos prioridad (evidentemente) que los proyectos para clientes. Es por eso que se decidió darle un plano secundario a este proyecto y plantearlo como un Trabajo de Fin de Grado, pues no era invasivo ni interfería con el resto de tareas del departamento y se podía realizar de manera tangencial al resto de las tareas del equipo.

1.3 Objetivos

Los diferentes objetivos del proyecto quedaron delimitados al principio del mismo. Como objetivo principal, se pretendía estudiar Kubernetes para ver cómo

se podría y si se debía integrar en el *workflow* del equipo, apoyando a la ejecución de Jenkins para reforzar su estabilidad y disponibilidad.

Como objetivos más específicos, se trazaron una serie de tareas a realizar:

1. Estudio de cómo funciona Kubernetes y cómo de plausible sería su utilización en el entorno. Este es un paso importante porque, sin él, quizá se habría comenzado a hacer pruebas sin saber realmente si la tecnología era adecuada. Una vez se entendiese cómo se podría integrar, se debería escoger la tecnología más adecuada de entre todas sus variaciones y generar una arquitectura en la que utilizarlo.

2. Instalación de una plataforma de Kubernetes en la que realizar diferentes pruebas de concepto y simulacros de integración de los diferentes servicios que se pretendían utilizar. En este entorno no conectado al *workflow* se podrían probar todas las tecnologías que hiciesen falta y así comprobar previamente a la integración que dichas tecnologías funcionan entre sí de manera adecuada.

3. Despliegue de los servicios concretos que se necesiten en el TestCloud, en concreto y como objetivo principal de entre ellos, Jenkins. Además, una serie de productos *software* se debían poner en marcha también para conseguir que la integración fuese correcta.

4. Elaboración de tests a cada paso del proyecto, para asegurarse que se han realizado como es debido. Además, a ser posible, estos tests deberían ser automáticos para poder comprobar que nada nuevo añadido ha hecho que algo anterior ya completado dejase de funcionar.

Si estos objetivos se cumplían, el paso final habría sido integrar toda la tecnología y lo aprendido en el entorno real de trabajo, pudiendo entonces avanzar el TestCloud y dotarle de mayor disponibilidad y de nuevas posibles características.

De hecho, aunque no se incluyese como objetivo, se habló de utilizar las capacidades de monitorización y control de contenedores para sustituir Jenkins o utilizarlo en tándem. Esto fue más un comentario que una meta real, pero merece ser destacada como un posible objetivo a largo plazo en caso de que el proyecto se siga desarrollando.

1.4 Planificación temporal

Los primeros meses, que cuentan desde el inicio de las prácticas a mediados de septiembre hasta finales de año, se dedicaron a la investigación, comprensión y elección de las herramientas a utilizar. De esta manera, lo primero que fue necesario estudiar fue cómo funcionaba Kubernetes a bajo nivel, es decir, entendiendo los diferentes elementos que lo componen y conforman, empezando desde la unidad más básica que es el pod y acabando con clústeres enteros. Esto conlleva estudiar Kubernetes como es y sin ningún tipo de apoyo de herramientas externas o modificaciones de la tecnología.

Una vez esto se consiguió, la búsqueda de las herramientas más adecuadas ocuparon el resto del tiempo asignado a esta preparación. Esto incluía no solo la búsqueda y comparativa a nivel superficial de las herramientas, sino también la creación de pequeñas Pruebas de Concepto a nivel local con máquinas virtuales.

Durante este proceso, además, se tuvo que explicar al equipo cómo funcionaba la tecnología de Kubernetes, y más tarde ofrecer una comparativa entre las posibles herramientas a utilizar para que fuese una decisión en la que todos los del departamento estuviesen, al menos, informados.

Una vez todo esto fue estudiado, llegó el momento de crear un plan de ruta para la elaboración del proyecto, dado que la tecnología ya se conocía y se había elegido qué se iba a usar. Con esto en mente, se creó una división de tareas según las necesidades que tenía el proyecto y se les asignó una prioridad y estimación temporal, lo que dio lugar a una planificación temporal que también se expuso a los miembros relevantes del grupo y que serviría para llevar un seguimiento del proyecto.

En la figura 1.1 se puede observar cuáles eran dichas tareas a partir de enero, cuando todo el estudio previo ya se había realizado y se pretendía realizar un prototipo.

1.4.1. Despliegue Básico

Como se puede comprobar, lo primero que se estimó fue el conseguir tener un despliegue básico funcional con tres nodos maestros y tres nodos trabajadores que se utilizarían más adelante como nodos de almacenaje. Se estimó que esta tarea duraría un total de diez días, y se subdividió en otras más pequeñas, entre las que se encontraban una serie de tests para comprobar que todo funcionaba correctamente.

Esta tarea se cumplió a mitad, y fue la causa del bloqueo del resto del proyecto. Se consiguió un clúster funcional formado por un nodo maestro al que se le añadieron dos nodos trabajadores. Además de esto, se automatizó una prueba que levantaba un pequeño servicio al que se podía acceder mediante un puerto abierto (usando NodePort), y que comprobaba si un nodo trabajador funcionaba correctamente sirviendo una pequeña interfaz web en HTTP que indicaba información del nodo en que se ejecutaba.

Fue al intentar añadir nuevos nodos maestros que la configuración falló y no se pudo volver al estado anterior en que funcionaba con solo un maestro y dos trabajadores.

Aunque los siguientes puntos no se consiguieron terminar, merece la pena explicar cuáles eran las tareas que se iban a realizar en ellos y el porqué de cada una de ellas.

1.4.2. Almacenamiento

Después del primer punto, el segundo consistía en dotar de almacenaje persistente al clúster, permitiendo así que diferentes servicios como las instancias de

Jenkins pudiesen hacer uso del mismo para guardar la información que les fuera necesaria. Para ello se iba a crear un *namespace* propio para desplegar Rook mediante un *operator* que se encargaría de utilizar solo los nodos indicados como nodos de almacenaje. Este punto, aunque puesto en segundo lugar, se probó brevemente a la vez que se creaba el test automático para comprobar si un nodo funcionaba que se ha comentado anteriormente, y se consiguió desplegar aunque sin que realizase las funciones adecuadas. Simplemente aparecía como un servicio desplegado pero no ejecutaba las tareas que debía.

Dado que ese despliegue no pudo ser retomado, tampoco se pudo indagar en el porqué de que no funcionase, y este error quedó sin solución.

Como prueba de este punto, se pretendía crear una instancia de Jenkins que, aunque no fuese accesible por interfaz web sin usar un NodePort, pudiese modificarse usándolo. De esta manera, se realizaría algún cambio de configuración menor y se eliminaría la ejecución del servicio de manera forzosa, obligando al clúster a volver a levantarlo. Si la configuración utilizada por Jenkins mantenía los cambios de configuración, el almacenamiento al menos funcionaba parcialmente.

La razón de utilizar Jenkins como terreno de pruebas se debe a que, puestos a realizar una prueba de funcionamiento, usar Jenkins para ello suponía estudiar cómo debía instalarse el servicio que más tarde sería imprescindible para que el proyecto fuese exitoso.

1.4.3. Ingress

Como tercer punto, se pretendía desplegar un servicio de Ingress que se encargase de canalizar y distribuir el acceso a los diferentes servicios desplegados del clúster, y así no tener que depender de NodePorts u otras soluciones similares.

El proceso sería similar al del punto anterior, siendo necesario crear un *operator* que fuese capaz de detectar cambios en la red interna del clúster y ofrecer redireccionamiento incluso si los servicios eran relanzados en otros nodos diferentes al que originalmente tenían.

Como prueba, se pretendía reutilizar la prueba de HTTP del primer objetivo, y añadir además una manera de acceder a la interfaz gráfica de Jenkins sin usar NodePorts.

1.4.4. Permisos y roles

Este punto, el cuarto, se centraría en la creación de un sistema que permitiese utilizar las cuentas ya existentes en el Active Directory de la empresa, asignando a cada uno los roles estrictamente necesarios para que pudiesen trabajar en el clúster.

Esto requería, primero, conectar el acceso basado en roles propio de OKD a Active Directory y, una vez hecho eso, la creación de roles predeterminados y específicos para asignar a cada uno de los usuarios.

Cada rol nuevo debía tener su propia comprobación de que funcionaba, así como además realizar varias pruebas comprobando que, si se introducían datos erróneos o se pretendían realizar actividades prohibidas para ciertos roles, estas serían evitadas por el clúster.

Para facilitar todo esto, se crearía un *namespace* de prueba en el que probar todas estas configuraciones.

1.4.5. Servicios comunes

Este punto, dado que por planificación quedaba fuera del tiempo dedicado al proyecto, se estableció como una posibilidad a más largo plazo, en caso de que este se quisiese continuar.

Por ello, el desglose es mucho menos detallado, y básicamente tiene en cuenta la creación de un *namespace* nuevo en el que desplegar dos productos del *software* sin mayor profundidad y con una estimación posiblemente demasiado optimista.

De todas formas, el proceso habría sido similar a los de los puntos anteriores, y habría consistido de una creación de un *operator* por cada servicio y de una serie de *tests* que comprobasen que en efecto funcionasen.

1.4.6. Conclusiones de la estimación temporal

La estimación temporal en sí no parece del todo arriesgada ni demasiado optimista. El mayor problema que hubo durante el desarrollo del proyecto fue el gran bloqueo en el primer punto de todos, que era esencial para el desarrollo de las siguientes.

Si se hubiese dispuesto de un clúster funcional, el resto de las actividades posiblemente hubiesen podido realizarse dentro del marco temporal establecido.

1.5 Estructura de la memoria

El primer punto se dedicará a los apartados de motivación y cómo surgió el proyecto, así como una pequeña descripción de lo que se pretende solucionar y cómo.

En el segundo punto, se estudiará el estado actual de la tecnología a utilizar. Esto incluye los desarrollos punteros y las implementaciones actualmente disponibles a mayor escala y a mayor cantidad de usuarios. Además, se presentarán las tecnologías empleadas y el porqué de su selección.

En el tercer punto se expondrá el diseño que se pretendía implementar y cómo trabajaría en tándem con las herramientas y *workflows* que ya tenía implementados la empresa. Además de la arquitectura a utilizar, se explicarán los requisitos *hardware* y *software* que eran necesarios para el proyecto.

El cuarto punto estará dedicado a los resultados obtenidos y una serie de explicaciones relacionadas con los errores cometidos y problemas encontrados du-

rante la ejecución del proyecto. Se detallará el porqué de ciertos problemas, así como las soluciones y *workarounds* encontrados.

Más adelante, en el punto quinto, se procederá a relatar las conclusiones que se han extraído del proyecto, tanto datos objetivos como pinceladas de opinión personal y previsiones o propuestas de cara al futuro de la tecnología Kubernetes y sus derivados.

Más allá de estos puntos se encontrarán una sección de bibliografía y otra de apéndice. Estas secciones dispondrán de material adicional que sirva para mostrar las fuentes que se han usado como referencia y añadir contenido suplementario, respectivamente.

ETA	Days of work						
January 27th	10	Basic Deployment - 3 Masters & 3 Storage Nodes					
		3	Deploy 1 Master Node				
		2	Add Worker Node (Dummy)				
		1	Run Sanity Check (Http Hello World) at Dummy				
		1	Automate Sanity Check				
		1	Add 2 Master Node and Perform Sanity Check				
		1	Add 2 Worker Node with Affinity and Perform Sanity Check				
		1	Kill 1 Master Node to test High Availability				
February 7th	18	Storage - Ceph and Rook					
		1	Create Namespace for Rook				
		5	Configure Operator (YAML) for 3 Nodes				
		5	Deploy Operator				
		7	Jenkins Test				
			5	Deploy container			
			1	Change configuration			
			0,5	Kill container			
			0,5	Check that configuration persists			
March 4th	18,5	Ingress - Access to the cluster					
		2	Deploy Ingress Node				
		4	Configure Ingress Operator				
		3	Deploy Ingress Operator				
		1,5	Test Basic Hello World HTTP				
		6	Test Jenkins				
			2	Access to Test Cloud GUI			
			4	Add Jenkins slave to Jenkins Instance			
		2	Wildcard DNS				
April 1st	14,5	Permissions - Role Based Access Control and Active Directory Integration					
		5	Connect K8S RBAC to Active Directory				
		2	Set Default role for users				
		1	Create Namespace for project				
		2	Define Global Roles				
		2	Define Project Local Roles				
		2,5	Test Roles				
			0,5	IT			
			0,5	Admin			
			0,5	Dev			
			0,5	Systest			
			0,5	Negative test (Try to login with a user that shouldn't be able to)			
	6,5	Common Services - Shared Project					
		0,5	Create Common Namespace				
		3	Deploy Pypi				
		3	Deploy Docker Registry				
	67,5	Whole Project					

Tabla 1.1: Planificación temporal

CAPÍTULO 2
Contexto Tecnológico

En este punto se explicará en qué estado se encuentra actualmente la tecnología y qué herramientas se han utilizado para la realización del trabajo. Para ello, se revisará en qué punto está el estado del arte y se dará una explicación al uso de las diferentes herramientas.

Además, se le otorgará un apartado especial a la elección de la distribución de Kubernetes escogida, pues fue un punto determinante en el proyecto y requirió de una evaluación especial por parte del equipo y del estudiante.

2.1 Estado del Arte

Para entender el estado actual de la tecnología y gran parte de este proyecto, se debe explicar qué son y para qué se usan los contenedores.

Los contenedores [1] son conjuntos de procesos que se ejecutan de forma aislada en una partición de los recursos de un sistema, y proporcionan mejoras a la hora de la entrega de aplicaciones, la contención controlada del uso de recursos o el aislamiento de entornos. Por diseño, los contenedores forman parte del sistema operativo y, por tanto, están pensados para ser ejecutados en un entorno de solo un recurso, pero es precisamente por la cantidad de beneficios que ofrecen y por su popularización en el ámbito de los microservicios que surgió la necesidad de gestionar varios grupos de contenedores a lo largo de varios recursos distribuidos.

Es por eso que, a mitad de 2014, un grupo de ingenieros de Google decidieron poner en marcha el proyecto Kubernetes, con la idea de ser capaz de distribuir varios contenedores en diferentes máquinas y que se pudiesen comportar como un único sistema. Aunque Kubernetes no es la única tecnología que se usa actualmente para cubrir esta necesidad, sí es la que se va a tratar en este proyecto y por tanto la que más va a ser nombrada. No obstante, cabe destacar que hay otras opciones también muy populares, como Mesos y Nomad, que ofrecen soluciones similares.

El proyecto Kubernetes rápidamente cogió forma y se asoció con la *Linux Foundation* [2], y con todo el esfuerzo dedicado a su desarrollo se convirtió en uno de los proyectos con más commits de todo GitHub.

Previamente, lo más utilizado para manejar varios grupos de contenedores (en concreto de la tecnología Docker) era *Docker Swarm*. Sin embargo, al llegar Kubernetes al mercado, con una propuesta mucho más robusta y con más capacidades, *Docker Swarm* se dejó de lado en pos de Kubernetes en muchos ámbitos, aunque sigue siendo una tecnología muy eficaz para gestionar unos pocos recursos.

Actualmente, una larga serie de empresas punteras en el sector tecnológico hacen uso de Kubernetes o sus derivados, contándose entre ellas BlaBlaCar, Booking.com, Huawei, IBM, ING, Nokia, Pinterest y muchas más [3].

2.2 Tecnologías empleadas

Durante el resarrollo de este proyecto se ha hecho uso de muchas y muy variadas tecnologías, que funcionan en tándem para poder obtener el resultado deseado. Aquí se procede a explicar cuáles eran las principales y cómo encajan en la imagen global del proyecto, así como mostrar su funcionamiento básico.

2.2.1. Docker

Es importante hacer una clara distinción entre la virtualización tradicional y los contenedores Docker. Así como en la virtualización típica se emula un Sistema Operativo entero, Docker utiliza únicamente lo estrictamente necesario que se especifica en la imagen.

Docker es, a día de hoy, la tecnología de virtualización por contenedores más popular de la industria. Creada en 2013, permite la creación de imágenes muy livianas que se ejecutan con el servicio de Docker Engine y dan un rendimiento igual o superior al de la virtualización tradicional (aunque con menor aislamiento).

Una de las mayores ventajas que ofrece Docker es la capacidad de empaquetar en un mismo contenedor tanto el producto *software* (o productos) deseado junto con sus librerias necesarias y requisitos de software. Esto proporciona una capacidad de migración enorme, dado que lo único que se necesita tener instalado de forma nativa en la máquina en la que se va a ejecutar es el propio motor de Docker.

Además de eso, proporciona la posibilidad de tener en ejecución dos versiones del mismo *software* simultáneamente. Esto puede ser una tarea difícil en ocasiones, pues estas versiones diferentes pueden requerir el acceso a los mismos servicios o captar las mismas llamadas y provocar problemas de incompatibilidad. Dado que los contenedores Docker ofrecen un mayor aislamiento, cada uno accede únicamente a los ficheros de su imagen, permitiendo que funcionen de manera independiente.

Entre las ventajas que ofrece encontramos, también, la de evitar efectos laterales o no deseados. Al tratarse de un entorno cerrado sin acceso real a ninguna ruta o archivos que no se les hayan permitido de antemano, no pueden causar daño en el sistema en el que se ejecutan. Esto puede ser de vital importancia para

muchos proyectos, y Docker (junto con el resto de opciones de contenerización) proporciona una opción fiable en la que realizar el desarrollo de los mismos.

De la misma manera, el uso de Docker conlleva una serie de problemas por necesitar el uso de *root* y por estar concebido como un software *single-tenant*, es decir, que solo tiene un usuario que maneja todos los contenedores. Esto da, de esta forma, acceso a los contenedores de diferentes usuarios a aquel que sea el usuario de Docker en esa máquina. Kubernetes da solución a este problema con la creación de las *service accounts*.

La razón por la que Docker es la tecnología escogida para este proyecto es, precisamente, por ser un estándar de la industria. Se plantea buscar o crear versiones en contenedor de los programas que se quieran ejecutar en el clúster de ordenadores, así que entender cómo funciona esta tecnología era importante a la hora de llevar a cabo el proyecto. Además, es la tecnología preferida por Kubernetes para manejar contenedores, aunque da soporte a varios tipos más de contenedores [4].

2.2.2. Kubernetes

Una vez ya se entienden las ventajas y beneficios que ofrece utilizar contenedores, llega la hora de explicar la tecnología que se utilizará en este proyecto: Kubernetes.

Como ya hemos visto, uno de los problemas que trae consigo Docker y la tecnología de contenedores en general es que están diseñadas para trabajar en sistemas de un único recurso. Dado que en este proyecto se pretende utilizar una gran cantidad de máquinas y recursos diferentes en los que realizar diversas ejecuciones, se debe encontrar alguna solución que permita expandir su uso.

Es por eso que Kubernetes es la elección tomada, pues se encarga de controlar a lo largo de múltiples recursos la ejecución de diversos contenedores, y ofrece funcionalidades de abstracción para facilitar su manejo.

Aunque ya se ha explicado algo en un punto anterior de la memoria, en este apartado se procederá a ahondar en cómo funciona a nivel más básico y cómo interactúan entre sí los diferentes componentes de Kubernetes.

He aquí los componentes principales:

Objetos

- **Pod**: Los *Pods* (o Vainas, en castellano, aunque les llamaremos por su nombre en inglés), son la unidad más básica de la tecnología de Kubernetes, la célula sobre la que se construye en resto de elementos. Se tratan de elementos que se componen de uno o varios contenedores que han de trabajar muy estrechamente, y se asegura que todos estos contenedores compartirán red interna y máquina local.

- **Replica Set**: Este componente se encarga de organizar un conjunto de pods y mantenerlos. Es capaz de, de manera declarativa, garantizar que un número de pods concreto será servido siempre que sea posible. Esto significa

que se pueden especificar diferentes políticas que especifican la manera en la que se encarga de manejar los pods una vez estos fallan o se caen, o cuando se pide que el número de pods deseados aumente o disminuya.

- **Deployment**: Al igual que los Replica Set se encargan de manejar de manera declarativa una serie de pods, un deployment se encarga de hacer lo mismo pero con Replica Sets. Esto provee al usuario de un nivel de abstracción mayor que le permite organizar de manera más sencilla múltiples pods a lo largo de varios recursos.
- **Configmap**: Este tipo de objeto sirve para almacenar variables y configuraciones ajenas al código de las aplicaciones que pueden estar siendo ejecutadas por Kubernetes. Aunque muchos objetos de Kubernetes ya tienen una sección denomidada *spec*, un ConfigMap tiene una sección *data* para almacenar items, identificados por una clave, y sus valores.
- **Service**: Este objeto es una abstracción que define un conjunto de *pods* y la política de acceso que se ha de seguir para conectarse con ellos. Los hay de varios tipos según lo que se necesite, formando un conjunto de cuatro diferentes que se utilizarán en diferentes situaciones: **ClusterIP**, **NodePort**, **LoadBalancer** y **ExternalName**. Estos se explicarán a continuación.

Servicios

- **ClusterIP**: Este tipo de servicio expone el servicio con una IP interna del clúster, lo cual le da acceso solo a procesos ya pertenecientes al clúster que quieran conectarse.
- **NodePort**: El NodePort asigna un puerto estático dentro de la IP del nodo que esté ejecutando el servicio, de tal manera que se pueda acceder a él mediante <IPDelNodo>:<PuertoDelNodePort>.
- **LoadBalancer**: Este servicio es especial, en el sentido de que crea servicios tanto de ClusterIP como de NodePort para poder exponer el servicio a la red, permitiendo acceso desde fuera del clúster a los contenedores internos que se hayan definido.
- **ExternalName**: Este tipo de servicio se mapea a la dirección que se le haya otorgado en su campo llamado externalName.

Componentes de nodos Kubernetes utiliza una serie de elementos para poder tomar control y manejar los pods en los diferentes nodos que conforman el clúster. Por eso, hay una serie de componentes específicos que se despliegan dentro de cada nodo, entre los cuales se destacan:

- **Kubelet**: El Kubelet es una pieza de vital importancia en el correcto funcionamiento de un clúster de Kubernetes. Es un servicio que se despliega en cada uno de los nodos o máquinas de la red, y se encarga de observar los contenedores de su nodo, y de pararlos, iniciarlos y organizarlos según se le indique por el Plano de Control. Además, recibe de Kubernetes una

serie de especificaciones de Pod, llamadas *PodSpec*, y se asegura de que los contenedores que se definen en ellas funcionan adecuadamente.

- **Kube-proxy**: Es un proceso que se despliega en cada nodo y es el encargado de proporcionarle los servicios de red necesarios para pertenecer al clúster. Permite mantener las reglas de red propias del anfitrión y hace reenvío de conexiones.

Otros conceptos importantes No solo es importante saber la estructura de las diferentes arquitecturas creadas para desplegar contenedores, sino que además hay que entender una serie de conceptos que son necesarios para poder manejar un clúster de Kubernetes de manera eficaz:

- **Plano de Control**: El *Control Plane*, en inglés, es como se denomina al conjunto de procesos y *software* encargados de manejar la estructura del clúster. Sin él, no hay nada que se encargue de mantener la salud del mismo, y los procesos que son necesarios para que el clúster funcione como tal, fallarían o no serían mantenidos.
- ***High Availability***: Este término se utiliza para definir un tipo de despliegue en el cual se dispone de cierta redundancia para poder mantener el clúster activo a pesar de fallos inesperados. No solo hace referencia a un clúster de Kubernetes en concreto, sino que se trata de un término general utilizado para definir esta capacidad de un sistema o servicio que implementa redundancia.
- **kubectl**: Esta herramienta de control por línea de comando es la que permite interactuar y controlar el clúster de Kubernetes. Requiere ser instalada, aunque es apenas poco más que un binario, y utiliza un archivo de configuración válido de una red de Kubernetes, ya sea porque el usuario se lo otorga mediante una *flag* determinada o porque lo encuentra en una ruta predeterminada. Esta herramienta permite observar el estado actual del clúster a varios niveles y según diferentes definiciones, dando así información tanto de nodos a nivel global como de pods en específico. También se usa para desplegar servicios usando un archivo de configuración previamente creado, generando así estados deseables de manera declarativa.

Toda esta información se encuentra en la web oficial de documentación de Kubernetes [7].

2.2.3. Ceph

Uno de los puntos de los que depende este proyecto es la capacidad de almacenar de forma persistente tanto archivos, como configuraciones, como reportes de *tests* y similares. Por ello, había que encontrar una tecnología adecuada para utilizar de manera distribuida en un clúster de Kubernetes.

Que fuese almacenamiento ditribuido era importante por la misma razón por la que lo era que lo fuesen el resto de servicios: la redundancia y accesibilidad.

Si los archivos se corrompían o eran inaccesibles, incluso de manera temporal, el flujo de trabajo de la compañía se podía quedar estancado, y por eso mismo se buscó una solución que cumpliese esos requisitos.

Ceph apareció como una opción muy importante a tener en cuenta. Se trata de un sistema de archivos **distribuido** y libre, con años de experiencia y gran reputación.

Aunque Ceph es una herramienta muy potente de por sí, está pensada principalmente para ser utilizada como un producto *standalone*, y no fue diseñada con Kubernetes en mente. Por tanto, su integración en el proyecto requiere de *software* de terceros. En este caso, el que se va a explicar en el punto siguiente, Rook.

2.2.4. Rook

Rook es otra tecnología también adoptada por la *Linux Foundation* [2]. Se trata de un orquestrador de múltiples soluciones de almacenamiento, orientado específicamente a Kubernetes.

Funciona con diferentes proveedores de almacenamiento, como EdgeFS, Cassandra, NFS o, en el caso que es de mayor importancia para esta memoria, Ceph. Rook se encarga de añadir un nivel de abstracción al manejo del almacenamiento persistente que facilita su incorporación en el ecosistema de un clúster de Kubernetes, y consigue hacer esto creando un *Operator* (Operador, en castellano).

Un *Operator* es un proceso o serie de procesos que se despliegan en forma de contenedores dentro del clúster, y que se encargan de manejar diferentes servicios integrados. Suelen utilizarse para integrar software que trabaja mano a mano con la estructura distribuida de Kubernetes, normalmente para apoyarla y expandirla.

Además, dado que los servicios desplegados en el clúster iban a hacer uso de este almacenamiento, que fuesen capaces de hacerlo de manera sencilla e integrada con la tecnología de Kubernetes era altamente importante.

2.2.5. NGINX

El acceso a los diferentes servicios que se han de desplegar en el clúster, si es ajeno al mismo, debe ser monitorizado y controlado por algún servicio que trabaje bien en conjunto con Kubernetes. Para esto, el propio Kubernetes ofrece lo que llama un servicio de *Ingress*. Aunque el propio *software* ofrece esta posibilidad, se apoya en la existencia de otros productos más especializados para manejar el tráfico tanto de entrada como de salida a los diferentes servicios desplegados en la red.

Es aquí donde entra en juego NGINX. Es un producto del *software* que, entre sus capacidades, tiene la de ser un servicio de *Load Balancing* (Distribución de carga), y está ya preparado para desplegarse en Kubernetes de forma prácticamente nativa, dedicándole la propia página de Kubernetes una sección a cómo configurarlo [5].

2.2.6. CentOS 7

CentOS es el Sistema Operativo Linux que es la contraparte gratuita y de código libre de la distribución proprietaria de Linux a la que da soporte RedHat, Fedora.

La razón por la que se incluye esta distribución en la lista es porque es uno de los requisitos que hay que cumplimentar a la hora de utilizar el *software* elegido para crear el clúster de Kubernetes, al menos usando la opción libre de pago.

Funciona como prácticamente cualquier otro sistema GNU/Linux que utilice systemd. En concreto, en este proyecto se hace uso de la versión 7, con la que fuese la versión más actual y estable del kernel en cada paso de la investigación.

2.2.7. SELinux

SELinux (***S**ecurity **E**nhanced **Linux***, o Linux de Seguridad Mejorada) es un sistema de etiquetado y protección adicional que viene incluido con CentOS y que sirve para dar una capa extra de seguridad.

Este producto del *software* no solo es requerido por nuestra configuración, sino que además es un artículo importante y recomendable para aumentar nuestra seguridad, dado que la creación de una red de computadores es siempre un punto a tener en cuenta a la hora de evitar posibles ataques a la empresa o el entorno en que se use.

Por diseño, este es un producto del *software* muy estricto, y durante varios puntos del proceso de creación de este proyecto aparecieron problemas y bloqueos relacionados con una pobre implementación de SELinux en el sistema. Estos problemas se tratarán en más profundidad en el cuarto punto de esta memoria.

2.2.8. Jenkins

Jenkins es un producto del *software* que se encarga de automatizar y optimizar las tareas de *DevOps* (*Development and Operations*, Desarrollo y Operaciones), y facilitar el *CD/CI* (*Continuous Development/Continuous Delivery*, Desarrollo Continuo/Entrega Continua).

Es una herramienta utilizada por el equipo de QA para automatizar el despliegue y ejecución de *tests* en diferentes recursos, y permite organizar los resultados y ejecuciones de manera que los ingenieros tengan que gastar menos tiempo en estas tareas. Además, permite la conexión con otras herramientas de análisis y control del proyecto con funciones orientadas a metodologías ágiles como el SCRUM, lo cual facilita entender en qué estado están los diferentes proyectos que pueda estar abordando el equipo.

Por tanto, este producto del *software* es una de las piezas clave, si no la que más, a la hora de desplegar en el nuevo clúster.

2.2.9. Ansible

Ansible es un motor de automatización de IT que se utiliza para automatizar muchas tareas de aprovisionamiento de *software*. Es muy útil para tareas de gestión de servidores, y permite realizar despliegues y configuraciones en recursos de manera sencilla.

En este proyecto, se utilizará Ansible como herramienta requerida por OKD para instalar en los nodos que se vayan a añadir al clúster todo el *software* necesario, así como configurarlo de la manera adecuada.

No se deberá instalar en todos los nodos del clúster, sino en un único recurso que se utilizará de *launcher* o lanzador desde el cual se provisionará a todos los recursos del clúster.

2.3 Análisis de las Tecnologías Disponibles

A la hora de afrontar la posibilidad real de montar un clúster de Kubernetes *baremetal*, es decir, en máquinas locales y provistas por la empresa o el usuario, se han de estudiar las diferentes alternativas que hay que faciliten no solo el despliegue del mismo, sino también su mantenimiento.

Por suerte, Kubernetes se trata de un proyecto de código libre y por tanto se han desarrollado muchas variaciones a su alrededor, diferentes giros y sabores que se pueden aplicar a diferentes entornos. Esto es precisamente lo que se tiene que estudiar: Qué alternativas existen y cuáles se adaptan mejor al contexto de este proyecto.

Durante el estudio de cuáles podrían ser estas alternativas, surgieron 3 posibilidades principales, que serán detalladas en los próximos fragmentos.

2.3.1. Kubernetes

Aquí tendríamos la opción más básica. Un clúster de Kubernetes, sin ningún tipo de variación, únicamente construido desde cero usando las herramientas de la página oficial.

Cierto es que esta es una opción muy llamativa, sobre todo a la hora de querer un control total y absoluto de todo lo que se ejecute en la red de computadores. Por desgracia, como solución empresarial no se trata de la mejor idea.

Esto es así por varias razones:

- Dado que la creación de este proyecto fue realizada por un alumno en prácticas, y su contratación no era segura, la empresa corría el riesgo de no ser capaz de tener un trabajador que pudiese coger el relevo a la velocidad adecuada. Aunque la automatización del despliegue era plausible, dado que una vez realizados los archivos de configuración todo esto se podría simplificar a una sola receta de Ansible, por ejemplo, el hecho de necesitar a alguien que conociese la tecnología suficientemente como para crear un clúster desde cero sin necesidad de archivos ya generados es importante si

se quiere modificar en algún momento futuro, o incluso si se ha realizado algún error de configuración que no se haya detectado.

- Si se quisiese mantener un sistema tan complejo, habría o que mantener al alumno contratado o que formar a uno de los trabajadores en plantilla y asignarle más tareas de las que tiene actualmente.

- No existe una solución empresarial a la que acudir en caso de querer hacer consultas sobre la implementación del sistema montado. Cierto es que existen profesionales con títulos certificados en tecnología Kubernetes, pero el propio proyecto de Kubernetes no ofrece ninguna forma de contactar con dudas de nivel negocio sin pedir ayuda a la comunidad.

- La GUI (*Graphical User Interface*, o Interfaz Gráfica de Usuario), aunque existente, no cubriría todas las necesidades que tiene el equipo. Además, la existencia de una CLI (*Command Line Interface*, o Interfaz de Línea de Comando), aunque bienvenida, no es lo ideal para este proyecto. Esto es así porque, aunque dispone de una API muy rica y completa, la mayoría de los miembros del equipo harían uso del clúster de manera casi utilitaria o transparente, accediendo directamente a la instancia de Jenkis que necesitasen o similar. Volviendo al punto explicado anteriormente, el hacer uso específico de la CLI introduce una complejidad que no es deseable porque añade tiempo de aprendizaje por miembros del equipo o contratamiento de gente nueva que sea capaz de realizarlo, así que una opción más simple y gráfica es preferible.

Como se puede observar, aunque es una opción muy potente y que realmente puede funcionar muy bien en ciertos escenarios, en este caso no es la opción idónea, así que se han de buscar alternativas.

2.3.2. Rancher

Una vez descartada la opción de Kubernetes sin ninguna modificación, se deben estudiar las diferentes versiones que se han creado usándolo como tecnología base. Hay varias alternativas para ello, pero una de las que más relevancia tiene actualmente en el sector es Rancher.

La compañía Rancher Labs, creadora del proyecto Rancher, fue fundada en 2014 y pretende ofrecer herramientas para manejar Kubernetes a nivel empresarial desarrollando herramientas que faciliten su despliegue y control.

Una de las ventajas que tendría elegir Rancher consiste en la amplia cantidad de herramientas que ha creado la compañía para facilitar el uso de Kubernetes. Esto incluye:

- Una variación de Kubernetes más minimalista, llamada **K3S** (Un juego de palabras basado en la abreviación típica de Kubernetes, K8S)

- Un sistema operativo diseñado para únicamente realizar tareas propias de un clúster de Kubernetes, llamado **RancherOS**.

- Una distribución de Kubernetes que se despliega completamente en contenedores llamada **RKE** (Rancher Kubernetes Engine), que además facilita mucho su despliegue.

- Una herramienta para crear almacenamiento persistente, llamada **Longhorn**.

En definitiva: Al elegir Rancher, no solo eliges una distribución de Kubernetes, sino que accedes a todo un ecosistema, si así lo deseas.

Otro de los puntos importantes a tener en cuenta es el soporte existente para empresas. Este es un punto que tiene cubierto Rancher, pues aunque el código de su *software* es Open-Source y su uso es gratuito, disponen de atención profesional de pago por parte de la compañía, y con diferentes rangos según el tamaño del clúster creado.

Esta es, por tanto, una opción muy interesante a la hora de tenerla en cuenta para el desarrollo de este proyecto, y más tarde estudiaremos si su elección es la más adecuada.

Además, la GUI ofrecida por esta opción es bastante más capaz que la de Kubernetes estándar, y ofrece un sistema incluido de manejo de usuarios y permisos. Este último punto es muy importante dado que el contexto de trabajo en equipo en el que se va a utilizar requiere de un sistema robusto de control de acceso basado en roles.

2.3.3. OKD / Openshift Origin 3.11

La última alternativa que se va a presentar en esta memoria es la versión creada por RedHat. Se trata de la versión de código abierto y dirigida por la comunidad del *software* proprietario de Openshift, la distribución de Kubernetes de grado profesional de RedHat.

Esta distribución, a diferencia de Rancher, no incluye una gran cantidad de herramientas desarrolladas específicamente para su uso en conjunto con lo ofrecido por RedHat o sus versiones de comunidad, sino que se centra en tener un único producto potente que pueda facilitar el despliegue y manejo del clúster. Además, la propia wiki de OKD se preocupa de hablar sobre la integración de ciertos productos externos que hacen el trabajo mejor que las herramientas propias que ellos proporcionarían.

Uno de los puntos más atractivos de utilizar OKD es que se trata de prácticamente el mismo *software* que Openshift, el producto extremadamente popular de RedHat. Esto se traduce en una amplia y completa documentación hecha por una empresa que se dedica a ofrecer soluciones de IT a empresas, y que RedHat es una compañía con peso e influencia en la industria.

A tener en cuenta como factor muy positivo está el hecho de que OKD se basa en una serie de tecnologías que la empresa ya controla y utiliza de manera habitual. El despliegue del clúster se realiza utilizando una herramienta llamada Ansible, que es altamente usada por el equipo a la hora de desplegar máquinas virtuales en los bancos de trabajo. Además, uno de los Sistemas Operativos válidos para su instalación es CentOS, que ya es habitual para el departamento de IT

y tiene ya preparadas varias imágenes listas para su instalación como máquinas virtuales.

Además, la GUI que ofrece es tan competente como la de Rancher, y ofrece también un sistema de control de usuarios RBAC (*Role Based Access Control*, Control de Acceso Basado en Roles), y ofrece una versión de la CLI más potente y totalmente compatible con la herramienta original de Kubernetes, *kubectl*.

Cabe decir, a modo de nota, que aunque la versión 3.11 era la más actual al inicio de este proyecto, se hablaba ya por las redes del lanzamiento de la versión 4.0. Esto es así porque Openshift ya estaba en ese momento en marcha y en producción en esa versión. Al estar finalizando el proyecto, apareció en versión beta, aunque bastante estable, la versión 4.0. En las últimas semanas del proyecto, se le dedicó atención para comprobar si podría funcionar, cosa que se explicará en el punto cuarto.

2.3.4. Decisión

A la hora de elegir, y tras haber analizado todas las alternativas, se decidió hacer una comparativa según las necesidades de la empresa, y se hizo una presentación a todo el equipo para que todos pudiesen dar su opinión y elegir entre OKD y Rancher, pues se concluyó que eran las dos opciones más adecuadas.

	Rancher	OKD	Best
Upgrade Cycle	Every 6 months	Every 3 months	-
Update Process	More downtime	Less downtime	okd
Ceph and Rook Integration	Documented	Documented	-
Deployment and Resizing	RKE	Ansible	okd
RBAC	Only GUI	GUI and Command line	okd
Disconnected Installation	Documented	Documented for Openshift	RANCHER
Active Directory	Documented	Documented	-
Official Support	Paid	Officially non existent, but probable if paid	RANCHER
Community Support	More involved	Lesser, but has Openshift	RANCHER
Other applications	Generic	Generic	-

Tabla 2.1: Tabla comparativa entre las diferentes opciones

Además de las cosas importantes que ya se han explicado de manera individual en cada punto dedicado a cada tecnología, cabe destacar una serie de puntos que se pusieron cara a cara específicamente para esta comparación. A continuación se explicarán estos puntos y el por qué de cada ganador.

- **Ciclo de *Upgrade*:** En este punto se trata de observar cada cuánto se genera una nueva versión menor del software, sin contar betas ni alphas. Para ello, y dado que ambos productos se encuentran de forma pública en GitHub, se hizo un escrutinio para observar la periodicidad con la que se lanzaban nuevas *releases*. Dado que no se encontró una clara mejora entre una periodicidad u otra, se decidió dejar este punto como un empate.

- **Proceso de *Update*:** Aquí se trata de comparar cómo es el proceso de actualizar todas las máquinas del clúster, o al menos las que se pretenda actualizar. Para esto, se revisó que Rancher no tenía capacidad de hacer una actualización *in-place*, es decir, sin tener el clúster no funcional en ningún momento. Por el lado contrario, OKD permite utilizar un algoritmo llamado *green-blue* que permite ir actualizando los nodos en diferentes fases. Este proceso, aunque tedioso y algo complejo, permitía no tener que dejar el clúster inaccesible ni no funcional en ningún momento, aunque sí mermando sus capacidades. Este punto se lo lleva OKD, por tanto.

- **Integración de Ceph y Rook**: Dado que la tecnología deseada para proveer almacenamiento persistente se trata de una combinación de Ceph y Rook, la integración de dichas tecnologías era importante. En ambos casos, la documentación oficial de Rook se encargaba de explicar cuáles eran los pasos a seguir para ponerlo en marcha, así que no hay un ganador para esta sección.

- **Despliegue y cambio de tamaño**: Aunque ya se ha explicado brevemente, tanto OKD como Rancher hacen uso de un conjunto de scripts y *software* para desplegar y ampliar los clústers que creen. Aunque es cierto que RKE es una tecnología muy potente, el hecho de que en el equipo ya se conociese Ansible y se supiese que es una tecnología mucho más versátil, inclina la balanza a favor de OKD.

- **Control de Acceso Basado en Roles**: Aunque ambas tecnologías son capaces de crear una serie de restricciones muy modulares basadas en los roles asignados a cada usuario, la única forma de hacerlo en Rancher era vía GUI, que podía suponer un problema a la hora de querer automatizar ciertos procesos. Por contra, OKD utiliza su versión extendida de *kubectl* para poder realizar estos cambios vía CLI, lo cual lo hace mucho más versátil a la hora de generar roles para todos los empleados de la empresa.

- **Instalación Desconectada**: Este punto hace referencia a la posibilidad real que se comentó a la hora de proyectar este trabajo sobre utilizar esta tecnología en un centro de datos en una localización ajena a la empresa, en la cual no habría acceso a internet por parte del clúster. Esto significaría que no habría ningún tipo de acceso a internet ni siquiera durante la instalación, así que la capacidad de hacerlo sería interesante. Aunque Rancher tenía ampliamente documentado en su web cómo realizar este tipo de instalación, OKD no tenía dicha información. Cabe destacar que en el caso de Openshift (recordemos, prácticamente el mismo código, pero proprietario) sí existía un apartado detallando el proceso a seguir.

- **Active Directory**: Este punto no apareció hasta poco antes de realizar la decisión, y cabe destacar que ambos tienen también integración del repositorio de usuarios de Microsoft *Active Directory*, el cual se usa de forma habitual en la empresa para muchos otros servicios. Por tanto, en este punto no hay un claro ganador.

- **Soporte Oficial**: Aquí se ha de estudiar si hay algún tipo de soporte de pago oficial por parte de las propias compañías. Rancher utiliza precisamente esto como modelo de negocio, dado que no cobra por la utilización de su

software. Sin embargo, dado que OKD es un proyecto de la comunidad, no existe una compañía que acepte pagos por su consultoría. Sin embargo, tras investigar por la red, se concluyó que RedHat acepta habitualmente dar soporte a los proyectos realizados con OKD. Rancher, por tanto, se corona en este punto por ofrecer el soporte de manera oficial.

- **Soporte de la Comunidad**: Dado que este es un punto más bien poco preciso a la hora de comparar, se realizó un pequeño estudio basado en números que podrían indicar el estado actual de la comunidad. Se obtuvieron los números de las preguntas en StackOverflow etiquetadas con ambas tecnologías, el número de *issues* abiertos y cerrados en GitHub y, en el caso de Rancher, se contabilizaron los posts publicados en su foro privado de soporte de la comunidad. Además, y para tenerlo en cuenta, se añadió (indicando las diferencias) los números relacionados con Openshift y Openshift Origin (el nombre antiguo de OKD, que en el momento de la realización del proyecto seguía siendo nombrado de esta manera por un sector de la comunidad). Los resultados llevaron a pensar que Rancher no solo tenía una comunidad más activa que OKD, sino que además tenía un foro en el cual varios trabajadores del proyecto se encargaban de solucionar diferentes dudas. Esto, por tanto, hace que Rancher se lleve este punto a su favor.

	Rancher	OKD	Openshift (Partially compatible)
Questions on StackOverflow	1,365	268 + 793 (Origin)	14,706
Open issues on GitHub	2,224	270	-
Closed issues on GitHub	18,806	8,471	-
Private Forum Topics	1,374		

Tabla 2.2: Comparativa de soporte de la comunidad

- **Otras aplicaciones**: Con respecto a las tecnologías que se querían ejecutar en el clúster, y que se querían ofrecer como servicios resilientes, se hizo una investigación previa para comprobar si había una implementación ya hecha o documentada. Dado que en ninguno de los casos se encontró, se concluyó que no había ninguna ventaja en ninguna de las tecnologías.

Tras hacer una comparativa punto por punto, al final se debatió cuáles de estos eran totalmente imprescindibles o tenían más peso, y se llegó a la conclusión de que OKD 3.11 era la decisión más adecuada para el proyecto entre manos.

CAPÍTULO 3
Diseño

Cabe mencionar que este proyecto no se trata de una implementación en el vacío. Se trata de crear un sistema que se adapte a una tecnología ya existente y con unas limitaciones claras y bien delimitadas.

Se explicará a continuación, pues, una serie de apartados importantes en los que se hizo incapié y cómo se desarrolló su diseño, así como el porqué de las decisiones tomadas.

3.1 *Hardware*

Se pretende desplegar un clúster de Kubernetes en local, sin hacer uso de servicios web de computación como los ofrecidos por Google, Microsoft o Amazon. Para ello se deberá estudiar qué tipo de *hardware* sería necesario para poder ejecutar los diferentes procesos que hacen que el clúster se mantenga en marcha y pueda correr los servicios deseados.

Estos nodos se compondrán de una mezcla de máquinas locales sin virtualizar, conectadas a la red local de la empresa, y máquinas virtuales desplegadas con el servicio de VSphere por parte del departamento de IT, de manera indistinta en la mayoría de casos, y guiándose por las preferencias y necesidades del departamente de IT para proveer dichos recursos.

Con el *software* a utilizar ya decidido para desplegar el clúster de Kubernetes (OKD), lo primero a realizar es buscar en la propia documentación cuáles son los requisitos mínimos de hardware. Se puede observar que los requisitos varían según el tipo de máquina. Para ello, vamos a explicar qué diferentes funciones puede realizar cada uno de los nodos y qué requisitos de hardware precisan.

3.1.1. Maestros

Los nodos maestros son los encargados de ejecutar el *software* que mantiene en funcionamiento y administra el clúster. En el número de ellos que se utilice reside la clave de tener un sistema a prueba de errores de hardware. Si se quiere conseguir que haya una alta disponibilidad y resiliencia a fallos inesperados causados por la parada de un nodo maestro, se recomienda utilizar más de una

máquina maestra. Estas máquinas deben estar estrechamente en contacto, pues se repartirán el trabajo y se adjudicarán la carga si alguna de ellas falla.

Cabe destacar que es muy importante que estos nodos estén conectados entre sí de la manera más eficaz posible, en una red de área local rápida con la mínima latencia entre ellos. Aunque una manera de reducir la latencia y el tiempo de conexión pudiese ser crearlos como máquinas virtuales de la misma máquina, esto debe evitarse, pues si esta falla, todas las instancias virtualizadas caerán con ella. Por ello, usar diferentes máquinas físicas debería ser una prioridad a la hora de asegurar el continuo servicio de los nodos maestros. Además, una serie de medidas de seguridad se tendrían que tener en cuenta, como que no fuesen alimentadas por el mismo cableado eléctrico (para evitar fallos con el suministro de energía), o que se encontrasen en habitaciones diferentes (en caso de incendio, robo, o similar).

Según la propia documentación, además, los nodos maestros deben disponer de los siguientes requisitos mínimos para poder servir al clúster en cuando a componentes:

- Mínimo de 4 vCPU (aunque se recomiendan fuertemente más).
- Mínimo de 16 GB RAM (aunque se recomiendan más, especialmente si se pretende co-localizar el etcd).
- Mínimo de 40 GB de disco duro para el sistema de archivos que contenga `/var/`.
- Mínimo de 1 GB de disco duro para el sistema de archivos que contenga `/usr/local/bin/`.
- Mínimo de 1 GB de disco duro para el sistema de archivos que contenga el directorio temporal del sistema.

3.1.2. Trabajadores

Los nodos maestros se encargan de todo aquello que no tiene que ver con gobernar el clúster. Son los que albergan los programas y ejecuciones que el plano de control les ordena. Por ello, no necesitan tener unos mínimos de hardware tan estrictos como los maestros.

Una ventaja de estos nodos es que normalmente pueden encontrarse mucho más separados del resto de nodos del clúster, sin que esto suponga un impedimento mayor para el correcto funcionamiento del conjunto. Esto, por supuesto, también depende del tipo de ejecuciones que se vayan a realizar, pues un nodo cuyos contenedores ejecuten procesos que requieran de mucho acceso al sistema de almacenamiento podría verse afectado por disponer de una conexión menos fuerte.

Según la documentación, los requisitos son:

- 1 vCPU.

- Mínimo 8 GB RAM.
- Mínimo de 15 GB de disco duro para el sistema de archivos que contenga `/var/`.
- Mínimo de 1 GB de disco duro para el sistema de archivos que contenga `/usr/local/bin/`.
- Mínimo de 1 GB de disco duro para el sistema de archivos que contenga el directorio temporal del sistema.
- Un mínimo adicional de 15 GB de espacio sin anexar para el correcto funcionamiento de Docker.

3.1.3. Nodos etcd

Los nodos etcd se encargan de mantener una capa de almacenamiento conjunta entre todos los nodos maestros, en la cual pueden almacenar información relevante con respecto al funcionamiento y organización. Cabe destacar que no es estrictamente necesario que estos nodos sean nodos independientes. De hecho, en el modelo a utilizar, se decidió que había un mayor beneficio en aplicarle estas funciones a los nodos maestros, siguiendo una de las prácticas habituales a la hora de crear un clúster de Kubernetes.

De esta manera, aunque nuestros nodos maestros ya tienen una serie de especificaciones requeridas por parte de sus funciones como maestros, hay que tener en cuenta que, según la documentación, han de ser capaces de cumplir además las siguientes espectativas:

- Mínimo de 20 GB de almacenamiento de disco duro para los datos del etcd.

3.1.4. Máquinas de almacenamiento persistente

En este proyecto se necesitarán máquinas que funcionen de manera muy similar a los nodos maestros, pero para encargarse de manejar y supervisar el almacenamiento de los datos que se puedan generar en los procesos ejecutados por los nodos trabajadores.

Estos nodos serán utilizados por el tándem de las ejecuciones de los servicios Rook y Ceph, y como requisitos mínimos tendrán los de la figura 3.1.

Dado que estos requisitos están sacados de la página web oficial y no se han descrito todavía en este documento las diferencias entre cada uno de los procesos que especifican, cabe decir que al final se llegó a la decisión de que se usarían unas máquinas que estaban libres de uso en el momento y que cumplían ampliamente las necesidades que iba a tener la prueba de concepto.

Process	Criteria	Minimum Recommended
ceph-osd	Processor	• 1x 64-bit AMD-64 • 1x 32-bit ARM dual-core or better • 1x i386 dual-core
	RAM	~1GB for 1TB of storage per daemon
	Volume Storage	1x storage drive per daemon
	Journal	1x SSD partition per daemon (optional)
	Network	2x 1GB Ethernet NICs
ceph-mon	Processor	• 1x 64-bit AMD-64/i386 • 1x 32-bit ARM dual-core or better • 1x i386 dual-core
	RAM	1 GB per daemon
	Disk Space	10 GB per daemon
	Network	2x 1GB Ethernet NICs
ceph-mds	Processor	• 1x 64-bit AMD-64 quad-core • 1x 32-bit ARM quad-core • 1x i386 quad-core
	RAM	1 GB minimum per daemon
	Disk Space	1 MB per daemon
	Network	2x 1GB Ethernet NICs

Tabla 3.1: Requisitos mínimos de Ceph

3.1.5. *Launcher*

Cabe destacar que, aunque sea durante un tiempo muy pequeño, habrá que utilizar una máquina auxiliar para levantar el clúster. A esta máquina la llamaremos *Launcher*, Lanzador o Controlador de Ansible.

Esta máquina solo tiene un requisito: Tener una instalación de Ansible actual y poder dedicarle 75 MB de RAM.

Esta máquina se encargará de instalar el resto de requisitos *software* en los nodos del clúster, como Docker y las configuraciones adecuadas de IP y similar.

3.1.6. Máquina de Compilación

Se debe decir que, además, una sugerencia por parte de los empleados que supervisaban el proyecto fue añadir como nodo trabajador una máquina normalmente reservada para compilaciones pesadas por parte de otro grupo de la empresa.

Dado que esto consistía en simplemente añadir un kubelet y ser capaz de asignársele trabajos y tareas a través de los maestros de Kubernetes, se decidió implementarla en el diseño inicial, como se podrá ver en los diagramas de diseño en los que se muestren las partes específicas de hardware utilizadas.

3.2 Software

Una vez los requisitos hardware han sido complimentados, hemos de tener en cuenta las necesidades de *software* de nuestro clúster y de cada uno de los nodos, según su tipo.

No hará falta, eso sí, dividir este apartado en más subapartados, como en la sección anterior, pues los requisitos son prácticamente idénticos. Todo tipo de nodo que forme parte del clúster ha de utilizar como Sistema Operativo:

- Fedora 21
- CentOS 7.5
- Red Hat Enterprise Linux (RHEL) 7.5 con la opción de instalación "Minimal" y los últimos paquetes del canal de Extras, es decir, del repositorio de *CentOS Extras*, uno de los adicionales de CentOS
- RHEL Atomic Host 7.4.5 o superior. Si se decide usar RHEL, se debe utilizar una de las siguientes versiones mínimas de kernel:
  - RHEL 7.5: 3.10.0-862.31.1
  - RHEL 7.6: 3.10.0-957.27.2
  - RHEL 7.7: 3.10.0-1062

3.3 Integración en el *workflow* existente

Como ya se ha descrito antes, este proyecto en ningún momento estuvo pensado como algo a realizar en un vacío, sino como una nueva rama a explorar para poder añadir a un sistema que actualmente funciona y está en marcha.

Es por eso que, para saber qué ha de realizarse para poder integrarlo correctamente, hay que estar al tanto de exactamente cómo funciona dicho sistema ya existente. Para ello, en este punto se va a explicar cómo funciona a día de hoy y cómo se diseñó y organizó este proyecto para integrarse de la forma menos invasiva y más eficiente posible.

3.3.1. Workflow existente

En el grupo de *Quality Assurance* es de vital importancia ser capaz de generar tests automáticos que aseguren el correcto funcionamiento de los productos que se pretenden evaluar. Para ello, el equipo trabaja en la creación de pruebas que puedan ser replicadas una y otra vez en diferentes versiones, asegurándose así de que las diferentes iteraciones del proyecto futuras funcionan también con lo anteriormente comprobado. Este tipo de pruebas reciben el nombre de "*tests* de regresión".

Hace falta, por tanto, un amplio volumen de capacidad de cómputo para realizar todas estas pruebas, y una lógica suficientemente avanzada como para poder

decidir qué *tests* se realizan y cuáles no. Esto es así porque, en ocasiones, hay *tests* que tienen menos importancia inmediata que otros, y conviene acotar cuáles de estas pruebas se ejecutarán.

Para solucionar el primer punto, el de la necesidad de cómputo, el equipo tiene a su disposición una serie de ordenadores que están conectados entre sí por la red interna de la empresa. Cada una de estas máquinas tiene entre 1 y 8 sistemas operativos corriendo, así que al final el poder de cálculo de este laboratorio es notable. A cada una de estas instancias de sistema operativo se le llama internamente un "banco de *test*".

Estos bancos de test son utilizados y organizados por una instancia de Jenkins, un *software* de CI/CD que es capaz de crear y organizar *workflows* para su ejecución automática. Esta instancia no se ejecuta en ninguno de los bancos, sino en un servidor diferente (aunque también dentro de la red interna de la empresa).

Además de estos bancos de trabajo, hay algunas tareas, como las de compilación, que se ejecutan en un recurso específico que se denomina la máquina de compilación, y que actualmente también se ve gobernada por la instancia de Jenkins.

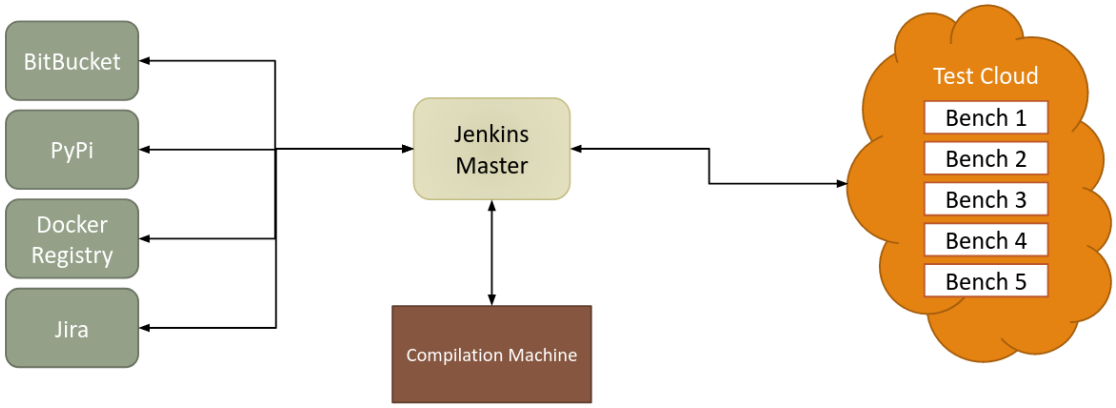

Figura 3.1: Configuración actual

Por supuesto, Jenkins no es el único encargado de todos los procesos que se ejecutan. Hay una serie de servicios auxiliares que le dan soporte, los cuales se integran todos en lo que se conoce como el *Test Cloud*. Estos servicios son transparentes para el proyecto y no requieren de una explicación detallada, pues son principalmente sistemas que permiten modificar el número de bancos y asegurarse de que están en funcionamiento.

3.3.2. Integración de Kubernetes

Aquí es precisamente donde entra en juego este proyecto. Al tratarse de una única instancia dirigiendo todos los trabajos que se ejecutan en las diferentes máquinas, si esta deja de funcionar de manera inesperada (ya sea por error humano, apagón inesperado o simplemente errores de software), los resultados de los *tests* se ven comprometidos. Puede que no les llegue la tarea completa y por tanto su ejecución sea innecesaria, pudiendo devolver datos erróneos; o que los resultados

no puedan ser devueltos porque la instancia de Jenkins es incapaz de recolectar los datos.

Kubernetes, como ya se ha convenido antes, es una herramienta que funciona extremadamente bien en entornos en los que se quiere mantener la ejecución constante de instancias de vital importancia, siendo capaz de monitorizarla y mantenerla activa. Aquí es, por tanto, donde se implementaría un clúster de Kubernetes.

Dado que se pretende mantener activa una instancia de Jenkins en todo momento, trasladar su ejecución a un nodo que esté controlado por Kubernetes supone que su *up-time* (tiempo activo) sea superior y tenga una mayor disponibilidad, evitando así la gran mayoría de fallos por imprevistos.

No se especifica que esto sea lo único que se integre en el clúster de Kubernetes. Además de la instancia de Jenkins, otros servicios importantes han de ser eficazmente integrados. Entre estos servicios se encuentran:

- Una instancia de **PyPi**: Esto es un repositorio de paquetes de Python. Por necesidades del *workflow*, este repositorio también ha de estar en continuo acceso, por lo cual es un candidato ideal para verse sometido al cambio de paradigma que es ser controlado por el clúster de Kubernetes.

- Una instancia de registro de Docker, en la cual se guardarán las imágenes de las que pueda requerir cualquier servicio que se planee del clúster.

- Un servicio **Rook** en tándem con **Ceph**: Aunque ya se ha explicado en cierta medida en el apartado de tecnologías empleadas, cabe decir que este servicio es necesario para proporcionar una capa de almacenamiento al clúster, incluyendo las instancias de Jenkins y PyPi.

- Un servicio de Ingress: Dado que los posibles servicios desplegados en el clúster pueden disponer de servicios de *front-end* web a los que pueda acceder el usuario, hace falta algún tipo de controlador o balanceador de carga que se encargue de redirigir el tráfico a los puertos adecuados de los contenedores adecuados. Por tanto, este servicio ha de ser *cluster-aware* (es decir, ser consciente del clúster) y tomar decisiones basándose en los cambios que dentro del clúster se puedan producir. Como ejemplo más directo, Jenkins dispone de una interfaz web a la que se accede a través de una dirección URL local, a través de un puerto expuesto en la máquina en la que se ejecute. Dado que esta es una parte vital del *workflow* de muchos empleados, se ha de proporcionar un acceso consistente y transparente para ellos.

- Dockers de compilación y división: Una de las tareas que se propuso trasladar al clúster de Kubernetes es el trabajo combinado de dos ejecuciones realizadas en un contenedor cada una. Uno se encargaría de dividir diferentes *tests* en *tests* más pequeños o unitarios, y otro se encargaría de realizar compilaciones necesarias por otros grupos de la empresa. Ambos contenedores deberían ejecutarse en la máquina de compilación, por petición y diseño del equipo para facilitar la integración con los otros grupos.

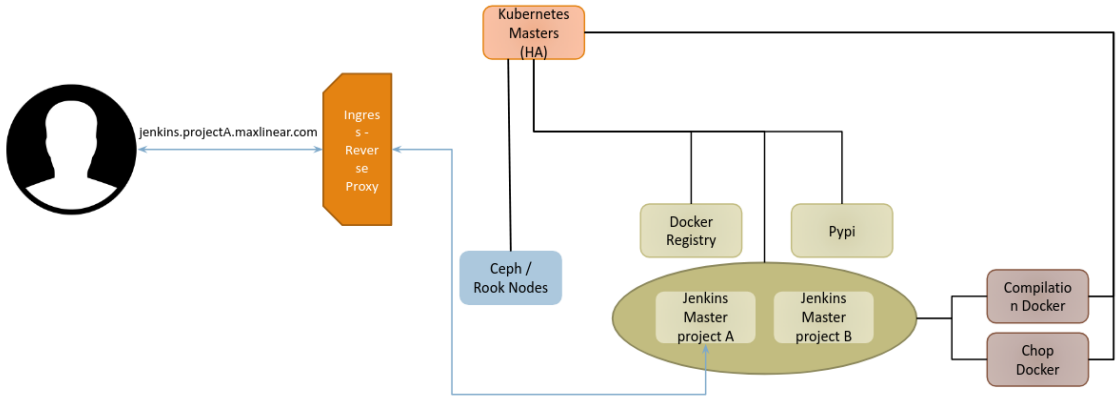

Figura 3.2: Acceso a GUI de Jenkins por Ingress

De esta manera, se puede mostrar con un gráfico cuáles de los servicios que se usan actualmente se insertarán en el clúster de Kubernetes, como se ve en la figura 3.3

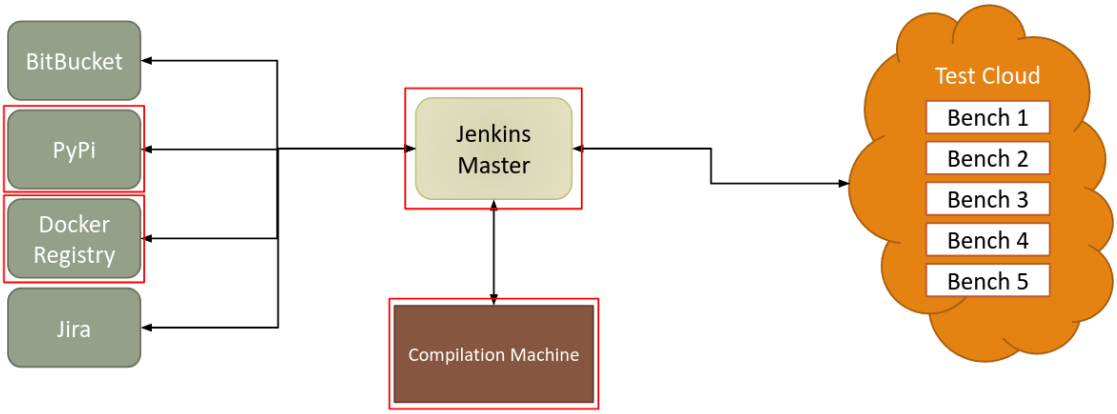

Figura 3.3: Partes a introducir en el clúster

No solo se consideran los diferentes productos *software* a ejecutar, sino también las máquinas y nodos en los que se realizarán estas ejecuciones. Es por eso que se ha de especificar de qué se va a disponer y en qué máquinas va a correr cada servicio.

De esta manera, el diseño final debería quedar como en la figura 3.4.

Si este proyecto llegase a integrarse de forma efectiva, se podría considerar una integración mayor, pudiendo llegar a modificar el *workflow* del equipo e integrando ciertas tareas del Test Cloud actual como tareas y procesos de Kubernetes. Esto incluiría, por ejemplo, integrar los bancos de trabajo como nodos propios de Kubernetes, unificando así en mayor medida el clúster y simplificando, potencialmente, algunos de los procesos.

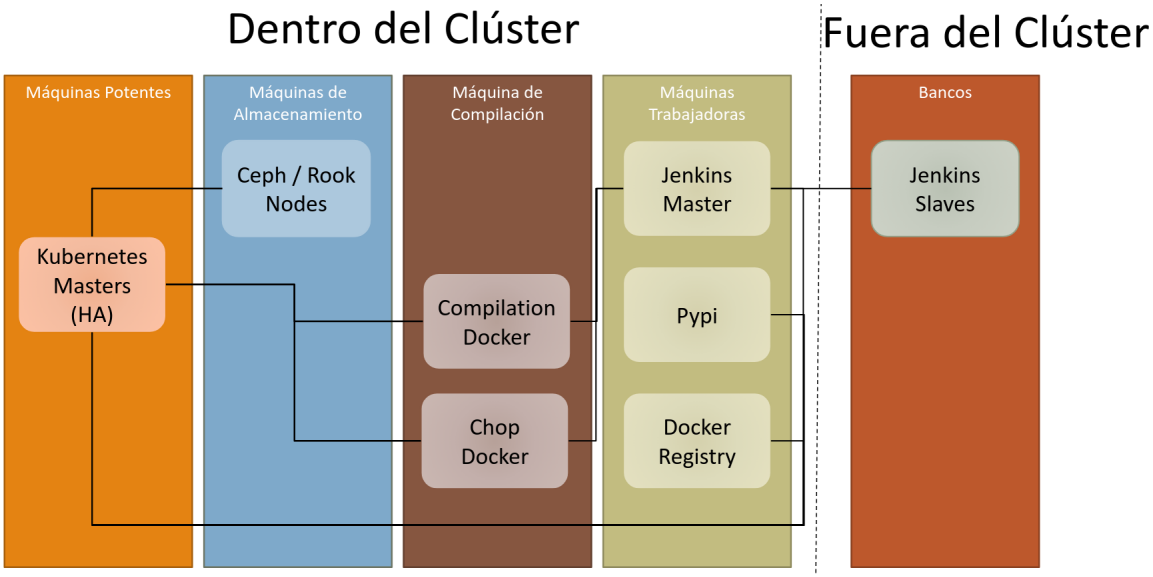

Figura 3.4: Configuración final

CAPÍTULO 4
Resultados y Discusión

Este apartado se va a centrar en el propio desarrollo del proyecto en sí, desde los primeros pasos que se dieron cómo se fueron solucionando los diferentes problemas.

Dado que el proyecto resultó ser muy complicado en algunos aspectos, y plagado de errores de compatibilidad y despliegue poco documentados que se iban teniendo que solucionar sobre la marcha, este punto puede leerse casi como un cuaderno de bitácora que recoge los problemas con los que ha habido tropiezos y cómo se han logrado o intentado solucionar.

4.1 Pasos previos

Durante la primera sección del proyecto, había que estudiar las tecnologías, así que alrededor de dos meses fueron utilizados para aprender cómo funcionaba Kubernetes puro y recopilar información sobre las posibles ofertas que el mercado ofrecía.

Esta formación se obtuvo a través de varias fuentes de internet, ya fuesen documentaciones oficiales, cursos de pago en plataformas de educación independientes (Udemy), charlas impartidas por los propios trabajadores y desarrolladores de varios proyectos de interés, y foros tanto propios como extraoficiales de las tecnologías en cuestión.

Al tratarse de la primera fase, se podría considerar más un primer acercamiento que un desarrollo estricto del proyecto, pues la tecnología todavía era desconocida y había que absorber tanta información como fuese posible.

Es aquí donde se realizan pequeños ejercicios rudimentarios de manejo y creación de proyectos locales de Docker. Para ello se utilizó el Docker *engine* ofrecido por la propia compañía para facilitar la virtualización y manejo de contenedores.

Una vez se comprendió cómo funcionaba Docker, el paso a Kubernetes primero también fue teórico, y se fue solapando con la búsqueda de una solución empresarial que escoger.

Aquí es donde entra en juego la discusión y comparación ya expuesta en el punto 2.3. Es aquí donde se realiza el trabajo de investigación que lleva a propo-

ner las opciones al resto de compañeros, junto con una rúbrica y permitir que una decisión global fuese obtenida.

Para tomar una decisión informada, por supuesto, se hicieron pequeñas pruebas de concepto, prácticamente *tests* de comprobación de que eran tecnologías adecuadas. Vamos, a continuación, a observar los resultados de dichas pruebas.

4.1.1. Rancher

Al tratarse de la opción que más se vendía como la puerta de entrada o *newbie-friendly* (adecuada para novatos) de las estudiadas, la primera prueba de ejecución se hizo con esta tecnología.

Rancher dispone de una distribución de Kubernetes que facilita la creación de un clúster llamada RKE. Su instalación es muy sencilla y consiste en la creación de un archivo de inventario que contiene la información del clúster y la ejecución de un comando para comenzar el despliegue.

Descripción general de los apartados del inventario En su página web [6] dan ejemplos de cómo sería esta configuración en su mínimo exponente (creando un único nodo que haga todas las funciones del clúster) y con un ejemplo de un clúster completo A. En esta última se ve cómo se pueden modificar muchos de los aspectos del clúster desde el primer despliegue. Vamos, pues, a mostrar el primer ejemplo a continuación, y explicar los puntos más importantes o interesantes que se proveen en el ejemplo.

```
1 nodes:
2 - address: 1.2.3.4
3 user: ubuntu
4 role:
5 - controlplane
6 - etcd
7 - worker
```

Este en un ejemplo básico que sirve como prueba de funcionamiento y ayuda a verificar la instalación. A continuación, los diferentes puntos explicados:

- **nodes**: En este apartado, cada uno de los *items* que se muestren se corresponderán con la configuración individual y propia de cada uno de los que vayan a ser los nodos del clúster. En este caso, como ya se ha dicho, se trata de un único nodo que realiza todas las funciones de un clúster entero, así que solo se halla listado uno.
- **address**: Aquí se especifica la dirección IP de acceso del nodo en cuestión que se quiera utilizar como parte del clúster y, por tanto, que forme parte de la instalación. En este ejemplo es una que muy probablemente sea inválida, dado que no es más que una muestra, y deberá cambiarse para cumplir las necesidades del nodo en cuestión.
- **user**: El usuario que se defina aquí será el usuario desde el cual se realizarán las tareas de instalación de todo el *software* que requiera Rancher para

funcionar, así como el usuario que se encargará de realizar las configuraciones necesarias para poder formar parte del clúster. Se recomienda, por tanto, que el usuario en cuestión tenga los permisos necesarios para poder hacer todas estas tareas, ya sea concediéndole privilegios de superusuario o buscando una solución alternativa.

- **role**: En este apartado se le pueden asignar una serie de tareas ya definidas en grupos a cada uno de los nodos de manera individual, dejando que la lógica interna de RKE se encargue de organizarlo. Dado que aquí están representados los 3 roles principales que se necesitan en un clúster (recordemos que este nodo ha de hacer todos los trabajos de uno), podemos definirlos a partir de este ejemplo:

- **controlplane**: Este rol se le ha de asignar a los que sean los nodos maestros, los encargados de que funcione el clúster y de comunicarle al resto de nodos qué se ha de hacer y cómo se deben dividir las tareas.

- **etcd**: Los nodos a los que se les asigne este rol serán los encargados de organizar y proporcionar a los nodos del rol anterior todo el espacio que necesiten para el correcto funcionamiento del clúster. Proporcionan almacenamiento en el cual se guarda información que el plano de control requiere, ya sea de manera temporal o indefinida.

- **worker**: Este debería ser el rol más común en un clúster desplegado en producción. Los nodos con este rol será los encargados de ejecutar las tareas que se les asigne desde el plano de control, y deberán estar en contacto con el mismo para poder cumplir su función.

Una vez explicados los puntos más simples, se pueden poner algunos ejemplos de ciertas configuraciones que podrían ser necesarias en algunos de los casos que se quieran realizar. Para ello se tomará como ejemplo el mostrado en la documentación de Rancher, el completo, pero escogiendo solo los puntos más llamativos o importantes. Todo esto se podrá ver de manera más amplia en el anexo A.

```
1 ignore_docker_version: false
```

ignore_docker_version: Este apartado se ha de utilizar en caso de que, por alguna razón, exista una necesidad de utilizar una versión de Docker diferente a la que espera RKE. Si no se cambia el valor a *true*, una vez RKE encuentre un problema de versión, la instalación y despliegue del clúster entero pararán y se tendrá que hacer el despliegue de nuevo desde cero.

```
1 cluster_name: mycluster
```

cluster_name: Un campo autoexplicativo, pero que es importante de completar sobre todo si se planeasen desplegar varios clústers a la vez.

```
1 ssh_key_path: ~/.ssh/test
```

ssh_key_path: Este apartado es de vital importancia cuando se realiza el despliegue. Se debe proporcionar una clave ssh que tenga acceso a todos los usuarios

que queremos utilizar de cada una de las máquinas. Se puede especificar de forma global o de manera individual por nodo, pudiendo proporcionar claves ssh específicas si hiciesen falta por razones de seguridad en algunos de los equipos.

```
authorization:
mode: rbac
```

rbac: Ya se ha explicado lo que significa RBAC (*Role Based Access Control*) en el punto 2.3, pero conviene destacar que requiere menos pasos y es un proceso más intuitivo - al menos para alguien sin experiencia previa - de configurar, pues los ajustes seleccionados de manera predeterminada ofrecen este tipo de control de usuarios *out-of-the-box*.

```
ingress:
provider: nginx
node_selector:
app: ingress
```

nginx: El servicio de ingress que propone Rancher es NGINX, y funciona al menos en un entorno local de pruebas sin realizar ninguna configuración (a no ser que se tope con problemas de compatibilidad con la red en la que se despliega).

Instalación del clúster de prueba Una vez explicados los puntos más importantes y que se llegaron a tratar en la puesta en marcha de práctica, se procederá a explicar qué es lo que se utilizó para su despliegue.

Se creó una máquina virtual en el mismo equipo que iba a servir como *launcher* para un clúster de prueba *all-in-one* (todo en uno), que significa que realiza todas las tareas del clúster en un único nodo. Esto se ha podido ver en el archivo de inventario de ejemplo que se ha explicado en este mismo punto. Se estableció que esta máquina virtual utilizaría el sistema operativo CentOS, pues se podía emplear como sistema operativo tanto en esta solución como en OKD, y la elección aún no estaba hecha.

Para poder hacer el despliegue, lo más sencillo era utilizar la versión de Docker, y así se hizo aunque fuese simplemente para revisar de primera mano cómo era la interfaz gráfica. Para ello, se utilizó el comando:

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

Este comando preparó un entorno delimitado a un contenedor Docker al cual se podía acceder mediante el puerto 443 de localhost. La interfaz, por tanto, se mostraba similar a la figura 4.1.

Cuando se terminó de realizar este paso, se facilitaron por parte de la empresa tres máquinas virtuales en red para hacer un despliegue de 3 nodos que hiciesen todas las funciones del clúster teniendo *High Availability*. Estos nodos fueron dispuestos de CentOS, y se procedió a realizar una instalación.

Para crear un archivo de inventario, en lugar de hacerlo de manera manual totalmente, se utilizó una de las funciones que se integran en el *software* de RKE. Después de instalar el ejecutable siguiendo los pasos de la web, se ejecutó el comando encontrado en la documentación [9]:

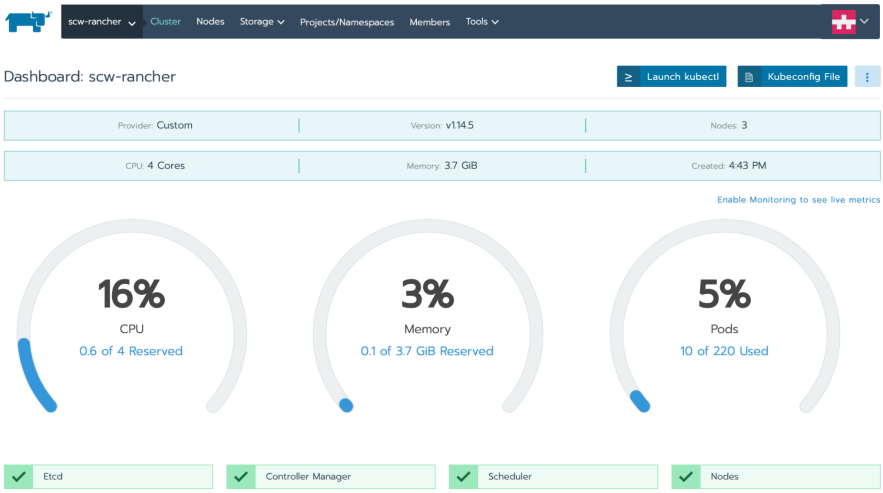

Figura 4.1: Dashboard de Rancher

```
rke config
```

Este comando básicamente ejecuta un proceso interactivo que guía al usuario a la hora de crear una configuración válida de inventario, y termina devolviendo un archivo preparado para ser ejecutado por el siguiente comando:

```
rke up
```

Este comando se encarga de manejar y ejecutar el despliegue del servidor, creando todas las configuraciones necesarias. Por desgracia, el proceso no fue tan sencillo como podría parecer tras leer la documentación, y esto se tradujo en una serie de errores que tuvieron que ser arreglados.

Los puertos necesarios para el correcto funcionamiento del clúster estaban configurados previamente al despliegue del mismo, así que se produjeron varios errores de red que hubo que solventar. Por suerte, para ello el equipo recomendó simplemente eliminar las configuraciones previas que pudiesen existir, dado que la imagen que se había utilizado para hacer los despliegues de los nodos era una que posiblemente tuviese configuración específica para proyectos anteriores y, dado que en este paso se estaba haciendo una pequeña prueba de concepto, era más sencillo eliminar esos errores en pos de probar el software. Para ello, se utilizó el comando:

```
sudo iptables -F
```

Esto arreglaba el error que daba, en concreto, el puerto 6443, que aparecía como *filtered* (filtrado) al ejecutar un comando de diagnóstico de red como:

```
nmap -sS -p 6443 <IP>
```

Hay que tener en cuenta que hay que introducir la IP del nodo que se quiera investigar, en concreto de aquellos que den problema y errores a la hora de desplegarlos.

Una vez eso se solucionó, un error del archivo de la Interfaz de Red de Contenedores (*CNI*, del inglés *Container Network Interface*) surgió. No era necesariamen-

te un error de configuración, sino más bien de creación de archivos. Por alguna razón que no se pudo identificar, en los despliegues no era consistente la creación del archivo adecuado en todos los nodos. El archivo a veces se generaba y otras no, pero nunca se generaba de manera errónea. Para solucionar esto, el método más sencillo consistía en copiar la configuración que se hubiese desplegado correctamente en uno de los nodos y pegarla en aquellos en los que no se hubiesen generado. La ruta de creación de este archivo se trataba de:

```
/etc/cni/net.d/<nombre_del_archivo>
```

El nombre del archivo podía cambiar, pero normalmente comenzaba por el número 80 o 10. En caso de duda, y dado que todas las máquinas habían sido desplegadas a partir de la misma imagen y habían recibido los mismos cambios, saber el nombre del archivo era tan sencillo como hacer una comparativa entre los archivos de un nodo sano y uno fallido. Este método conseguía solucionar parcialmente el problema, pues el nodo volvía a estar sano y era reconocido por el resto del clúster, aunque fallaba a la hora de poderle mandar órdenes de ejecuciones. Estaba en un estado *unschedulable*, es decir, que no podía ser utilizado como nodo trabajador.

Evidentemente, este remedio no era más que un parche para tapar un error en la generación de la configuración de red, y se decidió que en lugar de investigar más a fondo qué lo estaba generando, se utilizase este *workaround* para intentar avanzar y conseguir información que se consideraba más útil para el proyecto.

Una vez estos errores fueron solucionados y se pudo hacer un clúster de prueba (aunque algo inestable), se decidió probar entonces la otra opción de software, OKD.

4.1.2. OKD

Una vez esa prueba fue completada, el siguiente paso era probar OKD. Entre las razones para dejar esta prueba para después, se encuentra el público a quien se ofrece cada una de las dos opciones. Mientras que Rancher se enorgullece de ser una solución sencilla para aquellos que quieran comenzar con la tecnología de Kubernetes, y poder expandir su valor y complejidad con el tiempo conforme lo entiendan más, OKD es una solución orientada a empresas creada por una de las empresas más respetables de la industria a nivel profesional.

Por eso mismo, la prueba de OKD se atrasó hasta haber entendido, a nivel práctico, cómo funcionaba Kubernetes. Una vez se comenzó a hacer pruebas con OKD, se crearon otras 3 máquinas virtuales en línea de CentOS, siguiendo los requisitos mínimos del sistema que ya se han especificado.

Gracias a la práctica con Rancher y RKE, el concepto de crear un archivo de inventario ahora era mucho más sencillo de comprender, y no hizo falta ninguna utilidad como el comando `rke config` de RKE que guiase en la creación del mismo. Simplemente seguir la documentación, entender los ejemplos y modificarlos según las necesidades del proyecto fue necesario.

Aunque el propósito del inventario es el mismo que con Rancher, el formato es diferente. A continuación se estudiará y explicarán los puntos principales de uno

de los archivos de ejemplo de un inventario de OKD, en concreto de un único nodo que haga de clúster completo, aunque algo modificado. Esto se hará así para explicar los apartados más basicos y para mantener cierta consistencia en la explicación con lo ya hecho.

Para ver el inventario completo, se puede encontrar anexado en el apéndice A.

Descripción general de los apartados del inventario Un inventario de clúster de Kubernetes OKD tiene varias secciones que dividen el documento y que se pueden identificar fácilmente gracias a su forma. Cada uno de estos apartados recibe un nombre concreto que debe ser obtenido de la documentación. Para que sea un formato válido, además, tiene que estar escrito entre corchetes []. De esta manera nos podemos encontrar los siguientes apartados:

```
[OSEv3:children]
masters
nodes
etcd
```

children: En este apartado hemos de poblar las líneas con el nombre de los grupos que vamos a querer proporcionar al clúster al ejecutar el comando de creación o modificación del mismo. En este caso de ejemplo podemos ver cómo están los tres tipos principales de nodo: masters, nodes y etcd.

En algunos casos, como ampliar el clúster actual, se añaden grupos nuevos como new_masters. Si es el caso de, por ejemplo, querer añadir un *LoadBalancer* (Un balanceador de carga), se deberá añadir el grupo lb.

```
[OSEv3:vars]

ansible_ssh_user=root

ansible_become=true

openshift_deployment_type=origin

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login':
    'true', 'challenge': 'true', 'kind': '
   HTPasswdPasswordIdentityProvider'}]
```

vars: Este apartado es posiblemente el más complejo de todo. Es en el que se estipulan los valores de las diferentes variables que harán que podamos configurar el clúster según las necesidades del cúster. Vienen en el ejemplo las siguientes cuatro opciones, aunque algunas de ellas comentadas para evitar que realmente sean parte efectiva de la configuración. Para su explicación, sin embargo, se encuentran aquí descomentadas.

- *ansible_ssh_user*: Este apartado es el análogo de *user* en la configuración de Rancher. Se trata del nombre de usuario que se ha de utilizar para instalar el *software* necesario y las configuraciones adecuadas en todos los nodos. Hay que tener en cuenta de que se trata de un único usuario que ha de servir en todos los nodos, así que se recomienda para el despliegue de un clúster de

OKD que se cree un usuario con el mismo nombre y privilegios en todos los nodos.

- *ansible_become*: Esta variable ha de estar activada si se pretende usar algún método para dar privilegios de superusuario al usuario definido en la variable anterior. Cabe destacar que hay varios métodos posibles para proporcionar dichos permisos, pero el que se encuentra seleccionado por defecto es sudo. Esta variable se pasa al programa de Ansible como método nativo (es decir, que ya dispone Ansible de la opción *become* sin necesitar configuración propia de OKD), y este *software* se encarga de utilizar el método seleccionado en los nodos a desplegar.

- *openshift_deployment_type*: Esta variable sirve para diferenciar entre una instalación con Openshift y una con OKD. Como ya se ha nombrado anteriormente, OKD no es más que el nombre con el que se rebautizó a Openshift Origin, la versión de la comunidad del *software* Openshift de RedHat. Es por eso que, por razones históricas, esta variable se ha de poner en origin para informar al *software* de instalación de que se pretende desplegar la versión de comunidad.

- *openshift_master_identity_providers*: Si este apartado se decide comentar o ignorar, el servicio de RBAC (*Role Based Access Control*) se vería controlado por HTPasswd en modo permisivo, con lo cual cualquier combinación de usuario y contraseña serían válidas para entrar al clúster. En cambio, al seleccionar este configuración, se buscará el archivo HTPasswdPasswordIdentityProvider en /etc/origin/master/ para proveer una configuración correcta de usuarios, contraseñas y permisos.

```
1 [masters]
2 master.example.com
3
4 [etcd]
5 master.example.com
```

masters y *etcd*: Estos apartados requieren de cierta preparación. No pueden utilizarse si no han sido declarados previamente en el apartado *children*. De la misma manera, declarar el uso de alguno de estos en el apartado *children* pero luego no hacer uso de los mismos supone un error a la hora de realizar el despliegue. Aquí se han de declarar las IPs o nombres de *host* mediante los cuales se puede acceder a los nodos que se quieren utilizar en cada uno de los grupos. Dado que en este ejemplo queremos un clúster íntegramente corriendo en un único nodo, tanto *masters* como *etcd* comparten nodo.

```
1 [nodes]
2 master.example.com openshift_node_group_name='node-config-all-in-one'
```

nodes: Aquí se han de exponer todos los nodos que se vayan a utilizar, ya hayan sido especificados en otro de los apartados (como *masters* o *etcd*) o no. Además, se le asignará un nombre de grupo que viene con una serie de características asociadas. Estos grupos están predeterminados y los hay de varios tipos, según se quieran asignar.

En este caso, se utiliza el nombre de grupo node-config-all-in-one, indicando que va a realizar todas las tareas al tratarse de un clúster AIO.

Instalación del clúster de prueba Para proceder al despliegue del clúster de prueba se utilizó una de las máquinas de CentOS previamente creadas para este propósito con los requisitos adecuados cumplidos. En ella se instaló el ejecutable ofrecido por OKD, oc, que es una versión modificada y ampliada del cliente de terminal nativo de Kubernetes, kubectl. A este nodo se le instalará, además, la versión mas actual de Ansible o, en caso de que diese errores de compatibilidad, la 2.6.

La información obtenida para este punto se ha extraído de la documentación oficial de OKD [10].

Para ello hubo que descargarse el binario desde la página oficial de OKD y asegurarse que se sitúa o en un directorio actualmente perteneciente al PATH o se adecúa que al PATH se añada la carpeta en la que se encuentra descargado.

Más tarde se ha de clonar el repositorio de openshift/openshift-ansible, en concreto la versión 3.11, que es la que se estudia en este proyecto. Aunque no es necesario descargar el proyecto entero, dado que simplemente se utilizará la carpeta de playbooks, es recomendable hacerlo, pues hay inventarios de ejemplo, configuraciones ya creadas para diversos casos de uso y otros documentos que pueden ser válidos si se quiere buscar información.

Una vez esto está hecho, se han de completar dos ejecuciones enteras de Ansible. La primera se utiliza para asegurarse de que el nodo cumpla los requisitos previos del clúster, y por tanto se habrá de ejecutar el comando:

```
1 ansible-playbook -i /ruta/al/inventario /ruta/de/repo/playbooks/
    prerequisites.yml
```

Este proceso puede durar más de 20 minutos, dependiendo de la velocidad de la máquina empleada o de la configuración que se realice. Dado que los *playbooks* de Ansible los hemos obtenido directamente de OKD al clonar el repositorio de GitHub, no debería hacer falta ningún tipo de modificación, aunque es buena idea seguir la traza en caso de que suceda algún error. Una vez este proceso dé una lista de pruebas superadas, se puede pasar a la segunda ejecución de Ansible:

```
1 ansible-playbook -i /ruta/al/inventario /ruta/de/repo/playbooks/
    deploy_cluster.yml
```

Este comando se encargará de desplegar el clúster en el nodo, y cuando se acabe la instalación se podrá acceder al clúster con el comando:

```
1 oc get nodes
```

Esto nos dará una lista de un solo nodo y nos deberá decir que su *STATUS* es *Ready*. Si es el caso, se podrá acceder a la interfaz web a través del link https://master.openshift.com:8443/console. Cabe decir que este comando asume que master.openshift.com es la ruta de acceso al nodo que se está utilizando. Hay que tener en cuenta, además, que es muy probable que el navegador que se use para acceder a dicha ruta avise de que se trata de un sitio no seguro. Esto se debe a un problema con el certificado https, pero no es un problema que sea necesario arreglar para

este proyecto, pues el único uso que se le iba a dar era interno por parte de la empresa.

Maneras alternativas de crear un clúster AIO Aunque OKD no provea un *software* o *script* que automatice la creación de un inventario desde el principio como sí lo hace RKE, un (antiguo) empleado de Openshift creó junto a la comunidad un repositorio de Git con herramientas de automatización de dicho ejemplo.

No es un tema que vaya a ser tratado extensamente, pero sí que cabe destacar su uso previo a la creación de inventarios propios basándose en ejemplos y documentación oficial de OKD. El repositorio de GitHub en cuestión es gshipley/installcentos, y actualmente ha sido movido a okd-community-install/installcentos.

4.2 Puesta en marcha del proyecto con OKD

Una vez las pruebas estaban realizadas en un solo nodo tanto con OKD como Rancher, y habiendo seleccionado OKD como la opción a utilizar, se pretendía desplegar un clúster de 3 nodos maestros, un *LoadBalancer*, 3 nodos trabajadores que se dedicasen exclusivamente a ejecutar las funciones de Rook y Ceph, y un nodo trabajador de prueba.

Cabe decir que todos los nodos se redesplegaron con la imagen inicial para poder evitar problemas con configuraciones hechas en los pasos previos que pudiesen interferir.

Con el sistema de hardware ya puesto en marcha y montado, se procederá a explicar los puntos que se realizaron y los posibles problemas que acarrearon.

4.2.1. SSH Keys

Lo primero que se hizo fue preparar el acceso por clave SSH a todos los nodos por parte del *launcher*. Para ello se creó una clave SSH que permitiese al *launcher* conectarse sin que se le pidiese contraseña.

El primer comando que se ejecutó creo una clave SSH única nueva que se pudiese compartir con el resto de nodos:

```
1 ssh-keygen -t rsa
```

Simplemente se han de seguir los pasos que te pide el pequeño *script* interactivo, pero preferiblemente no usar frase cuando la pida.

Después de eso, se debe copiar la clave pública en cada uno de los nodos que van a pertenecer al clúster. Para ello, se usa el comando:

```
1 ssh-copy-id -i ~/.ssh/<clave publica> <usuario>@<nombre de host>
```

Se recomienda utilizar un *script* para automatizar este proceso, posiblemente con un simple bucle for en bash o utilizando Ansible y una de las recetas disponibles en el repositorio de *Ansible Galaxy* [8], como la llamada *sshd*.

4.2.2. SELinux

Así pues, uno de los primeros pasos consiste en activar SELinux. Como ya se ha explicado, SELinux es un *software* de protección del sistema que categoriza con etiquetas todos los archivos del mismo, y que evita el acceso o ejecución de aquellos que tengan una etiqueta que lo impida o no la tenga en absoluto.

En las instalaciones que se tenían de CentOS, SELinux no estaba activado, y esto era un requisito para la instalación de OKD. Para activarlo durante la sesión se puede utilizar el comando

```
setenforce 1
```

Esto hará que hasta que se reinicie el orenador, SELinux se halle activado. Sin embargo, esta no es la única configuración que se exige de SELinux. Además, debe estar en modo *Enforcing*. Si está en modo *Permissive* puede dar errores aunque a priori parezca funcionar con total perfección.

Y además, el comando anterior solo modifica su ejecución durante la sesión actual, y no ofrece un cambio permanente. Es por eso que la manera de modificar esta configuración es mediante cambios en el archivo de opciones de SELinux. En los equipos que se utilizaron, la ruta que le pertenecía era `/etc/selinux/config`.

La configuración final debe ser tal que así:

```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
# enforcing = SELinux security policy is enforced.
# permissive = SELinux prints warnings instead of enforcing.
# disabled = No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
# targeted = Targeted processes are protected,
# mls = Multi Level Security protection.
SELINUXTYPE=targeted
```

Una vez esta configuración ha sido aplicada, se puede proceder a reiniciar el nodo y comprobar que todo funcione.

Por desgracia, esto no funcionó a la primera, y de hecho supuso un gran quebradero de cabeza a la hora de instalar OKD. Esto se debe a que, en los nodos provistos por la empresa, SELinux se hallaba instalado de manera incompleta o errónea, y el sistema de etiquetas no abarcaba todos los archivos. Esto se traduce en que, a la hora de conectarse por SSH a uno de los nodos que tuviese SELinux activado, se recibía un error que sugería que no se tenía acceso a `/bin/bash`, y que por tanto no se podía ejecutar una sesión de bash.

El error era tal porque el ejecutable bash no tenía una etiqueta asociada por SELinux y, al pretender ejecutarlo desde una máquina externa, el propio SELinux bloqueaba el acceso. De esta manera, y sin poder acceder al terminal, el nodo quedaba inservible a distancia. Por desgracia, se trataban de máquinas virtuales, y no se podía conectar un teclado y pantalla físicos para poder solucionar el error si no era haciendo uso de la ayuda de IT, y eso se estimó como poco apropiado dado que no disponían de tiempo suficiente.

De esta manera, la solución más sencilla era redesplegar el equipo y probar de nuevo. Esto llegó a pasar en repetidas ocasiones, y se convirtió en un auténtico bloqueo. Aunque, como ya se ha comentado antes, se consiguió circunvalar, al menos temporalmente, utilizando la opción permissive en lugar de enforcing. Por desgracia, esto volvió a dar problemas a la hora del despliegue y una solución adecuada tuvo que ser encontrada.

Para ello, se buscaron formas de reetiquetar el sistema de archivos de los nodos, y varias fuentes de internet sugerían la creación de un archivo vacío con un nombre específico en la raíz del sistema. Esto haría que SELinux lo detectase durante el inicio del sistema operativo y se dedicase al reetiquetado. El comando para la creación de dicho archivo es:

```
sudo touch /.autorelabel
```

Por desgracia, este método no funcionó, o mejor dicho, lo hizo de forma muy inconstante, provocando bloqueos nuevos de varias máquinas. Así pues, se buscó otra solución y se hallaron los siguientes comandos:

```
restorecon -R -v /
sudo fixfiles relabel
```

Estos comandos se encargaban de recuperar el etiquetado por defecto de todo el sistema de archivos. Aunque se trata de un comandos que llevan varios minutos de ejecución, gracias a ellos se consiguió sanear el sistema de etiquetado de todo el sistema, permitiendo así volver a la opción de enforcing en SELinux.

Cabe destacar que, en un intento por evitar el redespliegue de los nodos, se creó una tarea en cron que periódicamente, cada día a diferentes horas, activaba el modo permissive de SELinux, para poder volver a acceder de forma remota por SSH a máquinas que hubiesen sido bloqueadas. Este comando fue:

```
crontab -e 0 7,18 * * * /usr/sbin/setenforce 0
```

Aunque al final se consiguió evitar este problema, cabe destacar que esta es una solución que se podría utilizar en caso de que el problema aparezca de manera repetida.

El comando puede ser modificado según las normas generales de cron para que en vez de ejecutarse dos veces al día, se ejecute según lo requiera el usuario. Si se requiere algún tipo de ayuda, crontab.guru es una buena web para previsualizar las configuraciones temporales de cron [11].

4.2.3. Docker

Una de las impresiones que se tenían era que la instalación de Docker debía ser realizada previamente en los nodos. Por ello, en los nodos se instaló la versión más reciente de Docker. Por desgracia, y tras investigar en la documentación de OKD, se comprobó que la versión más actual que permitía era la 1.13.1.

De esta manera, se procedió a instalar la versión en cuestión, siguiendo los pasos de la web oficial de Docker para instalar versiones antiguas. Más tarde, sin embargo, se observó que no era necesario hacer una instalación previa, ya que los propios *playbooks* de Ansible se encargaban de instalar la versión adecuada.

Un problema que apareció y persistió tanto instalando Docker a mano como dejando que lo hiciesen los *playbooks* fue un problema de compatibilidad entre Docker y SELinux.

Mientras que OKD recomendaba utilizar Overlay2 como *driver* de almacenamiento de Docker, si esta opción era la seleccionada, el despliegue del clúster fallaba. En cambio, si se elegía *devicemapper* como el *driver* de almacenamiento, SELinux daba problemas de incompatibilidad.

Esto se podía parchear utilizando una opción en el archivo de configuración de almacenamiento de Docker. Para ello, había que modificar el archivo `/etc/sysconfig/docker-storage` añadiendo a la sección de opciones la opción:

```
1 --selinux-enabled=false
```

Aunque esto parecía arreglar la situación, ocurrió algún error en los pasos siguientes que pueden llevar a pensar que este parche causaba más problemas de los que solucionaba. Además, para automatizar su configuración en cada despliegue, se tenía o bien que utilizar un *script* personalizado, o modificar uno de los *playbooks* de instalación de OKD.

En retrospectiva, un posible error sería que el sistema no estuviese preparado correctamente con lvm2, y que por tanto *devicemapper* no fuese capaz de funcionar adecuadamente.

4.2.4. Desactivar swap

Un paso que se nombra en la documentación es la desactivación de cualquier swap que se pueda utilizar en cualquier nodo del clúster. Aunque durante la ejecución de este proyecto no se halló ningún error aun sin utilizar siempre sistemas con el swap desactivado, realizar este paso no es más que un comando en cada nodo y podría incluso añadirse a los *playbooks* de instalación, dado que Ansible es un *software* de uso muy variado y para nada exclusivo para esta aplicación.

Dicho comando es:

```
1 swapoff -a
```

Este comando debería funcionar en cualquier distribución de Linux, no necesariamente en CentOS de manera exclusiva.

4.2.5. RBAC

Dado que en este despliegue sí se pretende utilizar el acceso basado en roles, hay que preparar el inventario para poder utilizarlo de manera correcta. Para ello, lo primero que hay que hacer es instalar *htpasswd*, que se encuentra dentro del paquete *httpd-tools* que se puede instalar desde yum en CentOS.

Una vez esté instalado el paquete, se puede crear un documento en el cual se guarde la información del RBAC del clúster, y añadir un usuario predeterminado. Para ello se ha de urilizar este comando:

```
1 htpasswd -c /etc/origin/master/htpasswd <nombre_de_usuario>
```

Si se provee la información que pide el *script* interactivo, se debería tener un archivo ya preparado para que el despliegue pueda funcionar, siempre y cuando se añada la línea relacionada con RBAC que se ha explicado en el apartado en el que se explican los diferentes puntos del archivo de inventario.

4.2.6. Load Balancer

Dado que se pretende hacer un clúster con múltiples másters, se debe incluir un nodo que haga la función de *Load Balancer*. Para ello se añade al archivo de inventario lo siguiente:

```
[OSEv3:children]
#  ...
lb

[OSEv3:vars]
openshift_master_cluster_hostname=lb.example.com
openshift_master_cluster_public_hostname=lb.example.com

[lb]
lb.example.com
```

Véase que el nombre de *host* es un mero ejemplo y que se utilizó el nombre adecuado. De igual forma,

4.2.7. Despliegue

Al igual que en el despliegue de prueba, tras crear el inventario y dejarlo todo preparado, se tiene que proceder al despliegue. El proceso es el mismo que en la prueba, así que no se explicará de nuevo, pero sí se analizarán los errores.

El error más preocupante que apareció tenía que ver con el CIDR. El CIDR (*Classless Inter-Domain Routing*), o Enrutamiento Interdominio sin Clases, se encarga de asignar un número de IPs disponibles a cada uno de los nodos, y en este caso se refería a un problema de falta de IPs disponibles o conflicto con IPs ya ocupadas.

Cada nodo requiere no solo una IP propia de acceso en la red local, sino también un rango de IPs que pueda asignar libremente a los procesos que coran dentro de él.

Esto es así porque el clúster de Kubernetes estaba, por diseño, integrado en la red local de la compañía. De esta manera, una gran parte de los rangos de IPs que se querían utilizar ya estaban siendo utilizados en mayor o menor medida y, al tratarse de un nuevo clúster que iba a pertenecer a la red local, IT debía asignarle un rango de IPs que no entrase en conflicto.

Por tanto, se le pidió al departamento de IT que escogiese un rango de IPs libre y lo proporcionase. Esto llevó tiempo y no fue una solución sencilla, teniendo que indagar en qué hacía exactamente la tarea de Ansible que fallaba. Por desgracia, dado que la red de empresa estaba relativamente saturada, no se pudo ofrecer rango lo suficientemente cómodo como para acoger un gran número de IPs. Esto se convirtió en un bloqueo que se intentó sobrepasar eliminando el número de

nodos de manera temporal, para asegurar que ese error se podía eliminar en un entorno más definitivo.

Junto a este, apareció también un error similar a uno de los ocurridos con Rancher. Algunos de los nodos no presentaban el archivo de configuración de CNI que se debe crear para formar parte del clúster. Se intentó solucionar de la misma manera, copiando de los nodos sanos a los nodos no funcionales el archivo de configuración.

Por desgracia, esto solo funcionó de manera parcial, haciendo que el nodo apareciese como parte del clúster, incluso con el estado de READY al hacer el comando oc get nodes, pero a la hora de intentar que ejecutasen tareas, fallaban en programarse. Esto se achacó a un error de red que no se pudo encontrar, por más que se intentó. Lo más probable fuese que se tratase de configuraciones erróneas de red.

Además de esto, cabe destacar que en ocasiones el despliegue fallaba por errores temporales de DNS debido a reconfiguraciones automáticas dirigidas por el *software* Puppet. En otras ocasiones, el repositorio local estaba fuera de alcance y los paquetes necesarios para el despliegue no estaban disponibles. Esto, de todas formas, sucedió en contadas ocasiones.

También, y aunque sea un error menor, merece ser nombrado, al principio algunos de los nodos daban errores por no tener suficiente RAM o espacio libre, pero según la información que se pudo obtener en foros y otros lugares de internet, no eran condiciones estrictas y podían ser suprimidas para continuar con el despliegue a pesar de no cumplir con las especificaciones más convenientes.

Para suprimir estos errores, se han de especificar en el archivo de inventario, mediante una variable, qué errores y avisos se han de ignorar. En este caso, la línea a añadir era:

```
[OSEv3:vars]
# ...
openshift_disable_check=memory_availability,disk_availability
```

Esto evita que el despliegue falle al saltar un error de este tipo, y permite que se prosiga con simplemente un mensaje de aviso.

4.3 OKD 4

Dado que el tiempo de desarrollo establecido para el despliegue del clúster había sido casi totalmente consumido, y juntándose con problemas que no se podían predecir, como la pandemia causada por la COVID19, se decidió que el proyecto debía pararse apenas dos semanas antes de que se acabase la beca dedicada al proyecto.

Esto fue una decisión tomada por el equipo con la intención de seleccionar tecnologías hacia aquellas que quizá diesen menos problemas, dado que no se había encontrado una solución concreta a los errores que se estaban mostrando al utilizar OKD 3.11.

Durante el desarrollo del proyecto, apareció la beta de OKD 4, y actualmente es la versión del *software* que se anuncia en la web y a la cual redirige la documentación. Es por estas razones que se decidió hacer un pequeño estudio sobre cómo de probable sería, en un futuro, utilizar esta tecnología en lugar de la ya abandonada 3.11.

Este trabajo consistió en intentar levantar un clúster de un solo nodo, como se había hecho en las dos pruebas de concepto anteriores, aunque en esta ocasión no hubo éxito.

La tecnología estaba todavía algo prematura, la manera de despliegue era muy diferente a la de las versiones anteriores y la documentación todavía estaba algo incompleta, con la desventaja de que no podía verse fortalecida por la documentación de versiones menores anteriores, dado que el cambio en el código era mayor.

CAPÍTULO 5
Conclusiones

La tecnología ofrecida por Kubernetes es, desde luego, muy capaz y potente. Se trata de un producto software que no solo facilita el crecimiento horizontal de servicios, sino que además proporciona una mayor disponibilidad de los archivos y ejecuciones que maneja.

Se trata, también, de una tecnología desarrollada y mantenida inicialmente por una de las más grandes compañías tecnológicas del mundo, como es Google, y ahora se trata de un proyecto libre avalado por la CNCF y la *Linux Foundation*.

No solo eso, sino que además varias compañías han decidido aportar sus propios puntos de vista, ofreciendo aún más posibilidades para usar el software de la forma que más le convenga a cada proyecto.

Esto hace que Kubernetes sea una tecnología muy importante y que deba ser considerada por toda empresa que necesite cierto poder de computación.

Por desgracia, al tratarse de un producto tan completo, su facilidad de uso no es enorme. Es cierto que entender a nivel conceptual cómo funciona puede ser sencillo para alguien que haya estudiado o conozca conceptos de informática y computación, pero la complejidad a la que opera Kubernetes puede hacer que su despliegue y mantenimiento sean muy complejos.

Esto no se debe necesariamente a la manera en que Kubernetes se despliega o mantiene, sino a la enorme cantidad de pequeños detalles que pueden fallar y suponer un bloqueo tras otro, convirtiéndose en pequeños pozos de tiempo en los que hay que invertir horas, si no días, en solucionarlos. Un sistema tan intricado es algo que una empresa que realmente vaya a hacer uso de dicha tecnología merece, pero no es un proyecto que se tenga que tomar a la ligera.

Se debe estudiar con tiempo y preparar el camino para que su desarrollo sea lo más simple posible. Intentar crear un clúster en una red ya ocupada puede llevar consigo una enorme serie de problemas, y no disponer de equipos limpios con los que poder probar diferentes configuraciones puede acotar en gran medida la capacidad de solución de problemas de un equipo dedicado a su despliegue.

OKD, en concreto, es un proyecto muy interesante y claramente potente. RedHat es una empresa que lleva mucho tiempo en el sector y proporciona soluciones profesionales. Además, el hecho de utilizar la versión gratuita de la comunidad significa que se puede hacer uso del extenso conocimiento de la gente que ya ha tenido errores similares a los que puedan aparecer en un nuevo despliegue,

o incluso haya hecho tutoriales o artículos detallando cómo hacer determinadas cosas.

A pesar de esto, hay ciertas tecnologías que mantienen anclado a OKD, al menos en su versión 3.11. Docker es, evidentemente, un gran sistema de contenedores, pero se utiliza una versión anticuada como es la 1.13. Además de eso, el hecho de que el único sistema en que funcione de manera gratuita y soportada sea CentOS también disminuye la capacidad de adaptación del software, obligando a que las máquinas que se usen en el clúster no puedan ser utilizadas para nada más que requiera un sistema operativo diferente.

Otra tecnología que requiere de mucha atención y que usa OKD es SELinux. Ciertamente es una gran herramienta de protección adicional que ofrece CentOS, y que cuadra totalmente con uno de los aspectos clave que se quieren cumplir si se tiene una red de ordenadores: la seguridad. Eso no quita que sea un software más bien obtuso y difícil de entender, y que puede llevar a errores graves. En este proyecto, en concreto, supuso uno de los mayores bloqueos durante los primeros intentos de instalar un clúster de OKD en máquinas remotas.

La documentación de OKD es a ratos brillante y a otros casi ausente. En muchos casos, de hecho, la información se haya en la documentación de versiones anteriores, y no hay un análogo directo en las versiones más actuales. Tienen grandes ejemplos de archivos de configuración e inventarios, pero en pocas ocasiones todas las opciones que se pueden utilizar son mostradas claramente al usuario.

Otro problema que se le puede atribuir a la tecnología de Kubernetes en general es su orientación a la computación en la nube. Una inmensa parte de la documentación y artículos de Kubernetes y todas sus distribuciones está orientada a cómo conseguir que un clúster sea funcional en servicios de computación web como son Microsoft Azure, Google Cloud o Amazon Web Services. Esto deja en un segundo plano a la instalación en máquinas locales o *baremetal*, y muchas veces hace que la configuración de las mismas sea más difícil de lo que podría parecer porque muchos de los errores que podrían suceder se ven arreglados de forma automática con plugins o integraciones creadas por empresas para que Kubernetes trabaje en tándem casi perfecto con dichos servicios web.

No quiere decir esto que la creación de un clúster en la web sea extremadamente sencilla, sino más bien que es un objetivo claro de este producto *software*.

Esto genera el debate sobre cómo de plausible resultaría que la aceptación por parte de diferentes negocios de la computación en la nube pública creciese, y si es conveniente o no dependiendo de cada caso. Evidentemente hay situaciones en las que, por ley relativa a los datos que trata dicho negocio, es imposible albergar estos servicios en la nube. Pero también están las situaciones en las que la simple comodidad de desplegar el clúster en mucho menos tiempo, con capacidad de ampliación en el futuro de manera sencilla y pagando únicamente por aquellas ejecuciones realizadas, suponen un beneficio a tener en cuenta con respecto a poder utilizar equipos en desuso o tener acceso físico a los recursos.

Como recomendación a MaxLinear Inc. o cualquier otro usuario o empresa que quiera replicar este proyecto, daría una serie de consejos:

- **Minimizar errores de entorno**: La fuente de problemas más frecuente ha sido la configuración de red, así que intentar utilizar una red cerrada y no utilizada por ningún sistema más sería una gran idea para comenzar.
- **Estudiar bien las tecnologías requeridas**: Dado que hay una serie de productos que son requeridos por la configuración mostrada en este proyecto, es conveniente entender no sólo el porqué se pide su uso, sino también entender cómo funcionan. De esta manera, errores como el producido por SELinux se podrían solucionar mucho más rápido de lo que se hicieron.
- **Considerar usar recursos en la nube**: Muchos de estos problemas se solucionarían si la instalación del clúster hubiese sido decantada hacia la web, en lugar de utilizar recursos locales. Aunque evidentemente esto no siempre es posible, es importante hacer un buen balance para comprobar si esto compensaría.

En definitiva, Kubernetes es un gran, gran producto de *software* que debe ser considerado por todas aquellas empresas que realmente quieran poseer tecnología de estado del arte, pero no es un proyecto al que haya que entrar con poca preparación o con expectativas de dedicarle poco tiempo. Además debe considerarse mucho la orientación web que tiene esta tecnología, y si eso es algo que se prefiere o se pretende evitar.

Bibliografía

[1] Resource Containers: A New Facility for Resource Management in Server Systems Gaurav Banga, Peter Druschel *Rice University* Jeffrey C. Mogul Western Research Laboratory, Compaq Computer Corp.

[2] Proyectos que abala la Linux Foundation. Consultado en https://www.linuxfoundation.org/projects/.

[3] Empresas en las que se ha utilizado la tecnología Kubernetes. Consultado en https://kubernetes.io/case-studies/.

[4] Tipos de contenedores a los que Kubernetes da soporte. Consultado en https://kubernetes.io/docs/setup/production-environment/container-runtimes/.

[5] Documentación de Kubernetes para desplegar un Ingress. Consultado en https://kubernetes.io/docs/concepts/services-networking/ingress/.

[6] Página web con inventarios de ejemplo de un clúster de Rancher. Consultado en https://rancher.com/docs/rke/latest/en/example-yamls/.

[7] Documentación de Kubernetes donde se explican y enumeran los diferentes conceptos que lo crean. Consultado en https://kubernetes.io/docs/concepts/.

[8] Repositorio de recetas de Ansible realizadas por la comunidad y por empresas. Se encuentra en https://galaxy.ansible.com/.

[9] Web con las instrucciones de descarga en instalación del binario de RKE. Se encuentra en https://rancher.com/docs/rke/latest/en/installation/#download-the-rke-binary.

[10] Documentación oficial de OKD indicando los pasos para inslatar el la CLI. Se encuentra en https://docs.okd.io/3.11/cli_reference/get_started_cli.html#cli-reference-get-started-cli.

[11] Man page de cron. Consultada en https://linux.die.net/man/5/crontab.

APÉNDICE A
Inventario completo de ejemplo de Rancher

```
nodes:
    - address: 1.1.1.1
      user: ubuntu
      role:
        - controlplane
        - etcd
      ssh_key_path: /home/user/.ssh/id_rsa
      port: 2222
    - address: 2.2.2.2
      user: ubuntu
      role:
        - worker
      ssh_key: |-
        -----BEGIN RSA PRIVATE KEY-----

        -----END RSA PRIVATE KEY-----
    - address: example.com
      user: ubuntu
      role:
        - worker
      hostname_override: node3
      internal_address: 192.168.1.6
      labels:
        app: ingress

# If set to true, RKE will not fail when unsupported Docker version
# are found
ignore_docker_version: false

# Cluster level SSH private key
# Used if no ssh information is set for the node
ssh_key_path: ~/.ssh/test

# Enable use of SSH agent to use SSH private keys with passphrase
# This requires the environment `SSH_AUTH_SOCK` configured pointing
#to your SSH agent which has the private key added
ssh_agent_auth: true

# List of registry credentials
# If you are using a Docker Hub registry, you can omit the `url`
# or set it to `docker.io`
```

```
# is_default set to 'true' will override the system default
# registry set in the global settings
private_registries:
     - url: registry.com
       user: Username
       password: password
       is_default: true

# Bastion/Jump host configuration
bastion_host:
    address: x.x.x.x
    user: ubuntu
    port: 22
    ssh_key_path: /home/user/.ssh/bastion_rsa
# or
#   ssh_key: |-
#     -----BEGIN RSA PRIVATE KEY-----
#
#     -----END RSA PRIVATE KEY-----

# Set the name of the Kubernetes cluster
cluster_name: mycluster


# The Kubernetes version used. The default versions of Kubernetes
# are tied to specific versions of the system images.
#
# For RKE v0.2.x and below, the map of Kubernetes versions and their system images is
# located here:
# https://github.com/rancher/types/blob/release/v2.2/apis/management.cattle.io/v3/k8s_defaults.go
#
# For RKE v0.3.0 and above, the map of Kubernetes versions and their system images is
# located here:
# https://github.com/rancher/kontainer-driver-metadata/blob/master/rke/k8s_rke_system_images.go
#
# In case the kubernetes_version and kubernetes image in
# system_images are defined, the system_images configuration
# will take precedence over kubernetes_version.
kubernetes_version: v1.10.3-rancher2

# System Images are defaulted to a tag that is mapped to a specific
# Kubernetes Version and not required in a cluster.yml.
# Each individual system image can be specified if you want to use a different tag.
#
# For RKE v0.2.x and below, the map of Kubernetes versions and their system images is
# located here:
# https://github.com/rancher/types/blob/release/v2.2/apis/management.cattle.io/v3/k8s_defaults.go
#
# For RKE v0.3.0 and above, the map of Kubernetes versions and their system images is
# located here:
```

```
# https://github.com/rancher/kontainer-driver-metadata/blob/master/rke/k8s_rke_system_images.go
#
system_images:
    kubernetes: rancher/hyperkube:v1.10.3-rancher2
    etcd: rancher/coreos-etcd:v3.1.12
    alpine: rancher/rke-tools:v0.1.9
    nginx_proxy: rancher/rke-tools:v0.1.9
    cert_downloader: rancher/rke-tools:v0.1.9
    kubernetes_services_sidecar: rancher/rke-tools:v0.1.9
    kubedns: rancher/k8s-dns-kube-dns-amd64:1.14.8
    dnsmasq: rancher/k8s-dns-dnsmasq-nanny-amd64:1.14.8
    kubedns_sidecar: rancher/k8s-dns-sidecar-amd64:1.14.8
    kubedns_autoscaler: rancher/cluster-proportional-autoscaler-amd64:1.0.0
    pod_infra_container: rancher/pause-amd64:3.1

services:
    etcd:
      # if external etcd is used
      # path: /etcdcluster
      # external_urls:
      #   - https://etcd-example.com:2379
      # ca_cert: |-
      #   -----BEGIN CERTIFICATE-----
      #   xxxxxxxxxx
      #   -----END CERTIFICATE-----
      # cert: |-
      #   -----BEGIN CERTIFICATE-----
      #   xxxxxxxxxx
      #   -----END CERTIFICATE-----
      # key: |-
      #   -----BEGIN PRIVATE KEY-----
      #   xxxxxxxxxx
      #   -----END PRIVATE KEY-----
    # Note for Rancher v2.0.5 and v2.0.6 users: If you are configuring
    # Cluster Options using a Config File when creating Rancher Launched
    # Kubernetes, the names of services should contain underscores
    # only: 'kube_api'.
    kube-api:
      # IP range for any services created on Kubernetes
      # This must match the service_cluster_ip_range in kube-controller
      service_cluster_ip_range: 10.43.0.0/16
      # Expose a different port range for NodePort services
      service_node_port_range: 30000-32767
      pod_security_policy: false
      # Add additional arguments to the kubernetes API server
      # This WILL OVERRIDE any existing defaults
      extra_args:
        # Enable audit log to stdout
        audit-log-path: "-"
        # Increase number of delete workers
        delete-collection-workers: 3
        # Set the level of log output to debug-level
        v: 4
    # Note for Rancher 2 users: If you are configuring Cluster Options
    # using a Config File when creating Rancher Launched Kubernetes,
    # the names of services should contain underscores only:
```

```
148     # 'kube_controller'. This only applies to Rancher v2.0.5 and v2
            .0.6.
149     kube-controller:
150       # CIDR pool used to assign IP addresses to pods in the cluster
151       cluster_cidr: 10.42.0.0/16
152       # IP range for any services created on Kubernetes
153       # This must match the service_cluster_ip_range in kube-api
154       service_cluster_ip_range: 10.43.0.0/16
155     kubelet:
156       # Base domain for the cluster
157       cluster_domain: cluster.local
158       # IP address for the DNS service endpoint
159       cluster_dns_server: 10.43.0.10
160       # Fail if swap is on
161       fail_swap_on: false
162       # Set max pods to 250 instead of default 110
163       extra_args:
164         max-pods: 250
165       # Optionally define additional volume binds to a service
166       extra_binds:
167         - "/usr/libexec/kubernetes/kubelet-plugins:/usr/libexec/
              kubernetes/kubelet-plugins"
168
169 # Currently, only authentication strategy supported is x509.
170 # You can optionally create additional SANs (hostnames or IPs) to
171 # add to the API server PKI certificate.
172 # This is useful if you want to use a load balancer for the
173 # control plane servers.
174 authentication:
175     strategy: x509
176     sans:
177       - "10.18.160.10"
178       - "my-loadbalancer-1234567890.us-west-2.elb.amazonaws.com"
179
180 # Kubernetes Authorization mode
181 # Use 'mode: rbac' to enable RBAC
182 # Use 'mode: none' to disable authorization
183 authorization:
184     mode: rbac
185
186 # If you want to set a Kubernetes cloud provider, you specify
187 # the name and configuration
188 cloud_provider:
189     name: aws
190
191 # Add-ons are deployed using kubernetes jobs. RKE will give
192 # up on trying to get the job status after this timeout in seconds..
193 addon_job_timeout: 30
194
195 # Specify network plugin-in (canal, calico, flannel, weave, or none)
196 network:
197     plugin: canal
198
199 # Specify DNS provider (coredns or kube-dns)
200 dns:
201     provider: coredns
202
203 # Currently only nginx ingress provider is supported.
204 # To disable ingress controller, set 'provider: none'
```

```
# 'node_selector' controls ingress placement and is optional
ingress:
    provider: nginx
    node_selector:
      app: ingress

# All add-on manifests MUST specify a namespace
addons: |-
    ---
    apiVersion: v1
    kind: Pod
    metadata:
      name: my-nginx
      namespace: default
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80

addons_include:
    - https://raw.githubusercontent.com/rook/rook/master/cluster/
      examples/kubernetes/rook-operator.yaml
    - https://raw.githubusercontent.com/rook/rook/master/cluster/
      examples/kubernetes/rook-cluster.yaml
    - /path/to/manifest
```