Trabajo fin de máster

Máster Universitario en Ingeniería Informática

Reconfiguración dinámica de redes de laboratorios

Autenticación y monitorización con Kubernetes

Departamento de Sistemas de Comunicación y Control
Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Educación a Distancia
Curso 2019-2020

Autor: **Álvaro Téllez Rodríguez**

Director: **Antonio Robles Gómez**

Codirector: **Rafael Pastor Vargas**

Madrid, octubre 2020

“*Hay que tener la mente abierta. Pero no tanto como para que se te caiga el cerebro al suelo.*”
(Richard Feynman)

Agradecimientos

Quiero empezar agradeciendo a Antonio Robles y a Antonio Uzal su apoyo, sus consejos y sus palabras de ánimo durante la realización de este trabajo. Creo que el proyecto en el que estáis es muy interesante y puede dar grandes resultados.

También quiero tener palabras de agradecimiento para mi madre, mi hermano y mis sobrinas, por estar siempre ahí y por esas videollamadas tan amenas de estos últimos meses. Y por supuesto, a mi abuelo Miguel, al que lamentablemente no podré enseñarle este trabajo, pero al que le habría hecho ilusión verlo terminado.

Por último, quisiera agradecerle a mi pareja, Katerina, el gran apoyo que me ha dado durante las épocas de más carga de trabajo en estos últimos años, y sobre todo en estas últimas semanas, lo que me ha permitido que yo pudiese dedicarme más a fondo a este trabajo. Ευχαριστώ πολύ, κουτσούνι μου!

¡Gracias!

Resumen

La irrupción de las infraestructuras y plataformas como servicio han supuesto una revolución en muchos ámbitos, y abre un mundo de nuevas posibilidades que antes simplemente no eran posible. Si además se combinan con miniordenadores de reducido tamaño y peso, pero buenas prestaciones, nos da como resultado la posibilidad de poder crear clústeres de ordenadores de un tamaño inferior a la torre de un ordenador de sobremesa, pero con un gran potencial. Y, si se plantean este tipo de clústeres para actividades o aplicaciones de laboratorio, nos da como resultado una herramienta muy potente para la práctica y formación de alumnos. Pensando en esto, el Proyecto eNMoLabs está desarrollando y gestionando una serie de trabajos que permitan poner en marcha un *Lab of Things* basado en servicios, un laboratorio con diferentes funcionalidades integradas y que pueda ser accesible de forma remota para trabajar en él.

Es dentro de este Proyecto donde se enmarca este trabajo, en el que crearemos un clúster de kubernetes sobre una serie de dispositivos de bajo coste llamados Raspberry Pi. Sobre este clúster configuraremos una aplicación que permita la autenticación y autorización de usuarios por medio de las credenciales de la universidad, y con una gestión de permisos que aseguren que los usuarios autenticados estén asignados a los grupos que correspondan, que a su vez estos grupos se corresponden con roles para controlar los permisos de lo que podrán hacer en el clúster. Esta capa de seguridad se hará siguiendo uno de los estándares actuales de seguridad como es OIDC.

Posteriormente nos centraremos en la preparación y configuración de una monitorización general del clúster tanto a nivel de recopilación y presentación de métricas de uso y estado como a nivel de alertas en caso de que se produzcan situaciones anómalas. Y esto se hace con herramientas actuales muy populares como Prometheus o Grafana. Esto permitirá conocer el estado general del clúster y poder configurar notificaciones cuando correspondan, haciendo que la operación del clúster sea más sencilla.

De esta forma acabaremos con un clúster que podemos considerar base para que a partir de él se pueda empezar a usar para los diferentes usos que el laboratorio permita sabiendo que los requisitos de seguridad y monitorización están ya cubiertos.

Palabras clave: Kubernetes, Raspberry Pi, OIDC, autenticación y autorización, monitorización, Prometheus, Grafana, alertas

Abstract

The irruption of the infrastructures and the platforms as a service has marked a milestone in many fields and open a world of new possibilities that till then were just simply not possible. If additionally they are combined with microcomputers with reduced size and weight, although good outputs, we'll get as result the possibility of being able to create computer clusters with a size smaller than a conventional computer tower but with great potential. And, if this kind of cluster is meant to be used as laboratories, we'd get a very powerful tool for the students' tasks and education. Thinking about this, the eNMoLabs Project is developing and managing a set of works with the aim of initiating a Lab of Things based on services, a laboratory with several different integrated functionalities that can be accessed remotely to work on it.

Inside this Project is where the current work is defined. We'll create a Kubernetes cluster over a set of low-cost devices called Raspberry Pi. Over this cluster, we'll set an application that allows the user authentication and authorization with the university credentials, and with a permissions management that ensures that the authenticated users are assigned to the groups where they belong to, which at the same time corresponds to roles that are set to control the permissions of the things that can be done inside the cluster. This security layer will be done following one of the current security standards, as OIDC is.

Later we'll focus on the preparation and configuration of general monitoring for the cluster, both to gather and present usage and status metrics, and to prepare alerts that will raise in unexpected events. And this is done using tools that are very common currently and popular like Prometheus and Grafana. This will allow us to know the general status of the cluster and be able to set notifications when corresponding, doing the cluster operation simpler.

In this way, we'll end up with a cluster that we can consider as a base that can be used for the different uses that the laboratory permit, knowing that the requirements related to security and monitoring are already covered.

Keywords: Kubernetes, Raspberry Pi, OIDC, authentication and authorization, monitoring, Prometheus, Grafana, alerts

Tabla de contenidos

Agradecimientos v
Resumen vii
Abstract ix
Tabla de contenidos xi
Lista de abreviaturas y siglas xv
Lista de figuras xvii
Lista de tablas xxiii
1. Introducción 1
1.1. Objetivo 1
1.2. Alcance 2
1.3. Presupuesto 3
1.3.1. Recursos hardware 3
1.3.2. Recursos software 4
1.3.3. Recursos humanos 4
1.3.4. Resumen 5
1.4. Planificación del trabajo 5
1.5. Estructura y contenido del documento 6
2. Estado del arte 9
3. Tecnologías usadas 13
3.1. Listado de tecnologías consideradas 13
3.2. Relación de las tecnologías en el trabajo actual 18
3.3. Resumen del capítulo 18
4. Implementación llevada a cabo 21
4.1. Arquitectura a nivel de hardware 21
4.2. Arquitectura de autenticación y autorización en el clúster 30
4.2.1. Aplicación OIDC para la autenticación y gestión de autorización 33
4.2.1.1. Despliegue de la aplicación 33
4.2.1.2. Autenticación por parte del usuario 39
4.2.1.3. Well-known 42
4.2.1.4. Parámetros de configuración de la aplicación 43
4.2.1.5. Definición y configuración de roles en la aplicación OIDC 44
4.2.1.6. Ejemplo y explicación de un resultado correcto de la autenticación 46
4.2.1.7. Funcionalidad de gestión de usuarios locales o de aplicación 48
4.2.1.8. Auto escalado horizontal de la aplicación 49

4.2.1.9. Implementación de la aplicación 51
4.2.1.10. Otras notas relacionadas con la aplicación 52
4.2.2. Cliente OIDC para usuarios 53
4.2.2.1. Descripción funcional 53
4.2.2.2. Detalle de la implementación 55
4.3. Monitorización 57
4.3.1. Prometheus 57
4.3.2. Alert manager 70
4.3.3. Grafana 78
4.4. Dashboard de Kubernetes 89
4.5. Aspecto final del clúster a nivel de kubernetes 94
4.5.1. Namespaces 95
4.5.2. Despliegues y réplicas 96
4.5.3. Servicios expuestos fuera del clúster 98
4.5.4. Uso de memoria y CPU 99
4.6. Resumen del capítulo 100
5. Pruebas y resultados obtenidos 101
5.1. Preparación previa a las pruebas: detalle de operaciones que vamos a realizar durante las pruebas 101
5.1.1. Creación de un *namespace* 101
5.1.2. Creación de rol por defecto con permisos de modificación para trabajar con cada *namespace* 102
5.1.3. Creación de rol con permisos de solo lectura para un *namespace* 103
5.1.4. Creación de una aplicación de pruebas 104
5.1.5. Aplicación de automatización de creación de objetos en kubernetes 106
5.2. Acceso administrador al clúster y crear un *namespace* nuevo y objetos en él 108
5.3. Acceso con usuario limitado a un *namespace* donde está autorizado y crear otra aplicación ahí 112
5.4. Acceso con usuario limitado y forzar error al intentar crear un *namespace* 115
5.5. Acceso con usuario limitado a un *namespace* al que no está autorizado y forzar un error .. 116
5.6. Acceso con usuario no administrador y forzar un error al intentar listar recursos de *namespaces* del sistema o de seguridad 120
5.7. Acceso con un usuario con acceso al grupo general asociado a su dominio de email, que se corresponde con un grupo de solo lectura sobre un *namespace* 123
5.8. Monitorización en las pruebas relacionadas con los accesos al clúster 124
5.9. Resumen del capítulo 126
6. Conclusiones 129

6.1. Logros alcanzados 129
6.2. Trabajos futuros 131
6.2.1. Integración del trabajo realizado en el proyecto eNMoLabs 131
6.2.2. Mejoras en la aplicación OIDC 131
6.2.3. Monitorización 133
6.2.4. Logging 134
6.2.5. Ideas generales 135
Referencias y bibliografía 137
Anexo 141
A. Instalación del clúster 141
B. Exposición del clúster hacia fuera del mismo 152
C. Configuración del clúster en sí para su acceso remoto 154
D. Problemas y utilidades 159
E. Listado de ficheros YAML usados en la creación del clúster 166
F. Generación de imagen Docker para Raspberry Pi 169
G. Cómo se valida un token JWT 174

Lista de abreviaturas y siglas

eNMoLabs	Efficient Network Management of Laboratories
IoT	Internet of Things (Internet de las cosas)
IP	Internet Protocol (protocolo de internet)
K8S	Kubernetes
NTP	Network Time Protocol
OIDC	OpenID Connect
RPI	Raspberry Pi
SSH	Secure Shell
SSO	Single Sign-On
UNED	Universidad Nacional de Educación a Distancia
WoT	Web of Things
YAML	YAML Ain't Markup Language (*YAML no es un lenguaje de marcado*)

Lista de figuras

Figura 1-1: planificación del trabajo realizado 6
Figura 2-1: foto del clúster físico de Raspberry Pi para LoT@UNED. Imagen extraída de (eNMoLabs P. , Entregable 2.2, 2020) 10
Figura 2-2: solución técnica para LoT@UNED. Imagen extraída de (Pastor, 2020) 11
Figura 4-1: foto de las Raspberry Pi usadas para el presente trabajo 21
Figura 4-2: diagrama hardware de las Raspberry y la configuración de red 22
Figura 4-3: esquema de organización de los nodos del clúster y cómo accede un usuario del clúster al mismo 26
Figura 4-4: esquema de autenticación y autorización en kubernetes por cada petición. Imagen extraída de (Kubernetes, 2020) 32
Figura 4-5: configuración de la asociación del rol admin al grupo admin proveniente de la autorización. Fichero 40-cluster_admin.yaml 33
Figura 4-6: configuración del despliegue de la aplicación OIDC. Fichero 30-uned-oidc.yaml 35
Figura 4-7: configuración del servicio de la aplicación OIDC. Fichero 30-uned-oidc.yaml 37
Figura 4-8: configuración de la exposición del servicio de la aplicación OIDC. Fichero 30-uned-oidc.yaml 38
Figura 4-9: pantalla inicial de autenticación 40
Figura 4-10: diagrama de secuencia de los eventos principales para un proceso correcto de autenticación vía OIDC 41
Figura 4-11: contenido que devuelve el servicio well-known 42
Figura 4-12: contenido de la URL que devuelve las claves públicas para validaciones en cliente 43
Figura 4-13: configuración para el arranque correcto de la aplicación OIDC. Fichero 21-configmap-seguridad.yaml 43
Figura 4-14: configuración de grupos para dominios y usuarios. Fichero 22-configmap-map-grupos.yaml 45
Figura 4-15: ejemplo de id_token generado tras una autenticación correcta con la aplicación OIDC. 46
Figura 4-16: Información contenida en el id_token relativa al usuario que se ha autenticado 47
Figura 4-17: configuración de usuarios locales. Fichero 23-configmap-map-usuarios-locales.yaml.... 48
Figura 4-18: configuración de auto escalado para la aplicación OIDC. Fichero 31-uned-oidc-hpa.yaml 50
Figura 4-19: confirmación del resultado correcto del proceso de autenticación 54
Figura 4-20: información sobre el resultado incorrecto del proceso de autenticación 54
Figura 4-21: parámetros a configurar en la aplicación cliente OIDC antes de su distribución 55
Figura 4-22: diagrama de secuencia de la aplicación cliente 56
Figura 4-23: estructura sobre cómo se asientan las piezas de monitorización dentro del clúster 57
Figura 4-24: configuración del objeto ClusterRole dentro del clúster para otorgar permisos al usuario asociado a Prometheus. Fichero 10-prom-clusterrole.yaml 59
Figura 4-25: esquema de la configuración para Prometheus. Fichero 11-prom-config.yaml 60
Figura 4-26: despliegue y exposición de Prometheus. Fichero 12-prometheus.yaml. 62
Figura 4-27: exposición al exterior de Prometheus. Fichero 13-prom-exponer.yaml 64
Figura 4-28: pantalla inicial cuando se accede a Prometheus, con la interfaz gráfica que permite realizar consultas sobre las métricas disponibles 65
Figura 4-29: consulta en Prometheus del porcentaje de memoria en uso, con gráfica de resultado .. 66
Figura 4-30: consulta en Prometheus del porcentaje de memoria en uso, con tabla de resultado 66
Figura 4-31: consulta en Prometheus sobre el uso de memoria en cada nodo, con gráfico de resultado 67

Figura 4-32: consulta en Prometheus sobre el uso de memoria en cada nodo, con tabla de resultado 67
Figura 4-33: pantalla de alertas en la aplicación Prometheus 68
Figura 4-34: detalle de una alerta, vista desde la sección de alertas de la aplicación Prometheus 69
Figura 4-35: primeros elementos del listado de fuentes de métricas configuradas para Prometheus, vista desde la aplicación Prometheus 70
Figura 4-36: flujo básico de las alertas detectadas por Prometheus 71
Figura 4-37: ejemplo de configuración de una alerta en Prometheus 71
Figura 4-38: configuración de la aplicación alert manager. Fichero 30-alertmgr-configmap.yaml 73
Figura 4-39: notificación recibida al canal de MS Teams tras disparo de una alerta 75
Figura 4-40: notificación recibida al buzón de correo electrónico tras disparo de una alerta 76
Figura 4-41: plantillas por defecto usadas para las notificaciones en alert manager. Fichero 31-alertmgr-templates.yaml 76
Figura 4-42: despliegue de la aplicación alert manager. Fichero 32-alertmgr.yaml 77
Figura 4-43: exposición de la aplicación alert manager. Fichero 32-alertmgr.yaml 78
Figura 4-44: configuración para Grafana. Fichero 40-grafana-datasource-configmap.yaml 79
Figura 4-45: configuración sobre los dashboards de Grafana. Fichero 41-grafana-dashboard-configmap.yaml 80
Figura 4-46: dashboard creado para Grafana. Fichero 42-grafana-dashboard.yaml 80
Figura 4-47: despliegue de Grafana. Fichero 43-grafana.yaml 82
Figura 4-48: exposición interna de Grafana. Fichero 43-grafana.yaml 83
Figura 4-49: exposición hacia fuera del clúster de Grafana. Fichero 44-grafana-exponer.yaml 84
Figura 4-50: pantalla de autenticación de Grafana 85
Figura 4-51: pantalla de inicio de Grafana, tras una correcta autenticación. En la parte de abajo a la izquierda se puede ver el dashboard ya creado 86
Figura 4-52: sección del dashboard de Grafana que muestra información técnica (versión de número, IP, etc.) de cada uno de los nodos, así como el tiempo que llevan en funcionamiento 86
Figura 4-53: sección del dashboard de Grafana que muestra el uso global de memoria y CPU en el clúster, así como el porcentaje de réplicas no disponible 87
Figura 4-54: sección del dashboard de Grafana que muestra la memoria (MiB) y CPU (%) usada en cada uno de los nodos 87
Figura 4-55: sección del dashboard de Grafana que muestra información sobre las alertas activas y sobre el porcentaje de almacenamiento físico usado en el clúster 87
Figura 4-56: sección del dashboard de Grafana que muestra el uso de memoria y CPU a nivel de pod 87
Figura 4-57: sección del dashboard de Grafana que muestra información sobre el tráfico de red a nivel de pod 88
Figura 4-58: detalle de configuración en Grafana del panel relativo al uso de memoria en cada nodo 89
Figura 4-59: argumentos de arranque del contenedor de kubernetes dashboard. Parte del fichero dashboard.yaml 90
Figura 4-60: creación de usuario para kubernetes dashboard y su asociación al rol correspondiente. Fichero dashboard-user.yaml 92
Figura 4-61: exposición hacia fuera del clúster de kubernetes dashboard. Fichero exponer-dashboard.yaml 92
Figura 4-62: pantalla de autenticación para acceder al dashboard de Kubernetes 93
Figura 4-63: parte de la pantalla resumen del namespace seguridad en el dashboard de kubernetes 94
Figura 4-64: listado de namespaces tras la configuración del clúster 95

Figura 4-65: listado de despliegues tras la configuración del clúster 97
Figura 4-66: listado de servicios expuestos hacia fuera del clúster 98
Figura 4-67: recursos usados por el clúster una.. 99
Figura 5-1: fichero con la plantilla de configuración de un namespace nuevo................................. 102
Figura 5-2: fichero con la plantilla de configuración de un rol para un namespace nuevo 103
Figura 5-3: fichero con la plantilla de configuración de una asignación de un rol a un grupo 103
Figura 5-4: fichero con la plantilla de configuración del rol de solo lectura 104
Figura 5-5: fichero con la plantilla de configuración de una asignación del rol de solo lectura al grupo alumno... 104
Figura 5-6: fichero con la plantilla de creación de una aplicación y su exposición al clúster 105
Figura 5-7: fichero con la plantilla de exposición de la aplicación creada hacia fuera del clúster 106
Figura 5-8: información sobre la aplicación creada para la automación de creación de objetos en kubernetes 107
Figura 5-9: creación de un namespace con la aplicación creada para la automación de creación de objetos en kubernetes 108
Figura 5-10: creación de una aplicación con la aplicación creada para la automación de creación de objetos en kubernetes 108
Figura 5-11: comprobación del usuario actual atellez9 y sus permisos sobre el namespace nstest1 110
Figura 5-12: resultado de la creación del nuevo namespace nstest1 y un rol con permisos sobre los objectos de este 110
Figura 5-13: resultado del despliegue y exposición de la aplicación de prueba en el namespace nstest1 110
Figura 5-14: comprobación de que todos los objetos de la aplicación nred1 se han instalado dentro del namespace nstest1... 111
Figura 5-15: imagen de la aplicación instalada en el namespace nstest1 accediendo desde fuera del clúster 111
Figura 5-16: comprobación del usuario actual local_u1 y sus permisos sobre el namespace nstest1113
Figura 5-17: resultado del despliegue y exposición de la aplicación de prueba nred2 en el namespace nstest1 113
Figura 5-18: comprobación de que todos los objetos relacionados con la aplicación nred2 se han instalado dentro del namespace nstest1 114
Figura 5-19: imagen de la aplicación nred2 instalada en el namespace nstest1 accediendo desde fuera del clúster 114
Figura 5-20: error producido al intentar crear un namespace cuando no se tiene permisos para ello 116
Figura 5-21: comprobación del usuario actual local_u2 y sus permisos sobre el namespace nstest1117
Figura 5-22: error obtenido por el usuario local_u2 al intentar listar los deployments del namespace nstest1, al no tener permisos.. 118
Figura 5-23: error obtenido por el usuario local_u2 al intentar listar los pods del namespace nstest1, al no tener permisos... 118
Figura 5-24: error obtenido por el usuario local_u2 al intentar listar todos los objetos del namespace nstest1, al no tener permisos.. 119
Figura 5-25: error obtenido por el usuario local_u2 al intentar crear una aplicación en el namespace nstest1, al no tener permisos.. 120
Figura 5-26: comprobación del usuario actual local_u1 y sus permisos sobre los namespaces kube-system y seguridad 121
Figura 5-27: errores obtenidos por el usuario local_u1 al intentar recuperar objetos de los namespaces kube-system y seguridad 122

Figura 5-28: comprobación del usuario actual local_u3 y sus permisos sobre el namespace nstest2124
Figura 5-29: listado de objetos del namespace nstest2, recuperados correctamente por el usuario local_u3 124
Figura 5-30: error recibido al intentar crear un objeto en el namespace nstest2, por parte del usuario local_u3 124
Figura 5-31: paneles del dashboard de Grafana que muestran el uso de CPU y memoria por parte de los pods de las aplicaciones creadas para las pruebas realizadas........ 125
Figura 5-32: paneles del dashboard de Grafana que muestran el tráfico de red por parte de los pods y sus contenedores de las aplicaciones creadas para las pruebas realizadas 126
Figura A-1: Imagen inicial de la herramienta Raspberry Pi Imager para Windows........ 141
Figura A-2: algunos de los sistemas operativos que podemos instalar en la tarjeta SD con la herramienta Raspberry Pi Imager 142
Figura A-3: selección de tarjeta SD que queremos formatear y donde instalaremos el sistema operativo elegido con la herramienta Raspberry Pi Imager 142
Figura A-4: proceso de formateo e instalación en curso con la herramienta Raspberry Pi Imager ... 143
Figura A-5: confirmación de la herramienta Raspberry Pi Imager de escritura satisfactoria del sistema operativo en la tarjeta de memoria 143
Figura A-6: configuración de conexión WiFi a guardar en la tarjeta de memoria 144
Figura A-7: conexión SSH a la Raspberry........ 145
Figura A-8: cambio de contraseña por defecto del usuario por defecto de la Raspberry 145
Figura A-9: ejecución de la utilidad raspi-config que nos permite realizar varias tareas de configuración sobre la Raspberry........ 145
Figura A-10: menú inicial de la utilidad raspi-config 145
Figura A-11: selección del menú de opciones de red en la utilidad raspi-config........ 146
Figura A-12: selección del submenú de host en la utilidad raspi-config........ 146
Figura A-13: configuración del nombre de host en raspi-config........ 146
Figura A-14: comandos para actualizar los paquetes instalados en la Raspberry 147
Figura A-15: comando para la instalación de k3s en la Raspberry master 147
Figura A-16: ejecución del comando para la obtención del nodo recién instalado en k3s, en la Raspberry master 148
Figura A-17: obtención del token desde la Raspberry master para que otros nodos puedan unirse al clúster........ 148
Figura A-18: comando para la instalación de k3s en las Raspberry worker........ 148
Figura A-19: listado de nodos unidos al clúster. Comando ejecutado desde la Raspberry master, donde se ve que inicialmente los nodos worker no tienen un rol asignado........ 149
Figura A-20: asignación del rol worker a cada una de las Raspberry worker 149
Figura A-21: listado de nodos unidos al clúster, ahora ya con todos los nodos con un rol asignado. 149
Figura A-22: parámetros usados para la ejecución del clúster 150
Figura C-1: configuración del acceso al clúster en la Raspberry máster........ 156
Figura C-2: certificado para conectarse al clúster con una conexión segura........ 156
Figura C-3: comando para crear un clúster dentro de la configuración general de kubernetes en el dispositivo local........ 157
Figura C-4: comando para crear el contexto que relaciona el clúster con el usuario de ejecución, dentro de la configuración general de kubernetes en el dispositivo local 157
Figura C-5: comando para seleccionar el contexto con el queramos trabajar, dentro de la configuración general de kubernetes en el dispositivo local........ 157
Figura C-6: comando para crear/actualizar las credenciales de un usuario de kubernetes, dentro de la configuración general de kubernetes en el dispositivo local........ 158

Figura D-1: ejecución del comando para ver el estado de k3s, desde la Raspberry máster.............. 159
Figura D-2: ejecución del comando para ver los logs de k3s, desde la Raspberry máster................. 160
Figura D-3: comando para reiniciar el clúster k3s, ejecutado desde la Raspberry máster................ 160
Figura D-4: error de conexión con un dominio creado para una red local 161
Figura D-5: definición de hosts que serán resueltos sin usar servidores DNS, en la Raspberry máster 161
Figura D-6: error de validación de certificado al intentar conectar a la aplicación OIDC, al ser un certificado firmado por una entidad no reconocida 162
Figura D-7: error de validación de certificado al intentar conectar a la aplicación OIDC, al ser un certificado firmado por una entidad no reconocida 162
Figura D-8: nodo NotReady en el clúster, al consultar el listado de nodos del clúster...................... 163
Figura D-9: pasos a seguir en la Raspberry máster para intentar solucionar el problema de conectividad 164
Figura D-10: pasos a seguir en cada Raspberry worker para intentar solucionar el problema de conectividad 164
Figura D-11: comando para mostrar la configuración de kubernetes 164
Figura D-12: comando para mostrar el listado de contextos disponibles, e indicando cuál es el actual, desde la configuración de kubernetes 165
Figura D-13: comando para mostrar el id_token para un usuario concreto, recuperándolo desde la configuración de kubernetes.. 165
Figura F-1: contenido del fichero Dockerfile para la aplicación OIDC... 169
Figura F-2: comando usado para la generación de la imagen Docker para otras arquitecturas 170
Figura F-3: imagen extraída del proceso de generación de la imagen Docker multiplataforma donde se puede ver que se están instalando las dependencias de la aplicación NodeJS para las tres arquitecturas destino 171
Figura F-4: imagen extraída del proceso de generación de la imagen Docker multiplataforma donde se puede ver que se ha compilado para las tres arquitecturas y la imagen resultante se está enviando al hub de Docker.. 172
Figura F-5: imagen del proyecto para la aplicación OIDC dentro del hub de Docker 172
Figura F-6: imagen del hub de Docker donde vemos la última imagen de la aplicación OIDC que hemos publicado, y como ésta está disponible para las tres arquitecturas para la que la habíamos compilado.. 173
Figura F-7: comando para la creación de un secreto en el clúster que contiene las credenciales para poder acceder a un repositorio privado del hub de Docker 173
Figura G-1: información sobre el tipo de token y cómo se ha generado su firma............................ 174
Figura G-2: parte de la información pública asociada a la clave usada para generar el token........... 175

Lista de tablas

Tabla 1-1-Presupuesto relacionado con los recursos hardware 3
Tabla 1-2-Presupuesto relacionado con los recursos software 4
Tabla 1-3-Presupuesto relacionado con los recursos humanos........ 5
Tabla 1-4-Presupuesto total destinado al trabajo........ 5
Tabla 4-1-Extracto de la información obtenida del comando kubectl get nodes -o wide donde vemos cada nodo con su rol, versión de k3s, su IP interna asignada, la imagen del SO y la versión de kernel22
Tabla 4-2: Características técnicas del nodo rpi-master 23
Tabla 4-3: Características técnicas de los nodos rpi-worker1 y rpi-worker2 24
Tabla 4-4: Características técnicas del nodo rpi-worker3 24
Tabla 4-5: arquitectura, número de núcleos y RAM de cada nodo del clúster........ 25

1. Introducción

1.1. Objetivo

Este Trabajo Fin de Máster (TFM) se enmarca en el proyecto de investigación eNMoLabs (*efficient Network Management of Laboratories,* (eNMoLabs, 2020)) perteneciente a la UNED.

Los objetivos generales del Proyecto eNMoLabs giran en torno al desarrollo de laboratorios remotos y su gestión, así como la explotación de la información generada, y siempre ofreciendo los servicios con calidad en términos de tolerancia a fallos y rendimiento. Estos laboratorios pueden ser de diversa naturaleza y propósito, pero con el requisito común de que sea accesibles como un servicio por estudiantes y otros usuarios fuera del laboratorio que estén utilizando.

Para este trabajo nos centramos en la aplicación de mejoras a nivel técnico para un clúster formado por varias unidades de pequeños ordenadores de placa simple de la marca Raspberry Pi, con el objetivo de preparar un clúster que sea funcional no solo a nivel técnico, sino que provea de herramientas de seguridad y monitorización que permitan un uso más seguro y con mejor previsión y análisis de errores.

Partiendo del hardware usado, crearemos un clúster basado en una versión ligera de Kubernetes, llamada k3s (k3s, 2020), que a su vez se basa en gestión de contenedores de la tecnología Docker (Docker, 2020).. Una vez creado el clúster, instalaremos una aplicación OIDC (OIDC, 2020) basada en la autenticación usada en la UNED y configuraremos nuestro clúster para trabajar con dicha aplicación. Posteriormente implementaremos roles y grupos que permitan una gestión sencilla de permisos, y finalmente preparamos un conjunto de aplicaciones de monitoreo con algunas alertas que nos permitan llevar a cabo una gestión del clúster más sencilla y una reconfiguración automática, aprovechando las bondades de las tecnologías usadas.

Así pues, el objetivo principal del presente trabajo es la preparación de un clúster base, independiente y completamente operativo que posteriormente se pueda dedicar a una labor en concreto, pero que disponga ya dentro de esta base herramientas de seguridad y

monitorización que sean estándar, sea cual sea el uso que se le vaya a dar. Partiendo de este objetivo principal, los objetivos específicos marcados son:

1. Crear una aplicación que gestione la autenticación en el clúster con credenciales de la UNED.
2. Disponer de una variedad de roles, que finalmente impliquen diferentes permisos, que sean fácilmente asignables a usuarios.
3. Proveer herramientas de monitorización que muestren el estado del clúster, así como permitan hacer consultas sobre las métricas recopiladas.
4. Configurar alertas que se disparen automáticamente ante situaciones imprevistas o no deseables.

1.2. Alcance

El alcance del trabajo queda sujeto a una serie de mejoras sobre el clúster en sí. Y siempre teniendo en cuenta que estas mejoras puedan ser comunes a cualquier tipo de clúster, esto es, mejoras aplicables a cualquier clúster independientemente del uso que se vaya a dar al clúster (prácticas en laboratorios, servidores de algún tipo, etc.). Los desarrollos se harán sobre la autenticación y autorización de usuarios del clúster en sí, y sobre la monitorización que se pueda realizar sobre el mismo.

Por otro lado, nos centramos en un clúster basado en imágenes Docker gestionadas por medio de una versión de Kubernetes, que es "*una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores*" (Kubernetes, 2020).

1.3. Presupuesto

Para calcular el presupuesto necesario para llevar a cabo este proyecto debemos considerar los diferentes tipos de recursos implicados.

1.3.1. Recursos hardware

Aquí vamos a considerar todo el hardware usado, tanto aquel del que ya disponíamos con anterioridad, como todo lo que se ha adquirido para la realización de este trabajo.

Recurso	Coste (€)
Ordenador de trabajo	800
3x Raspberry Pi 3 Model B (con cargador)	150
1x Raspberry Pi 4 Model B (con cargador)	65
4x tarjeta de memoria SD 64GB	50
Ratón y teclado inalámbrico	25
Cable HDMI	10
Total	**1.100**

Tabla 1-1-Presupuesto relacionado con los recursos hardware

Como excepción, no hemos incluido el router, ni la televisión que hemos usado a modo de pantalla, ni otros elementos que hemos aprovechado para este trabajo pero que no han sido de uso exclusivo para el mismo. No obstante, hay que tener en cuenta que probablemente haya que disponer de alternativas a los mismos para poder trabajar con futuros clústeres.

El ratón, teclado y cable HDMI, así como la televisión-pantalla, no son estrictamente necesarios, pero pueden ser de utilidad en algún momento, por lo que el aprovisionarse de ellos es a discreción del equipo de trabajo.

1.3.2. Recursos software

Aunque la mayoría de software usado no supone un coste, aquí lo consideramos para tenerlo en cuenta y estar pendientes de las licencias.

Recurso	Coste (€)
Licencia Windows 10 Pro	259
Licencia Office 365	69
RPI imager	0
Kubernetes	0
Prometheus	0
Alert manager	0
Grafana	0
Total	**328**

Tabla 1-2-Presupuesto relacionado con los recursos software

Para la realización de este trabajo hemos usado tanto Microsoft Windows como Microsoft Office, pero no son imprescindibles para la realización de este tipo de trabajo, por lo que podrían considerarse otras opciones con el consiguiente ahorro económico.

1.3.3. Recursos humanos

Hemos considerado un único desarrollador-administrador del clúster, a un coste de 25€/hora. Por tanto:

Recurso	Horas	Coste (€)
Recopilación y estudio de información inicial	20	500
Recopilación y estudio de información adicional	20	500
Instalación inicial del clúster	15	375
Análisis y desarrollo aplicación OIDC	65	1.625
Análisis y desarrollo cliente OIDC	15	375
Preparación de la monitorización	65	1.625

Pruebas	25	625
Otras tareas derivadas del trabajo	15	375
Documentación	60	1.500
Total	**300**	**7.500**

Tabla 1-3-Presupuesto relacionado con los recursos humanos

1.3.4. Resumen

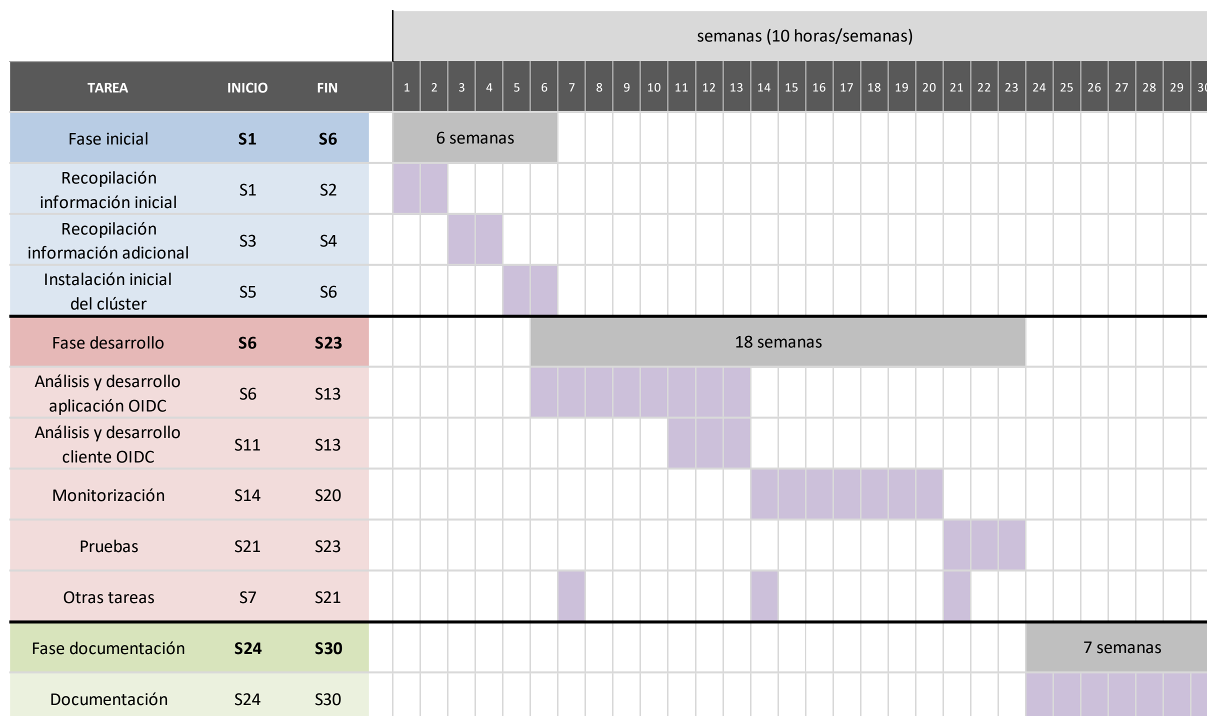
Así pues, agrupando todos los costes, el presupuesto total es:

Recurso	Coste (€)
Hardware	1.100
Software	328
Humano	13.000
Total	**14.428**

Tabla 1-4-Presupuesto total destinado al trabajo

1.4. Planificación del trabajo

Consideramos una dedicación al trabajo de unas 10 horas semanales durante unos 7 meses (30 semanas), y teniendo en cuenta que el trabajo no es estrictamente un trabajo en secuencia, por lo que algunas tareas se superponen a otras. Así pues, la planificación de las tareas del trabajo queda tal como se ve en la Figura 1-1.



TAREA	INICIO	FIN
Fase inicial	**S1**	**S6**
Recopilación información inicial	S1	S2
Recopilación información adicional	S3	S4
Instalación inicial del clúster	S5	S6
Fase desarrollo	**S6**	**S23**
Análisis y desarrollo aplicación OIDC	S6	S13
Análisis y desarrollo cliente OIDC	S11	S13
Monitorización	S14	S20
Pruebas	S21	S23
Otras tareas	S7	S21
Fase documentación	**S24**	**S30**
Documentación	S24	S30

Figura 1-1: planificación del trabajo realizado

1.5. Estructura y contenido del documento

Este documento se divide en los siguientes capítulos:

- Introducción: toma de contacto sobre el trabajo, detallando el objetivo y alcance de este.
- Estado del arte: presentación del clúster actual preparado por el laboratorio con anterioridad, así como de otras tecnologías usadas.
- Tecnologías usadas: listado de las tecnologías y herramientas principales que se han usado en el desarrollo del trabajo.
- Implementación llevada a cabo: en este capítulo entraremos en detalle de lo que se ha hecho, explicando funcionalmente como técnicamente los pasos seguidos y con qué finalidad se han llevado a cabo.
- Pruebas y resultados obtenidos: en base a la implementación anterior, aquí nos centraremos en realizar un conjunto de pruebas para validar todo lo realizado anteriormente, así como presentar y explicar los resultados obtenidos.

- Conclusiones, logros obtenidos y trabajos futuros: por último, resumiremos qué hitos hemos alcanzado con la realización del trabajo y mencionaremos posibles trabajos futuros que se pueden realizar para extender lo aquí hecho.

2. Estado del arte

El Proyecto eNMoLabs lleva varios trabajos realizados en relación con la preparación y explotación de laboratorios remotos y virtuales, trabajando en la línea de poder relacionar cada uno de los dispositivos IoT que componen los laboratorios para formar finalmente una WoT (W3C, 2020), o red que los englobe a todos los laboratorios con todos sus dispositivos. Esta WoT se bautizó como **LoT@UNED** (*Labs of Things at UNED*) y tiene por objetivo *implementar un entorno dinámico que permita el crecimiento horizontal (dispositivos/sensores) y vertical (servicios de aprendizaje ofrecidos en diferentes ámbitos educativos)* (eNMoLabs P. , Entregable 2.2, 2020).

Aunque este proyecto no es el único proyecto relacionado con la preparación de laboratorios remotos, sí es el que más características intenta cubrir, ya que la plataforma LoT@UNED cuenta con varias características y requisitos esenciales (Pastor, 2020) que otros proyectos sólo cubren den cierta medida: acceso y desarrollo con dispositivos IoT, desarrollo de soluciones en la niebla, análisis de los datos recuperados por los sensores e interacción con actuadores disponibles, configuración y gestión de protocolos de comunicación específicos para IoT y desarrollo de técnicas de seguridad en entornos IoT. Estas características han guiado los diferentes trabajos ya realizados dentro del proyecto, así como este trabajo fin de máster.

Uno de los primeros laboratorios usados como parte de este proyecto fue el reaprovechamiento de un par de prototipos de laboratorios remotos basados en energía solar y eólica (Al-Zoubi, 2014). Estos laboratorios exponen una interfaz HTTP que puede ser consumida por terceros. A partir de estos laboratorios se hizo un mecanismo de almacenamiento de la actividad de los estudiantes basado en registros mediante el estándar Experience API (xAPI) (eNMoLabs P. , Entregable 2.1 (E2.1), 2019).

Otro de los trabajos previos realizados fue un pequeño laboratorio de IoT con dispositivos Raspberry Pi (eNMoLabs P. , Entregable 2.1 (E2.1), 2019) que permitía acceder al mismo de forma remota y obtener datos generados por medio de los sensores de dicho laboratorio, como la temperatura y humedad de la sala donde se encontraba este laboratorio. También se permitía mandar órdenes a los actuadores integrados. Este laboratorio también contaba con otras capacidades como un GPS, cámara y micrófono. Este laboratorio fue usado

por los alumnos de las prácticas de la asignatura Cloud Computing del máster al que pertenece este trabajo (Pastor, 2020).

El último de los trabajos previos realizados por eNMoLabs que vamos a mencionar tiene relación con un conjunto de Raspberry Pi sobre el que se había configurado un clúster de kubernetes que había sido destinado al uso por parte de los alumnos de un par de asignaturas para la realización de alguna de sus prácticas, tales como la realización de actividades prácticas de seguridad en la nube, seguridad en los sistemas de información o monitorización de redes mediante el uso de redes virtualizadas (eNMoLabs P. , Entregable 2.1 (E2.1), 2019).

Este clúster kubernetes sido mejorado para mejorar sus capacidades de reconfiguración en caso de que ocurra alguna anomalía, mejorando su tolerancia a fallos. También se han efectuado mejoras de cara a reducir latencias y para realizar un balanceo de carga de red más eficiente (eNMoLabs P. , Entregable 3.1, 2020). Este clúster podemos verlo como el componente que ocupa la parte izquierda de la Figura 2-2. Sin embargo, el clúster, al componerse por dispositivos de tan reducidas dimensiones como son las Raspberry Pi, ocupa poco espacio físico, tal como se puede ver en esta figura:

Figura 2-1: foto del clúster físico de Raspberry Pi para LoT@UNED. Imagen extraída de (eNMoLabs P. , Entregable 2.2, 2020)

La solución técnica de la plataforma LoT@UNED es la siguiente:

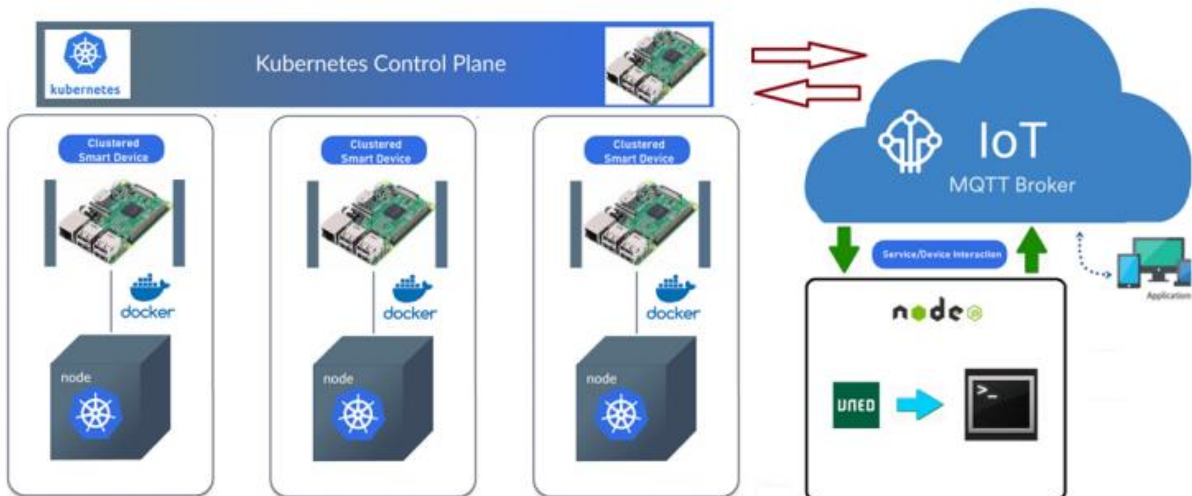


Figura 2-2: solución técnica para LoT@UNED. Imagen extraída de (Pastor, 2020)

La plataforma dispone de un portal de acceso para estudiantes y profesores, y de una Shell interactiva donde se van dando las órdenes que permita la práctica correspondiente (eNMoLabs P. , Entregable 2.2, 2020). Como se puede ver en la Figura 2-2, el estudiante/profesor accede al portal de acceso y tras la autenticación, llega a la Shell interactiva (esto se corresponde con parte inferior derecha de la figura) y desde ahí va dando órdenes que se van gestionando desde el bróker MQTT. Este bróker es el que iré transmitiendo los mensajes al plano de control de kubernetes, que es desde donde se gestiona el clúster de kubernetes mencionado anteriormente.

Si bien este clúster este clúster de kubernetes ya desarrollado sobre un conjunto de Raspberry Pi era perfectamente funcional, carecía de ciertas características relevantes para un uso más extendido, así como control de acceso al clúster, herramientas de monitorización, etc. El trabajo aquí desarrollado será integrado en el clúster kubernetes, ampliando las funcionalidades cubiertas y mejorando su gestión.

Por otro lado, y de forma casual debido a la situación extraordinaria vivida en los últimos meses, el clúster ha estado inactivo, motivo por el cual el desarrollo enmarcado en este trabajo no ha podido tomar ventaja del camino ya andado y realizar validaciones y pruebas sobre el clúster mencionado, que en cualquier caso será el clúster final una vez se pueda restaurar el acceso. Por ello, se ha tenido que crear un clúster nuevo emulando un

pequeño LoT@UNED particular para este trabajo. No obstante, los desarrollos hechos serán casi directamente migrables a la plataforma LoT@UNED.

3. Tecnologías usadas

En este capítulo veremos las tecnologías que se han usado o, al menos, considerado en algún momento del proyecto, así como las que se han decidido usar finalmente. Incluiremos herramientas, lenguajes, estándares, etc., con la intención de dar una mejor visión sobre el trabajo realizado.

3.1. Listado de tecnologías consideradas

Para hablar de las principales tecnologías usadas o referenciadas, vamos a hacer un repaso de las tecnologías, pero poniéndolas en el contexto que nos ocupa, para poder entender mejor su uso y necesidad.

Como hardware base hemos considerado únicamente dispositivos **Raspberry Pi**, al tratarse de dispositivos que pueden funcionar muy bien como servidores de trabajo y con un baje coste, y también debido a que estos dispositivos se han usado con anterioridad en el Proyecto eNMoLabs, previendo que se sigan utilizando en el futuro. En concreto, para este trabajo hemos usado tanto el modelo *Raspberry Pi 3 Model B* como *Raspberry Pi 4 Model B*. En el apartado 4.1 veremos las capacidades que ofrecen dichos modelos.

Para emular un pequeño LoT@UNED, necesitamos preparar un clúster que funcione como una sola unidad, y para ello nos decidimos por **Kubernetes**, que se encarga precisamente de esto, abstrayendo al administrador del clúster de su complejidad y ofreciendo la impresión de que se trata de un único dispositivo. Además, Kubernetes dispone de multitud de funcionalidades y mecanismos para facilitar el trabajo sobre el clúster. Una de las características de Kubernetes es que permite añadir o eliminar nodos (dispositivos hardware) de forma transparente y sencilla, reconfigurando y balanceando los elementos necesarios para que el clúster continúe funcionando de la mejor manera posible. De esta forma, en caso de que una Raspberry se estropee y deje de estar accesible, Kubernetes reconfigurará todo el clúster para dejar de usar dicho nodo y repartir su contenido, que será replicado, entre el resto de los nodos. Y todo esto haciéndolo automáticamente y sin necesidad de intervención humana. Otra característica de gran utilidad es la gestión de roles y permisos que permite hacer, de manera que podamos configurar los usuarios que queramos,

y de la forma que consideremos, y será el propio Kubernetes quien se encargue de validar que las órdenes que se van realizando sobre el clúster son legítimas y pueden llevarse a cabo.

Sin embargo, Kubernetes contiene de base muchas funcionalidades que no son necesarias para un clúster de reducidas dimensiones como es el nuestro. Y es que aquí donde entra **K3s**, que es una versión ligera de kubernetes que está muy bien optimizada y es perfecta para los dispositivos hardware usados. Si bien esta versión de Kubernetes dispone de un menor número de funcionalidades, nuestras necesidades quedan completamente cubiertas, y a un menor coste computacional. O, dicho de otra forma, con K3s podemos usar todas las funcionalidades que nos pueden interesar de Kubernetes, pero con un menor consumo de memoria y CPU, así como teniendo un sistema algo más simplificado.

Al final, Kubernetes es un orquestador de contenedores. Un **contenedor** es *una unidad de software que empaqueta su código y todas sus dependencias de manera tal que la aplicación pueda ejecutarse rápidamente y con gran fiabilidad, independientemente del entorno de ejecución* (Docker, 2020). Por tanto, surge la necesidad de disponer de una forma de crear y ejecutar contenedores, siendo para ello **Docker** la herramienta elegida. Con esta herramienta podemos definir un contenedor, construirlo y dejarlo disponible como imagen a la que posteriormente podemos hacer referencia y usarla en nuestro entorno. Además, Docker dispone de una amplia biblioteca o *hub* de imágenes que son públicas y pueden ser usadas.

Una alternativa a Kubernetes es **Docker Swarm**, que nos permite crear un clúster de motores Docker donde podemos ejecutar los contenedores. Sin embargo, hemos optado por Kubernetes debido a que una funcionalidad muy interesante de la que queremos disponer, como es el auto escalado de aplicaciones, sólo está disponible en Kubernetes. Además, el uso de Kubernetes está más extendido que Docker Swarm y Kubernetes cuenta con una mayor comunidad de usuarios (Mangat, 2019). Por el contrario, Kubernetes es más complejo de cara a iniciar un clúster nuevo, pero trabajos como el presente pretenden mejorar este hecho y reducir el esfuerzo necesario de la instalación y configuración inicial.

La definición de los objetos que queremos importar/desplegar en Kubernetes la hacemos a través del lenguaje **YAML**, guardando dichas definiciones en ficheros que simplemente debemos importar dentro de Kubernetes. YAML es un lenguaje que permite serializar los datos que estamos tratando de una manera estructurada pero fácilmente entendible, basando principalmente su estructura es un conjunto de espacios que, dependiendo de la cantidad de estos, va determinando su nivel en la jerarquía. Otra posibilidad es usar el formato **JSON** para almacenar la información de una forma equivalente. JSON es también fácilmente entendible y también es muy popular. Sin embargo, para este trabajo nos hemos decantado por YAML por preferencia personal más que por ninguna otra posible ventaja.

Como se puede intuir, es necesario establecer algún mecanismo de seguridad en Kubernetes para que no cualquiera pueda realizar cualquier operación. Para ello, Kubernetes permite gestionar la autenticación y autorización de varias formas, pero para este trabajo nos hemos decantado por **OIDC**, como protocolo seguido para la autenticación con el clúster. OIDC nos permite usar las credenciales de la universidad para autorizar usuarios, eliminando la necesidad de disponer de usuarios generales que son más difíciles de restringir y seguir el registro de actividad, al no estar sujetos a individuos particulares.

Por otro lado, resulta muy importante poder realizar una monitorización del clúster en sí para poder detectar problemas o consultar el estado general del mismo en cualquier momento. De esta forma, podemos conseguir un clúster más robusto ya que podemos anticipar problemas y ponerles solución antes de que ocurran, o minimizar su impacto (por ejemplo, reducir el tiempo en el que un recurso no está disponible). Para conseguir estos objetivos hemos elegido varias herramientas, que conjuntamente permiten alcanzar los objetivos mencionados. Así, hemos elegido **Prometheus** como herramienta para la recopilación de métricas de todo tipo generadas dentro del clúster, que permite la consulta y agrupación de estas métricas. Además, tiene la capacidad de poder definir unas condiciones en base a unas métricas que, cuando se den dichas condiciones, genere una alerta. Estas alertas son recibidas por la herramienta **Alert manager**, que, en base a su configuración, decide qué hacer con ellas. Podemos resumir que lo que esta herramienta hará será enviar notificaciones a diferentes canales a los destinos configurados, en función del tipo de alarma

y la severidad de esta. En realidad, permite mayores y más complejas configuraciones, pero eso lo veremos más en detalle en el punto 4.3.2. Por último, la herramienta **Grafana** nos permite crear un *dashboard* para visualizar todas las métricas, o conjuntos de estas, que queramos para poder disponer de una forma visual de la situación del clúster en el momento que deseemos.

Además de la monitorización, gestionar los logs de una forma correcta es un requisito importante. Los contenedores son volátiles y suelen destruirse y crearse nuevo con cierta frecuencia. Los logs que van generando los contenedores se generan dentro de sí mismos y por tanto sólo existirán hasta que el contenedor se destruya, perdiéndose los logs en ese momento. Esto puede ser un problema, sobre todo para analizar errores que provocan reinicios de aplicación, ya que se tendrían que consultar los logs justo antes de que se pierdan, pero no suele ser sencillo predecir los errores. Además, el hecho de tener los logs dentro de los contenedores nada más, no permite hacer una agrupación o búsqueda de varios logs al mismo tiempo, algo que puede ser muy interesante (por ejemplo: buscar un error en concreto en todas las instancias de una misma aplicación). Y como los logs dentro de los contenedores están almacenados como texto nada más, no es sencillo preparar un sistema que pueda detectar patrones concretos en los logs y generar alertas cuando estos patrones se detecten. Estos problemas se pueden solucionar exportando todos los logs a un mismo sitio, y tener implementados en este sitio todos los requisitos que necesitemos. Así es como resulta muy interesante contar en el clúster con un sistema de exportación y gestión de logs, lo que podemos conseguir con el conjunto de herramientas Fluentd, Elasticsearch, Kibana y Kibana Alerts. **Fluentd** es un recolector de datos que se conecta a cada uno de los nodos para seguir los logs que se van generando en los contenedores, y tras un filtrado y transformación los envía a **Elasticsearch**, que es un motor de búsqueda en tiempo real, distribuido y escalable, que al recibir los logs los indexará para tenerlos disponibles para su consumo. Quien los consume será **Kibana**, que es una aplicación de visualización que lanza las búsquedas que hacemos y/o guardamos a Elasticsearch y muestra los datos en el formato que consideremos, ofreciendo múltiples posibilidades: logs tal cual, gráficas, tablas, agrupación de logs, filtrado por ciertos campos y/o texto del log, etc. (Jetha, 2020). Hay otra aplicación más que podemos instalar, **Kibana alerting**, que permite configurar alertas en base a la información proveniente

de los logs, pero aplicando la búsqueda y filtro que consideremos para decidir si la alerta se debe disparar o no.

Debemos hablar también del lenguaje de programación **JavaScript** y del entorno de ejecución en el lado de servidor **NodeJS**, ya que sobre ellos han sido sobre los que hemos realizado las implementaciones de las aplicaciones realizadas.

También hemos usado una instancia de base de datos **Redis**, como almacén de las sesiones correspondientes a procesos de autenticación en curso, así como de aquellas sesiones que terminaron con una autenticación correcta.

Por otro lado, **Node-RED**, que es una herramienta común en el mundo IoT ya que permite el desarrollo de forma sencilla de aplicaciones ligeras que puedan ser usadas y consumidas por estos dispositivos. Node-RED realmente es una aplicación desarrollada en JavaScript y que es ejecutada en NodeJS, y que para nuestro trabajo es tratada simplemente como una imagen Docker ya existente que usaremos para ilustrar el caso de instalación de aplicaciones potenciales destinadas para uso de los usuarios del clúster.

En una gestión de un clúster seguramente haya claves o credenciales que deban ser custodiadas con seguridad, y para ello se pueden seguir varias vías, como pueden ser el uso de objetos de tipo secreto en kubernetes o un almacén seguro de credenciales (Osnat, 2019). Como almacén seguro para credenciales hemos considerado **Vault**, que almacena los datos internamente encriptados por defecto, y que permite una gestión de acceso, revocación y renovación muy sencilla. No obstante, la solución que tome en relación a este sentido debe estar alineada con el proyecto eNMoLabs, ya que este tipo de decisiones y soluciones deben dar soporte general al proyecto y no deben implementarse para piezas independientemente.

3.2. Relación de las tecnologías en el trabajo actual

Para nuestro trabajo hemos elegido y usado las siguientes tecnologías de entre las mencionadas en el punto anterior:

- Varias unidades de **Raspberry Pi** como hardware sobre el que montaremos el clúster.
- **Kubernetes**, y en concreto **K3s**, como gestor del clúster y orquestador de **contenedores** que iremos montando en el mismo.
- Los contenedores estarán basados en **Docker**, y, en su mayoría estarán alojados en *hub* de Docker.
- Para la definición de ficheros con los que queremos iremos cargando objetos en el clúster usaremos el lenguaje **YAML**.
- Como mecanismo de autenticación para el clúster usaremos **OIDC**.
- Para la monitorización usaremos **Prometheus** para recabar métricas del clúster y estar pendientes a situaciones excepciones en base a los valores de dichas métricas, **Alert manager** para gestionar qué hacer con las alertas que se vayan produciendo y, si es necesario, enviar notificaciones, y **Grafana** para preparar un *dashboard* que muestre de forma visual el estado del clúster.
- Como complemento a la aplicación OIDC que veremos más adelante, usamos **Redis** para almacenar información sobre las sesiones que se gestionan en dicha aplicación.
- Los desarrollos de aplicaciones que hagamos los haremos en el lenguaje **JavaScript** pensando en el entorno de ejecución **NodeJS**.

3.3. Resumen del capítulo

En este capítulo hemos hecho un repaso a todas las tecnologías usadas o consideradas para este trabajo.

Para ello, primeramente, hemos ido poniendo en contexto cada una de las tecnologías que se han considerado, explicando su utilidad y para qué se han considerado en nuestro

trabajo. Finalmente, a modo de resumen esquemático hemos indicado qué tecnologías se han usado en el clúster y para qué.

Así, el lector puede tener una idea de para qué sirven las tecnologías y su contexto dentro del trabajo, y luego un esquema con las que se han usado y para qué.

4. Implementación llevada a cabo

En este capítulo detallaremos la arquitectura implementada, así como los desarrollos en materias de seguridad y de monitorización, y finalmente mostraremos un resumen de cómo queda organizado el clúster una vez que se ha preparado y qué recursos disponibles quedan para poder ser explotados por sus diferentes usos.

4.1. Arquitectura a nivel de hardware

Para este proyecto hemos dispuesto de 4 Raspberry Pi de diferentes modelos y características que a continuación se detallan. Una de ellas ha sido destinada a un rol *master* (nombre asignado *rpi-master*) o gestora del clúster, y las otras 3 se han destinado a un rol *worker* (nombres *rpi-worker1*, *rpi-worker2* y *rpi-worker3*), o nodo de trabajo. Hay que aclarar que la Raspberry destinada al rol *master* también realiza tareas de trabajo.

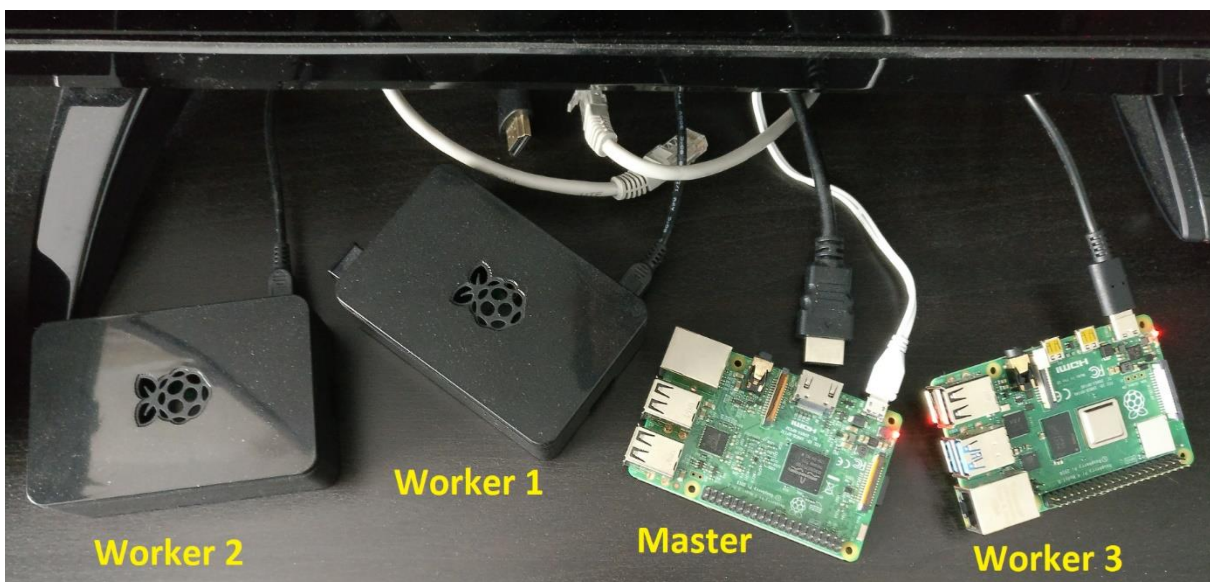


Figura 4-1: foto de las Raspberry Pi usadas para el presente trabajo

A cada una de las Raspberry se le ha asignado una IP fija dentro de la red interna, para evitar posibles problemas de funcionamiento del clúster una vez iniciado, y para simplificar el mapa de red.

En la Tabla 4-1 vemos los nodos usados y cómo han quedado configurados:

Nombre	Roles	Versión	IP interna	Imagen sistema operativo	Versión de kernel
rpi-master	master	v1.18.6+k3s1	192.168.178.30	Raspbian GNU/Linux 10 (buster)	4.19.118-v7+
rpi-worker1	worker	v1.18.6+k3s1	192.168.178.31	Raspbian GNU/Linux 10 (buster)	4.19.118-v7+
rpi-worker2	worker	v1.18.6+k3s1	192.168.178.37	Raspbian GNU/Linux 10 (buster)	4.19.118-v7+
rpi-worker3	worker	v1.18.6+k3s1	192.168.178.36	Raspbian GNU/Linux 10 (buster)	4.19.118-v7l+

Tabla 4-1-Extracto de la información obtenida del comando kubectl get nodes -o wide *donde vemos cada nodo con su rol, versión de k3s, su IP interna asignada, la imagen del SO y la versión de kernel*

En la Figura 4-2 podemos ver el esquema de la infraestructura que se ha montado con las Raspberry:

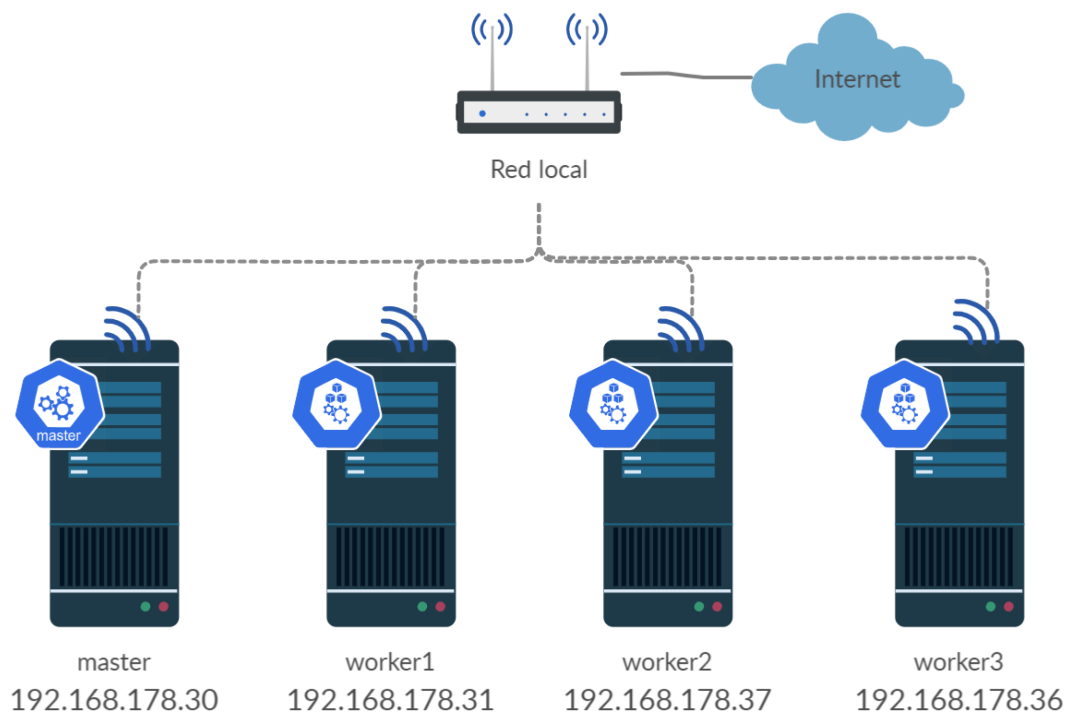


Figura 4-2: diagrama hardware de las Raspberry y la configuración de red

En las siguientes tablas Tabla 4-2, Tabla 4-3 y Tabla 4-4 podemos ver las características técnicas más importantes de cada una de las Raspberry Pi, con la información extraída directamente desde cada una de ella por medio de los comandos indicados:

<table>
<tr><th colspan="2">rpi-master</th></tr>
<tr><td>Modelo:</td><td>$ cat /sys/firmware/devicetree/base/model
Raspberry Pi 3 Model B Rev 1.2</td></tr>
<tr><td>Información sobre la CPU:</td><td>$ lscpu
Architecture: armv7l
Byte Order: Little Endian
CPU(s):   4
On-line CPU(s) list: 0-3
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s):   1
Vendor ID: ARM
Model: 4
Model name: Cortex-A53
Stepping: r0p4
CPU max MHz: 1200.0000
CPU min MHz: 600.0000
BogoMIPS: 76.80
Flags: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32</td></tr>
<tr><td>Información sobre la memoria:</td><td>$ vmstat -s
       948280 K total memory
…</td></tr>
</table>

Tabla 4-2: Características técnicas del nodo rpi-master

<table>
<tr><th colspan="2">rpi-worker1 y rpi-worker2</th></tr>
<tr><td>Modelo:</td><td>$ cat /sys/firmware/devicetree/base/model
Raspberry Pi 3 Model B Plus Rev 1.3</td></tr>
<tr><td>Información sobre la CPU:</td><td>$ lscpu
Architecture: armv7l
Byte Order: Little Endian
CPU(s):   4
On-line CPU(s) list: 0-3</td></tr>
</table>

	Thread(s) per core: 1 Core(s) per socket: 4 Socket(s): 1 Vendor ID: ARM Model: 4 Model name: Cortex-A53 Stepping: r0p4 CPU max MHz: 1400.0000 CPU min MHz: 600.0000 BogoMIPS: 38.40 Flags: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
Información sobre la memoria:	$ vmstat -s 948280 K total memory …

Tabla 4-3: Características técnicas de los nodos rpi-worker1 y rpi-worker2

rpi-worker3	
Modelo:	$ cat /sys/firmware/devicetree/base/model **Raspberry Pi 4 Model B Rev 1.1**
Información sobre la CPU:	$ lscpu Architecture: armv7l Byte Order: Little Endian CPU(s): 4 On-line CPU(s) list: 0-3 Thread(s) per core: 1 Core(s) per socket: 4 Socket(s): 1 Vendor ID: ARM Model: 3 Model name: Cortex-A72 Stepping: r0p3 CPU max MHz: 1500.0000 CPU min MHz: 600.0000 BogoMIPS: 270.00 Flags: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm crc32
Información sobre la memoria:	$ vmstat -s 1985984 K total memory …

Tabla 4-4: Características técnicas del nodo rpi-worker3

En las tablas Tabla 4-2, Tabla 4-3 y Tabla 4-4 vemos el modelo de cada Raspberry, información sobre la CPU (arquitectura, número de núcleos, ciclos por segundo, etc.) y la cantidad de memoria RAM (en KB) de la que disponen. En la siguiente Tabla 4-5 podemos ver de forma agrupada los datos técnicos que más nos van a interesar de cara a la realización del trabajo:

Nodo	Arquitectura	Núcleos	RAM (GB)
rpi-master	ARMv7	4	1
rpi-worker1	ARMv7	4	1
rpi-worker2	ARMv7	4	1
rpi-worker3	ARMv7	4	2
Total		**16**	**5**

Tabla 4-5: arquitectura, número de núcleos y RAM de cada nodo del clúster

Así pues, nuestro clúster contará con 16 núcleos y 5GB de capacidad. Y todas las imágenes de contenedores que queramos usar deberán estar compiladas para la arquitectura ARMv7, ya que todos los nodos tienen esta arquitectura.

La Raspberry *master*, como hemos dicho, es la que gestiona el clúster, y es a ella a la que debemos dirigir todas las órdenes que debamos dar al clúster. En caso de que la orden deba ser gestionada por otra Raspberry, será la *master* la que coordine a quién debe ir destinada la acción y se la transmitirá. De esta forma, el usuario del clúster sólo accederá a la master para cualquier cosa que necesite hacer. Por tanto, un pequeño esquema de cómo se ve el clúster, desde el punto de vista de quién opera a nivel kubernetes en él, se puede ver en la Figura 4-3:

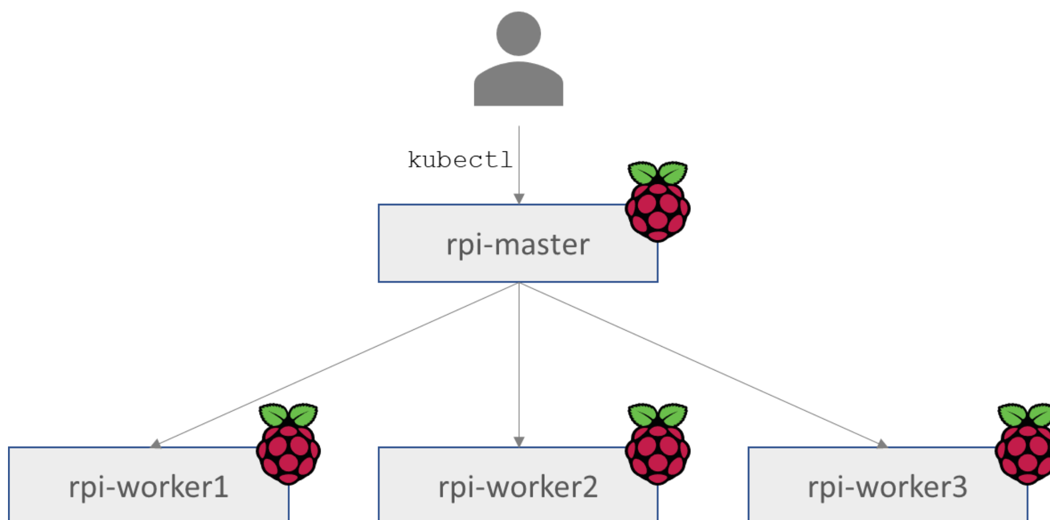


Figura 4-3: esquema de organización de los nodos del clúster y cómo accede un usuario del clúster al mismo

Si bien, en el clúster propuesto, la Raspberry que tiene las mejores prestaciones a nivel de memoria RAM y un ligero mayor número de ciclos de procesador por segundo es, la *rpi-worker3*, ésta se ha destinado al rol de *worker* en lugar de *master*. Esta decisión no tiene por qué ser así, y cualquier otra combinación de roles asignados a cada nodo sería posible. Como detallaremos después, no hemos restringido el despliegue de aplicaciones a según qué nodos en función de los *namespace*[1] donde se encuentren debido a que no disponemos de una capacidad suficiente en los nodos disponibles, y por tanto no hay una estrategia de distribución de carga que marque la diferencia entre destinar las Raspberry más potentes en un rol u otro.

En cualquier caso, y tal como hemos visto en la Tabla 4-5, nuestro clúster cuenta con un total de 16 CPUs o cores, y una memoria de 5GB, lo que lo convierte en un clúster pequeño, pero suficiente para un uso controlado y limitado a varias asignaturas y prácticas. Posteriormente, en los trabajos futuros, propondremos la posibilidad de auto escalado basado en tiempo, que podría permitir disponer de muchos más despliegues dentro del mismo

[1] Un clúster de kubernetes se organiza en espacios independientes entre sí, aunque con comunicación entre ellos si es preciso. De esta forma, se pueden establecer medidas de seguridad y otras políticas que apliquen solo a los espacios que interesen. Además, permite mantener una separación entre entidades distintas, como podrían ser las aplicaciones del sistema y las aplicaciones de unas prácticas de una asignatura, por ejemplo

clúster, pero estando disponibles únicamente a ciertas horas o días, lo que podría ajustarse a un uso por asignaturas.

Las Raspberry Pi usadas están destinadas única y exclusivamente al clúster, por lo que sus recursos están disponibles en su totalidad al mismo. En caso de que en alguna de las Raspberry hubiese algún proceso corriendo o se destine a algún otro uso adicional, habría que tenerlo en cuenta de cara a los posibles conflictos en el uso de los recursos en la propia Raspberry. No obstante, en la medida de lo posible, es recomendable dedicar las Raspberry que vayan a integrar el clúster en exclusiva al mismo, no ya solo por motivos de recursos sino también por motivos de seguridad y de integridad. Para entender mejor esta última afirmación pensemos simplemente que cuantos más procesos estén en funcionamiento en un momento dado, mayor es la posibilidad de que alguno de ellos falle, y en caso de que el fallo afecte a algo más que el propio proceso, podría impactar de alguna manera al clúster en lo que se refiere al nodo. Por otro lado, y con el mismo razonamiento, un mayor número de procesos corriendo incrementa la posibilidad de vulnerabilidad a nivel de seguridad, y una explotación de alguna vulnerabilidad de un proceso puede afectar a otros procesos, como el clúster para ese nodo. Poniendo un ejemplo sencillo, supongamos que en uno de los nodos destinados al clúster montamos también un servidor web con una aplicación PHP. En este supuesto, una vulnerabilidad (por ejemplo, escalado de permisos a nivel de sistema) o malfuncionamiento (por ejemplo, uso excesivo de memoria RAM) del servidor web podría afectar al nodo aun cuando el nodo ha funcionado en todo momento correctamente.

Por motivos de seguridad -ver (Michael Hausenblas, 2018)-, ninguna de las Raspberry Pi debería estar expuesta directamente a una red no controlada y restringida. Esto es, las Raspberry Pi en sí sólo deben quedar accesibles desde una red interna del laboratorio que las gestione, o desde una VPN, pero nunca deberían quedar accesibles desde internet o la red general de la universidad, ya que, de ser así, serían susceptibles de recibir ataques. Y su exposición no es necesaria para el correcto uso del clúster por parte de los consumidores. Sin embargo, aunque el acceso deba estar restringido a nivel de red (red interna, VPN, etc.), para trabajar con el clúster en sí no es necesario ni recomendable conectarse a la Raspberry que haga de máster, ya que esto supondría una gestión de credenciales en el lado de la Raspberry en sí para su acceso por SSH a los usuarios que corresponda y un incremento de riesgos de

seguridad al tener más puntos que cubrir. Para poder trabajar con el clúster, pero desde un ordenador local se puede configurar la URL de la API del clúster, que será hacia donde se envíen todas las peticiones (ver el anexo C para más información sobre cómo hacer esta configuración).

Por otra parte, las aplicaciones que se creen dentro del clúster sí deben estar accesible desde el exterior, en las redes en las que se desee que estén accesibles. El cómo se configure este punto dependerá de la estructura final en la que se trabaje, aunque como ejemplo de lo realizado en este trabajo se puede consultar el anexo B.

Cada Raspberry cuenta con una tarjeta de memoria *microSD* de 64GB de capacidad, si bien no estamos usando ni 10GB en ninguno de los nodos. De cara a un correcto dimensionamiento en cuanto a capacidad hay que tener en cuenta que con tarjetas de 16GB será suficiente para nodos con un uso que no ofrezca la posibilidad de volúmenes persistentes. Y en caso de que sea necesario habilitar volúmenes persistentes, estos deberían estar dispuestos únicamente sobre nodos que tengan una mayor capacidad. No obstante, los precios de las tarjetas *microSD* son bastante reducidos actualmente para capacidades como la usada, 64GB, lo que puede permitir una mayor flexibilidad en el clúster.

En el anexo A se detallan los pasos a seguir para la integración de un nodo dentro del clúster, y para la creación del primer nodo, el *master*, para un clúster nuevo.

Idealmente, debería haber más de una Raspberry *master* para un clúster kubernetes, y en general se recomienda que sea un número impar para ser capaz de resolver siempre cualquier posible conflicto en la toma de decisiones distribuidas (Data, 2020), (Strittmatter, 2018) y (Vallim, 2019). Por este motivo, las combinaciones más comunes suelen ser de 3 o 5 Raspberry destinadas a ser máster. Esto es así por si ocurriese que el hardware de una *master* falle, que no afecte al clúster ya que puede haber otro nodo que la sustituya. En nuestro clúster esto no es posible hacerlo, aunque contásemos con más Raspberry, porque actualmente k3s no lo permite, aunque es esperable que a futuro sí se incluya esta posibilidad. Hasta entonces se desaconseja rotundamente el uso de este tipo de clúster para actividades o servicios críticos. Sin embargo, el uso para prácticas de laboratorios, repositorios de

información para alumnos, exposición de servicios que puedan ser consumidos por alumnos, por mencionar algunos ejemplos, sí parecen usos que pueden ser satisfechos sin problema en este tipo de clúster.

En caso de que el clúster se vaya a querer usar para alguna actividad más crítica o que simplemente se quiera montar de una manera que pueda haber más de 1 nodo máster, se puede llevar a cabo la instalación oficial de Kubernetes (Mathias Renner, 2017) en lugar de alternativa más ligera que estamos usando, k3s. Esto implicaría que el clúster, tras la preparación inicial, contaría con menos recursos disponibles para su explotación (Mackenzie, 2020).

Con relación al punto anterior, es muy recomendable crear imágenes de la tarjeta de memoria de la Raspberry *master* con frecuencia, para que en caso de que fallase, pueda ser fácilmente recuperable el estado que hubiese cuando se hizo la última imagen (aunque seguramente si se cambia la Raspberry en sí es probable que hubiese que hacer otras tareas, como asociar la misma IP que tenía la *master* anterior a la MAC de la nueva Raspberry, etc.).

Sobre los requisitos hardware mínimos de la arquitectura montada, en la Figura 4-1 vemos las Raspberry que se han usado. En la imagen podemos ver que a cada Raspberry sólo llega un cable, el de la corriente. En realidad, ese es el único requisito, ya que todo el proceso de instalación, puesta en marcha y operación se puede hacer en remoto vía SSH, y las Raspberry se pueden conectar vía WiFi a la red que corresponda, lo que se puede configurar tras la preparación inicial de la tarjeta de memoria. También se puede ver en la imagen, sin embargo, que hay un cable HDMI que está conectado en el otro extremo a la televisión (que hace las veces de monitor) que hay al lado, lo que junto con un ratón y teclado inalámbrico nos ha permitido trabajar directamente desde una de las Raspberry para solucionar algún problema o simplemente para realizar comprobaciones cuando así lo hemos requerido.

Por último, en el anexo D se pueden encontrar un listado de problemas surgidos durante el desarrollo de este trabajo, que quizás puedan ser de utilidad a futuros administradores de clústeres similares.

4.2. Arquitectura de autenticación y autorización en el clúster

El clúster, tal cual se instala inicialmente, basa la autenticación en un usuario y contraseña fijos, siendo dicho usuario administrador del clúster. Sin embargo, este usuario no debería usarse tal como se propone en (Michael Hausenblas, 2018), ya que, si ese usuario se viera comprometido, todo el clúster podría estar comprometido. En general, no está recomendado el uso de usuarios genéricos, ya que no permiten controlar quién hace qué, ni restringir un conjunto de operaciones a ciertas personas, entre otras cosas.

La mejor estrategia se basa en usuarios personales, agrupados en roles, y fijar los permisos sobre los roles. De esta forma, cada persona es responsable de su usuario, y los permisos no están basados en usuarios concretos, lo cual es una fuente de problemas al olvidar quitar permisos cuando ya no son necesarios, sino en roles, siendo la gestión de roles más intuitiva y, por lo general, más sencilla. Además, idealmente se usarán los propios usuarios de la universidad, lo que nos dará la ventaja de no tener que almacenar ninguna información sobre credenciales y sabiendo que el estado de las credenciales será siempre el correcto, asegurando que cuando un usuario se dé de baja ya no será válido.

Las posibilidades de autenticación que ofrece Kubernetes son:

- Certificados de cliente. Esto implicaría un sistema de generación de certificados y de cancelación de estos cuando ya no sean necesarios.
- Token. Aquí puede haber varios tipos de tokens (token de aplicación kubernetes, token constante, token obtenido vía OIDC…).
- Proxy de autenticación. Esto implicaría disponer de una aplicación que valide los datos del usuario entrante, pero en el caso que nos ocupa implicaría que en cada petición tendría que haber un sistema que o bien se comunique con los servidores de la UNED o bien gestionase las sesiones de forma interna.
- Usuarios y contraseña. Esto implicaría una gestión de usuarios al margen de las credenciales de UNED.

Pensando en la gestión de roles y buscando poder sacar provecho de las credenciales propias de la universidad, tras analizar las alternativas que kubernetes ofrece para autenticar

y autorizar (Kubernetes, 2020) (autenticacion, 2020), hemos optado por un tipo de token, en concreto el generado por OIDC (OpenID Connect, más información en (OIDC, 2020)).

OIDC *permite a clientes verificar la identidad de usuarios finales basando la autenticación en un servidor de autorización, así como para obtener un perfil básico sobre el usuario final* (OIDC, 2020). Básicamente, nos permite poder autenticar usuarios en una aplicación externa al cliente, kubernetes en este caso, y devolver confirmación del éxito de dicha autenticación, así como información del perfil del usuario, lo que nos ayuda a perfilar la autorización. De esta forma, podemos usar la autorización que consideremos, en este caso con las credenciales de la UNED, y añadir la información que queramos, como el conjunto de grupos al que pertenece el usuario. Y todo esto sin que Kubernetes interfiera en nada, más allá de iniciar el proceso de autenticación. Además, Kubernetes será capaz de extraer el conjunto de grupos al que pertenece el usuario y así saber qué permisos aplican al usuario.

Para realizar esta autenticación con OIDC hemos creado una aplicación nueva que gestiona toda la capa de OIDC, e invoca a la página de autenticación de la UNED para validar las credenciales (usuario y contraseña) introducidos por el usuario. De esta forma, podemos gestionar una autenticación a través de las credenciales de la UNED y dar como resultado un token válido para operar en kubernetes.

La autorización, es decir, qué usuarios tienen qué accesos, se ha diseñado basándose en roles en el lado de kubernetes, y definiendo los roles según el dominio del email o a nivel de usuario particular. Posteriormente se verá más en detalle cómo funciona, tanto en el lado de OIDC, como en el de kubernetes.

El esquema de autenticación y autorización que realiza kubernetes para cada petición que se hace al clúster se puede ver en la Figura 4-4:

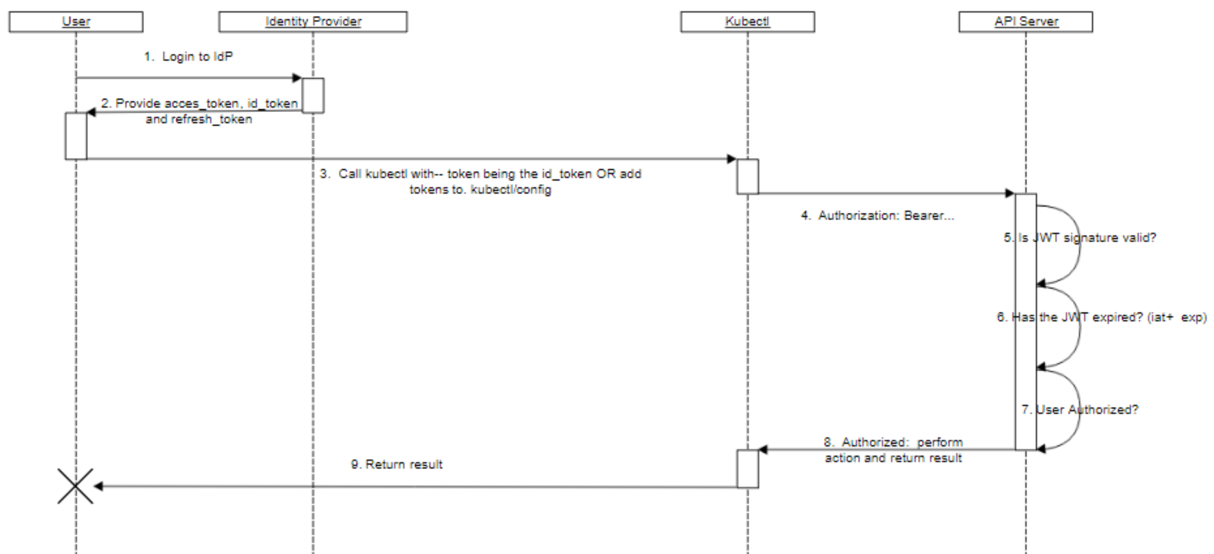


Figura 4-4: esquema de autenticación y autorización en kubernetes por cada petición. Imagen extraída de (Kubernetes, 2020)

Tras la autenticación en el proveedor de identidades (*Identity Provider*) se dispone de un token, *id_token*, con el que se realizarán todas las llamadas vía la línea la aplicación de control de Kubernetes *kubectl*. Y ya dentro del clúster se valida que el *id_token* sea correcto y sea válido.

El token resultado de la autenticación, conocido como *id_token* ya que su uso se limita a identificar al usuario logado, se debe almacenar en la configuración local de kubernetes del usuario que está realizando el proceso. Este token contiene la información suficiente para poder validar la autenticación en cada petición y poder comprobar la autorización que le corresponde al usuario activo.

Adicionalmente, y para facilitar el uso del proceso de autenticación a los usuarios, se ha creado otra aplicación que actúa a modo de cliente, que se encarga de orquestar las acciones oportunas para que el usuario sólo necesite iniciar dicho cliente e introducir sus credenciales cuando corresponda.

Decir que para el desarrollo del trabajo se ha usado un único usuario, perteneciente al autor, con un rol administrador. Para poder hacer uso de este rol ha sido necesario configurar la asociación del rol preexistente admin del clúster al grupo admin proveniente de la autenticación, lo que hemos hecho con la configuración que se puede ver en la Figura 4-5:

```yaml
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: oidc-cluster-admin
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: "rbac.authorization.k8s.io"
subjects:
- kind: Group
  name: oidc:cluster_admin
```

Figura 4-5: configuración de la asociación del rol admin al grupo admin proveniente de la autorización. Fichero 40-cluster_admin.yaml

4.2.1. Aplicación OIDC para la autenticación y gestión de autorización

En esta sección vamos a explicar todo lo realizado respecto a la nueva aplicación OIDC creada para la autenticación de usuarios en el clúster. Para mayor claridad, se ha dividido la sección en apartados que se centran cada uno en un aspecto en concreto.

4.2.1.1. Despliegue de la aplicación

La aplicación se despliega por medio de un objeto `Deployment`[2] de kubernetes, que contiene las secciones principales mostradas en la Figura 4-6:

```yaml
apiVersion: apps/v1
kind: Deployment
…
spec:
  replicas: 2
…
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: "app"
                    operator: In
                    values:
```

[2] Un objeto de clase `Deployment` se usa en Kubernetes para definir y actualizar aplicaciones definidas como contenedores.

```
              - oidc
        topologyKey: "kubernetes.io/hostname"
…
imagePullSecrets:
- name: docker.io
containers:
- name: oidc
  image: atellezr/uned-oidc:latest
  imagePullPolicy: Always
  ports:
  - containerPort: 4400
    name: http
  livenessProbe:
    initialDelaySeconds: 10
    periodSeconds: 10
    timeoutSeconds: 5
    successThreshold: 1
    failureThreshold: 3
    httpGet:
      path: /oidc/_health/status
      port: http
  readinessProbe:
    initialDelaySeconds: 10
    periodSeconds: 10
    timeoutSeconds: 1
    successThreshold: 1
    failureThreshold: 3
    httpGet:
      path: /oidc/_health/status
      port: http
  envFrom:
  - configMapRef:
      name: seg-config
  volumeMounts:
 - mountPath: /etc/groups-cfg
    name: groupsmap-vol
    readOnly: true
  - mountPath: /etc/local-users
    name: usersmap-vol
    readOnly: true
  resources:
    requests:
      memory: "64Mi"
      cpu: "5m"
    limits:
      memory: "128Mi"
      cpu: "100m"
volumes:
- name: sslcert-miclusternet-vol
```

```yaml
    secret:
      secretName: cert-miclusternet
  - name: groupsmap-vol
    configMap:
      name: map-grupos
  - name: usersmap-vol
    configMap:
      name: map-usuarios-locales
  hostAliases:
  - ip: "192.168.178.30"
    hostnames:
    - "rpi-master"
```

Figura 4-6: configuración del despliegue de la aplicación OIDC. Fichero 30-uned-oidc.yaml

Explicando los puntos más importantes del despliegue mostrado en la Figura 4-6:

- Se definen 2 réplicas (pods) para la aplicación. Esto quiere decir que la aplicación arrancará con 2 copias iguales. Puesto que esta aplicación es clave para el uso del clúster, es mejor tener más de 1 réplica, en caso de que pudiera haber algún problema puntual en una réplica. Pero 2 réplicas deberían ser suficientes para la aplicación. Y en cualquier caso, como se verá en el siguiente apartado, se pueden ampliar el número de réplicas.
- Se define una afinidad de tipo anti-afinidad a nivel de pod que indica que los pods deben ir a nodos distintos. Esto asegura que si por casualidad un nodo se cae afectará como mucho a 1 réplica, dejando otra réplica (porque hemos definido 2 réplicas) activa en todo momento. Y si eso ocurriese, en seguida se preparará otra réplica que la sustituirá.
- Se carga el *secret* llamado `docker.io`, que contiene mi usuario y contraseña del *hub* de Docker, para poder recuperar la imagen Docker, al tratarse de un repositorio privado en dicho *hub*. Ir al anexo F para más información sobre cómo generar este secreto en caso de que sea la primera instalación.
- Se define un único contenedor cuya imagen apunta a la última versión de la imagen Docker. Es una mejor práctica definir una versión en particular en lugar de apuntar a *la última*, ya que, para bien y para mal, en cuando se actualice la imagen Docker, en el siguiente reinicio de la aplicación, se traerá la nueva imagen. Para tener el entorno productivo más controlado se suele apuntar a una versión en concreto y llevar a cabo los cambios únicamente cuando se

desee, y no en un reinicio que puede no ser planeado. No obstante, en este caso, las versiones de esta imagen están totalmente controladas y siempre nos va a interesar usar la última versión disponible.

- Se definen las pruebas de disponibilidad inicial y de estado. Estas pruebas se hacen al arrancar la aplicación (cada réplica) para comprobar que la aplicación arranca correctamente, y luego se hace cada 10 segundos (porque así se ha configurado en este caso) para asegurar que cada réplica está funcionando correctamente. En caso de que la aplicación se quede colgada o, en general, deje de funcionar, kubernetes reiniciará la réplica. El *path* definido se corresponde con un servicio particular creado dentro de la aplicación OIDC. Este servicio siempre responde con un *OK*, salvo cuando la aplicación no esté levantada o no responda por algún problema, que es justo la situación que queremos detectar.
- Indica que las variables de entorno vendrán del *ConfigMap*[3] llamado `seg-config`, que es del que habíamos hablado en el apartado 4.2.1.4.
- Monta 2 volúmenes para que estén disponibles en las réplicas como directorios normales. Y estos volúmenes se corresponden con los *ConfigMap* mencionados en los apartados 4.2.1.5 y 4.2.1.7, para que las configuraciones sobre los roles a aplicar y el listado de usuarios locales estén disponibles dentro del a aplicación.
- Por último, define un alias para un host. Lo que queremos hacer con esto es indicar que la DNS `rpi-master` se resuelva con la IP `192.168.178.30`. Esto lo hacemos porque por defecto los nodos resuelven las IPs en función del clúster, y o bien añadimos una excepción a nivel de clúster, o bien tendríamos que hacer una configuración especial para esta aplicación, y puestos a hacer una configuración particular para el clúster, es mejor hacerla si es posible directamente donde aplica. El uso que puede tener es para resolver la ruta hacia la instancia de Redis, que está definida como `redis://rpi-`

[3] Un objeto de clase ConfigMap en kubernetes sirve para almacenar cierta información de forma constante. Se suelen usar para guardar variables de entorno para aplicación o para otros tipos de configuración. Generalmente se suelen referenciar en el despliegue de la aplicación para que al arrancar la aplicación, ésta tome los valores de dicho objeto y los almacene según corresponda para que puedan ser usados posteriormente durante la ejecución de la aplicación.

`master:30001`. Podríamos configurar esta URL directamente como la IP destino, pero por claridad hemos preferido hacerlo de esta manera.

Por otro lado, también es necesario exponer la aplicación para que pueda dar servicio, y para ello necesitamos un `Service`[4], que configuramos de esta forma:

```yaml
kind: Service
apiVersion: v1
metadata:
  labels:
    app: oidc
    asignatura: seguridad
  name: seguridad-oidc
  namespace: seguridad
  annotations:
    traefik.ingress.kubernetes.io/affinity: "true"
    traefik.ingress.kubernetes.io/session-cookie-name: "sticky"
spec:
  ports:
    name: web
    port: 8044
    targetPort: http
  selector:
    app: oidc
```

Figura 4-7: configuración del servicio de la aplicación OIDC. Fichero 30-uned-oidc.yaml

Este servicio se define como un servicio interno al clúster, lo que quiere decir que como tal no se puede invocar desde fuera del clúster, pero para ello se incluye la configuración relacionada con *traefik*, que es nuestro *Ingress*[5] para la gestión de exposición de servicios de forma sencilla. Esto lo vamos a entender mejor viendo la configuración relacionada para *traefik* de la Figura 4-8.

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: seguridad
  name: seguridad-oidc-ingress
```

[4] Un objeto de tipo Service nos sirve para poder exponer de forma interna al clúster una aplicación. Hay varios tipos de exposición que son posibles con este tipo de objeto, pero en todos los casos del presente trabajo lo hemos usado para exponer de forma interna al clúster la aplicación que corresponda.

[5] El objeto de tipo Ingress sin embargo se encarga de establecer unas reglas a nivel de peticiones que llegan al clúster que cuando se den, redirige el tráfico al objeto Service que se haya definido para esa regla. En caso de que se considere, se puede poner un balanceador de carga por encima de este tipo de objeto. Pero en este trabajo, y por simplificar, consideramos que este objeto nos permite la exposición hacia fuera del clúster.

```
  annotations:
    kubernetes.io/ingress.class: "traefik"
spec:
  rules:
  - http:
      paths:
      - path: /oidc
        backend:
          serviceName: seguridad-oidc
          servicePort: web
```

Figura 4-8: configuración de la exposición del servicio de la aplicación OIDC. Fichero 30-uned-oidc.yaml

Lo que definimos aquí es que todo lo que llegue a `/oidc` se mande al backend expuesto por el servicio `seguridad-oidc`, que es el que habíamos creado anteriormente. Y todo lo definido como *ingress* estará expuesto al exterior al clúster a todas las DNS/IPs que apunten al clúster. Aunque esto se explicará con más detalle en el anexo 0, podemos decir que tenemos una DNS que apunta al clúster, `https://tfm-atellez.ngrok.io`, y con esta configuración de *ingress* lo que conseguimos es que toda petición enviada a una URL que empiece por `https://tfm-atellez.ngrok.io/oidc` se mande a la aplicación OIDC, tal como queremos.

En el servicio también hemos incluido un par de anotaciones que permiten asegurar que haya afinidad de sesión. De forma resumida, la afinidad de sesión quiere decir que un mismo usuario acceda siempre a la misma réplica de la aplicación en todos sus accesos, asegurando que el posible estado en el lado de servidor se mantenga y sea accesible para dicho usuario. O, dicho de otra forma, cuando la aplicación no es completamente *stateless* (sin estado), es necesario que un usuario acceda siempre a la misma réplica, donde se encuentre almacenado (en memoria, ficheros temporales, o donde sea) la información relativa al mismo. En nuestro caso, hay una caché interna creada una vez que finaliza el proceso de autenticación con los datos del usuario logado. Si el usuario no vuelve a acceder hasta que su token expire no habría ningún problema ni necesidad de asegurar la afinidad, pero puesto que podría darse el caso de un usuario acceda antes de que su sesión termine, es necesario que el usuario acceda a la misma réplica a la que había accedido previamente, que es donde se encuentra su información.

4.2.1.2. Autenticación por parte del usuario

Como se ha dicho anteriormente, el proceso de autenticación de clientes se hace por medio del protocolo OIDC, basando la autenticación en sí en la autenticación SSO de la UNED.

Para empezar el proceso de autenticación, debemos acceder desde un navegador al servicio `/auth` del cliente OIDC, y enviando los parámetros oportunos. Un ejemplo de URL podría ser:

```
https://{dominio}/oidc/auth?response_type=id_token&client_id=kubernetes&redirect_uri=https%3A%2F%2F127.0.0.1:8123&scope=openid%20profile%20email%20&nonce=n-0S6_WzA2Mj
```

Ilustración 4-1: ejemplo de URL de entrada a la aplicación OIDC para el comienzo del proceso de autenticación

Los parámetros que se deben enviar son:

- **response_type**: siempre deberá tomar el valor *id_token*, que es para lo único que hemos habilitado la aplicación OIDC en este caso, ya que es lo que nos interesa a nosotros de cara al uso correcto que debemos hacer posteriormente en kubernetes.
- **client_id**: solo hemos habilitado un cliente, y este es *kubernetes*. Una aplicación OIDC estándar suele trabajar con multitud de clientes, configurando en cada uno de ellos diferentes tipos de respuestas, URLs, etc., pero en nuestro caso hemos preparado esta aplicación OIDC únicamente para ser usada por el cliente kubernetes y para la autenticación de este tipo de clúster, y no se pretende que esta aplicación vaya a estar disponible para ningún otro cliente o uso que no sea este.
- **redirect_uri**: esta es la URL a la que se redirigirá una vez que el proceso termine (para bien o para mal), y debe coincidir con una de las URLs definidas en la *whitelist* de la configuración del cliente. En este caso, queremos regresar a la URL https%3A%2F%2F127.0.0.1:8123, que se corresponde con la URL en la que está escuchando el cliente OIDC usado para facilitar el proceso de autenticación (en el punto 4.2.2 se explica en detalle)

- **scope**: aquí indicamos qué alcance queremos tener para este proceso de autenticación. Como se ve, indicamos siempre *openid profile email*, ya que por un lado queremos efectuar un proceso OIDC y por otro lado queremos tener acceso (de cara a que sea posible incluir los datos en la respuesta) de los datos del perfil y el email.
- **nonce**: esto es simplemente un parámetro con un valor aleatorio que sirve para la trazabilidad del proceso en los 2 sentidos (lado OIDC y lado de quién lo invoca), aunque en nuestro caso no le sacamos un gran partido al tratarte de una aplicación usada únicamente para nuestro clúster.

Según accede el usuario a la aplicación OIDC, la aplicación inicializará los datos relativos a este proceso de autenticación, almacenando lo que corresponda en su base de datos Redis, redirigirá al usuario a la pantalla donde debe introducir el usuario y contraseña:

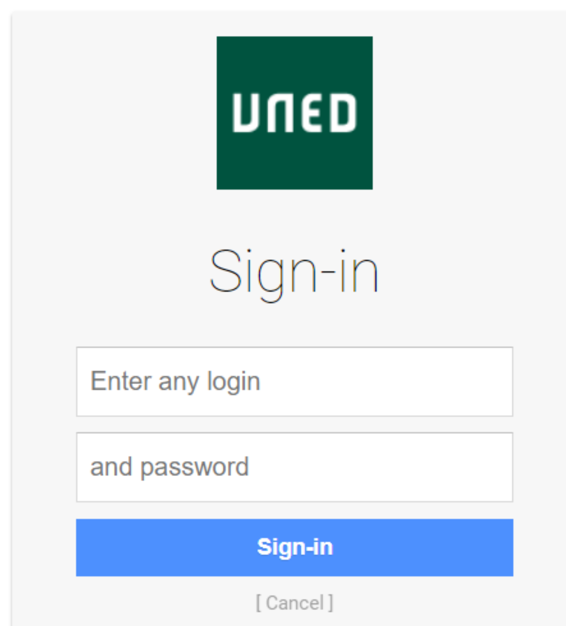


Figura 4-9: pantalla inicial de autenticación

Tras la introducción del usuario y contraseña, la aplicación invoca al servicio de la UNED para su validación, en caso de que sea correcto preparará el conjunto de roles que le aplican al usuario logado y, en cualquier caso, al finalizar ya sea de una forma u otra, la aplicación redirige con el resultado de la operación. Por ilustrar un caso positivo, la URL a la que se redirige es:

```
https://127.0.0.1:8123/#id_token=eyJhbGci...
```

Como podemos ver, es la URL que habíamos definido en la petición inicial, a la que se añade como marcador el *id_token* que necesitamos.

El proceso explicado anteriormente se puede ver con mayor claridad en el siguiente diagrama de la Figura 4-10:

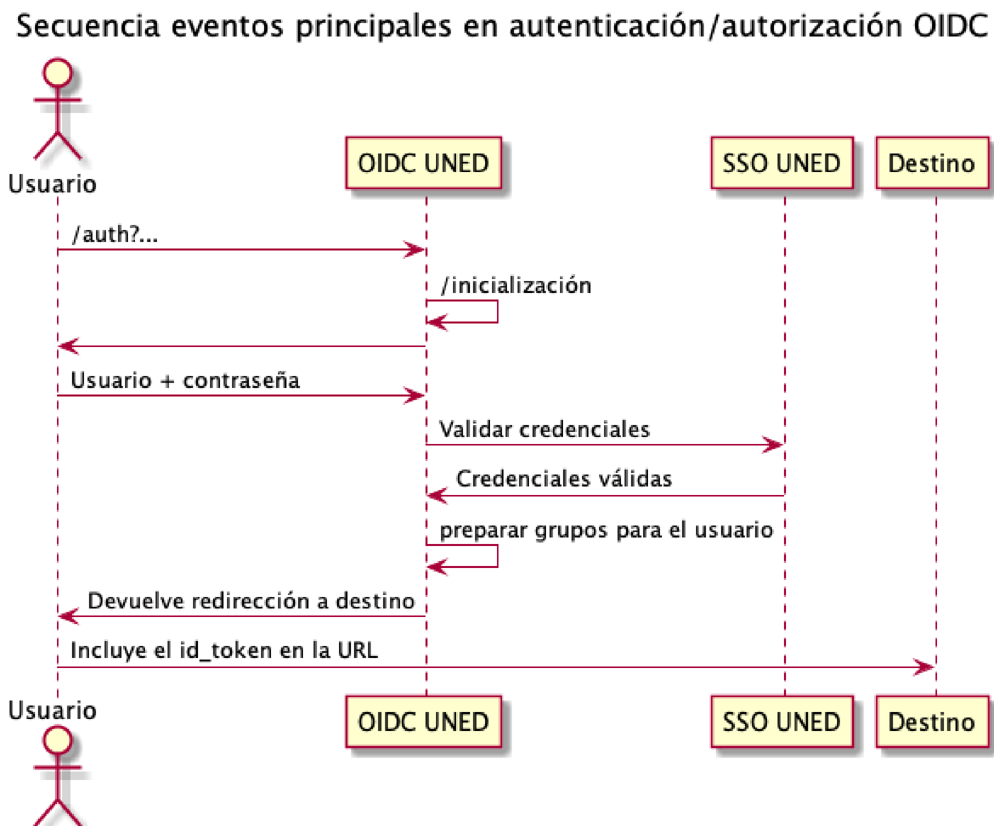


Figura 4-10: diagrama de secuencia de los eventos principales para un proceso correcto de autenticación vía OIDC

En el diagrama de la Figura 4-10 podemos ver diferentes componentes:

- OIDC UNED: esta es la aplicación que hemos creado para la gestión de la autenticación y autorización. Aunque no es estrictamente necesario, la hemos desplegado dentro de nuestro clúster.
- SSO UNED: esto se corresponde con el servidor de la UNED donde se accede vía el Single Sign-On que la UNED ofrece.
- Destino: el destino se corresponde a nuestro clúster de kubernetes, que es para donde se está generando el token.

La autenticación en la UNED queda completamente embebida en el lado del servidor, por lo que el usuario únicamente puede ver el flujo OIDC mencionado.

La aplicación OIDC cuenta con una base de datos propia que se corresponde con una instancia de Redis. Redis es una base de datos que almacena estructuras de datos (Redis, 2020) y que en nuestra implementación se usa como el almacén de sesiones de todos los procesos de autenticación que están en proceso, o de aquellos que han terminado correctamente durante el tiempo de validez de las sesiones (parámetro configurable en la aplicación OIDC).

4.2.1.3. Well-known

Por otro lado, la aplicación también expone otro servicio, `/.well-known/openid-configuration`, que devuelve información sobre la aplicación y su configuración (pública) para que el consumidor pueda acceder a los datos que necesite para poder validar el *id_token* generado.

Al acceder podemos ver mucha información relacionada con el uso de la aplicación y qué opciones ofrece, tal como podemos ver en la Figura 4-11:

Figura 4-11: contenido que devuelve el servicio well-known

De entre todos los datos devueltos que hemos visto en la Figura 4-11 hemos marcado especialmente un dato, `jwks_uri`, ya que su valor hace referencia a la URL que devuelve información pública sobre las claves con la que se generan los tokens (todos los token generados, incluyendo los *id_token*), tal como vemos en la Figura 4-12:

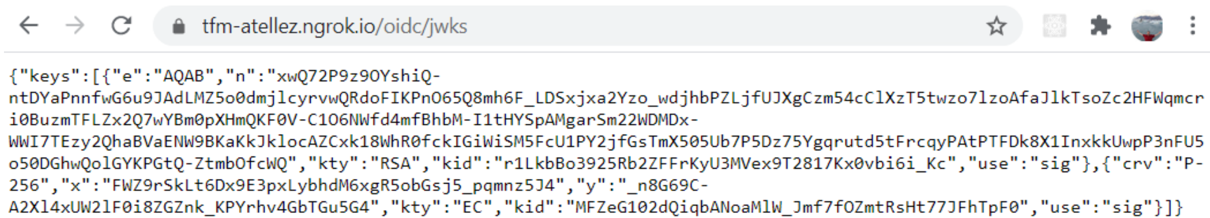


Figura 4-12: contenido de la URL que devuelve las claves públicas para validaciones en cliente

Estas claves son las que deben ser usadas por el cliente, kubernetes en este caso, para comprobar la autenticidad del token que se desee validar. Y la clave que se debe escoger dependerá del algoritmo con el que se haya generado la firma del token.

Para más información sobre la validación de un token JWT el lector puede acudir al anexo 0.

4.2.1.4. Parámetros de configuración de la aplicación

Desde el punto de vista del despliegue y arranque de la aplicación, es necesario configurar algunos datos, disponibles como variables de entorno, para su correcto funcionamiento. Estos datos se configuran por medio de objeto kubernetes *ConfigMap*. La configuración actual del clúster es:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: seg-config
  namespace: seguridad
data:
  REDIS_URL: "redis://rpi-master:30001"
  ISSUER: "https://tfm-atellez.ngrok.io/oidc"
  OIDC_PORT: "4400"
  NODE_ENV: "production"
  BASE_PATH: "/oidc"
```

Figura 4-13: configuración para el arranque correcto de la aplicación OIDC. Fichero 21-configmap-seguridad.yaml

La función de cada dato es:

- **REDIS_URL**: URL donde la instancia de Redis esté accesible. Esta URL nunca debería ser pública (desde fuera del clúster) ya que sólo debe usado por la aplicación OIDC. Aquí se almacenan los datos de los procesos de autenticación que se llevan a cabo y guarda cierta información durante el proceso. Debido a que no es posible, en la versión gratuita, montar un Redis compuesto por varias instancias que operen como 1 sola, solo hay 1 instancia en el clúster, pero esto no es algo crítico ya que, aunque la instancia se cayese, esto no afectaría a toda la gente que esté logada ya, y sólo afectaría a los nuevos procesos de autenticación, y solo mientras que la instancia estuviese caída (algo que puede ocurrir muy raramente). Por tanto, y aunque idealmente habría 2 instancias, tener solo 1 no debería ocasionar grandes problemas.
- **ISSUER**: esta será la URL que se devuelva como el emisor de los tokens generados (dentro del campo *iss* del propio token). Y en base a ella, el cliente usará las claves públicas que buscará en esa URL.
- **OIDC_PORT**: puerto interno donde la aplicación estará escuchando. Debe coincidir con el valor que se configure en el despliegue de la aplicación. Y en realidad, probablemente no sea necesario cambiar este valor en ningún caso.
- **NODE_ENV**: entorno de ejecución de la aplicación. Debe ser *production* en todo caso ya que, si no, habría ciertas funcionalidades relacionadas con seguridad que no se emplearían.
- **BASE_PATH**: la ruta, sin contar el dominio, donde la aplicación estará accesible. Esto se usa más a nivel interno, para asegurar que las redirecciones se hagan a las rutas que correspondan.

4.2.1.5. Definición y configuración de roles en la aplicación OIDC

Por otro lado, y tal como se había dicho, la aplicación OIDC se ha preparado para que en los casos de autenticación correcta se determine qué roles debe tener el usuario que se está logando en función de su email y de su ID de usuario y que estos roles se incluyen en el

token generado. La gestión de roles se puede configurar de forma sencilla por medio de otro objeto kubernetes *ConfigMap*, que tiene un aspecto como el de la Figura 4-14.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: map-grupos
  namespace: seguridad
data:
  dominios: |
    alumno.uned.es: alumno
  usuarios: |
    atellez9: cluster_admin
    local_u1: nstest1,nstest2
    local_u2: nstest1
```

Figura 4-14: configuración de grupos para dominios y usuarios. Fichero 22-configmap-map-grupos.yaml

Básicamente, se definen 2 ficheros (*dominios* y *usuarios*), y en cada uno de ellos se escribe en una línea un par `dato: grupo`, siendo el dato o bien el dominio del email, bien identificador del usuario (entendiéndolo como la parte anterior al símbolo '@' en el email), y siendo el grupo el rol que deberá estar definido en kubernetes (en caso de que no esté definido simplemente no tendrá efecto).

El funcionamiento es sencillo, tras una validación correcta de credenciales en el SSO de UNED, y antes de dar por terminado el proceso de autenticación, la aplicación OIDC toma el email del usuario que se acaba de autenticar y lo divide en 2 partes, usando la '@' como delimitador. A continuación, busca si hay alguna entrada definida en el fichero *dominios* para la parte del dominio del email y, de ser así, toma el listado de grupos (separados por coma) como roles a añadir al token. Y realiza el mismo proceso para la parte de identificación del usuario. Por tanto, al final el token contendrá la suma de roles definidos para el dominio y para el ID de usuario.

De esta forma, en un fichero de configuración externo a la aplicación de autenticación/autorización, se puede gestionar fácilmente quién tiene acceso a qué roles. Y luego los roles (qué pueden hacer y en qué *namespaces*) ya se pueden gestionar directamente en kubernetes, como veremos en el apartado 0.

4.2.1.6. Ejemplo y explicación de un resultado correcto de la autenticación

Como resultado del proceso de autenticación, en caso satisfactorio, se obtendrá un token (*id_token*) que contendrá información básica sobre el usuario logado, incluyendo los grupos o roles a los que el usuario pertenece. Pongamos un ejemplo para entender mejor qué información tenemos y cómo podemos sacarle rendimiento. Supongamos que yo me logo con mi usuario de UNED, como resultado se generará un token como el mostrado en la Figura 4-15:


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6InIxTGtiQm8zOTI1Um
IyWkZGckt5VTNNVmV4OVQyODE3S3gwdmJpNmlfS2MifQ.eyJzdWIiOiJhdGVsb
GV6OSIsImFjdGl2ZSI6dHJ1ZSwidXNlcnR5cGUiOiIyIiwiY2x1c3Rlcmdyb3V
wcyI6WyJhbHVtbm8iLCJjbHVzdGVyX2FkbWluIl0sImVtYWlsIjoiYXRlbGxle
jlAYWx1bW5vLnVuZWQuZXMiLCJub25jZSI6IjY4Mjc1NTAzMzgwMCIsImF1ZCI
6Imt1YmVybmV0ZXMiLCJleHAiOjE1OTY5MTM5NzQsImlhdCI6MTU5NjgyNzU3N
CwiaXNzIjoiaHR0cHM6Ly90Zm0tYXRlbGxlei5uZ3Jvay5pby9vaWRjIn0.jju
QeeouadqBa1jNrcwzzpUY8qcNhYz94Uo3nvipT3HJdjMmQbr8OoVR2BJKkVJTB
W1pXTwccaEvNLgg7nzknpJ0F-
4j0OEQ85pVrT9AZRy60U8cDYtmkqWEMWCaIMTpcNLhCCZ12pCach2YfjzbPSrY
tzrO0hT5sMJNAyeAF89qqnfZ3pZzBkrYr1cTmIQ_TDAQG4ckQOfI938v_d-
5aWF6lOyIs7WbUEQ4Hbai6CoGDD5UjaJOtrzszmuCwEgvdEazXGfMbmqXnhtuG
JmGVn7Dah7uH4Mr2ANvPMY1mrda8KyrGcDzHVwybqgHNMXs9ARVzctm4rlFQnP
hsv06Lg
```


Figura 4-15: ejemplo de id_token generado tras una autenticación correcta con la aplicación OIDC

Este token tiene formato JWT, formato definido por el RFC 7519 (IETF, 2020), se compone de 3 partes separadas por un punto. La primera parte es relativa a información sobre el token en sí, para su correcta validación. La tercera parte es una firma que sirve para asegurar la integridad del token. Obviamente son partes importantes, pero la parte que nos interesa a nosotros es la segunda. Cada parte está codificada en base64. Decodificando la segunda parte podemos ver información en formato JSON, tal como se muestra en la Figura 4-16:


```
{
  "sub": "atellez9",
  "active": true,
  "usertype": "2",
  "clustergroups": [
```

```
    "alumno",
    "cluster_admin"
  ],
  "email": "atellez9@alumno.uned.es",
  "nonce": "682755033800",
  "aud": "kubernetes",
  "exp": 1596913974,
  "iat": 1596827574,
  "iss": "https://tfm-atellez.ngrok.io/oidc"
}
```

Figura 4-16: Información contenida en el id_token relativa al usuario que se ha autenticado

De entre todos los datos que componen esta parte[6], los que más nos interesan son:

- **sub**: el ID del usuario logado. El ID de UNED.
- **active**: campo homónimo, aunque en inglés, que proviene del servicio de autenticación de UNED.
- **usertype**: campo homónimo, aunque en inglés, que proviene del servicio de autenticación de UNED.
- **clustergroups**: lista de roles o grupos que la aplicación de OIDC ha asociado al usuario. Esta lista se genera a partir del dominio del email, y a partir del ID de usuario. En este caso, al usuario se le han otorgado los roles *alumno* (debido al dominio del email) y *cluster_admin* (debido al ID del usuario).
- **email**: email del usuario logado en la UNED.

En el capítulo 5 se pondrán más ejemplos de tokens generados para diferentes usuarios, así que de momento nos podemos quedar solamente con este caso.

[6] para más información sobre el resto de los campos que no han sido mencionados o sobre cómo se valida un token JWT el lector puede acudir al anexo 0.

En cualquier caso, este token se debe poner en la configuración del clúster para que kubernetes pueda usarlo en cada una de sus peticiones. Para más información sobre cómo llevar a cabo esta configuración, se puede acudir al anexo C.

4.2.1.7. *Funcionalidad de gestión de usuarios locales o de aplicación*

Otra funcionalidad que se ha añadido en la aplicación es la posibilidad de definir usuarios locales para la aplicación que no llegan a validarse en el SSO de UNED. Esos usuarios se definen (email y contraseña) y se consideran usuarios válidos. Esta funcionalidad no debería ser usada en un clúster productivo, ya que estos usuarios no son personales. Sin embargo, para el desarrollo de este trabajo resultaba útil disponer de varios usuarios diferentes, a los que poder asignar diferentes roles, y la mejor forma de realizar esto ha sido por medio de esta funcionalidad. Y debido a que quizás pudiera haber algún caso en el que esto pudiera resultar útil (quizás en algún momento algún usuario de aplicación deba realizar acciones sobre el clúster, por ejemplo) se ha dejado la funcionalidad activa. No obstante, insistimos, esto no debería usarse en un clúster productivo a no ser que se sepa exactamente qué se está haciendo y posibles riesgos que conlleva.

En cualquier caso, esta configuración de usuarios locales se lleva a cabo por medio de otro objeto kubernetes ConfigMap, tal como se puede ver en la Figura 4-17:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: map-usuarios-locales
  namespace: seguridad
data:
  usuarios-locales: |
    local_u1@alumno.uned.es: password
    local_u2@alumno.uned.es: password
    local_u3@alumno.uned.es: password
```

Figura 4-17: configuración de usuarios locales. Fichero 23-configmap-map-usuarios-locales.yaml

Tal como se puede ver en la Figura 4-17, los usuarios locales se definen dentro del fichero usuarios-locales, como un listado de usuarios por medio del email y la contraseña de cada uno de ellos.

La contraseña de los usuarios está en claro, algo que no es nada recomendable en absoluto, y probablemente fuese una mejora recomendable el encriptar las contraseñas para que no estuvieran en claro, en caso de que esta funcionalidad se quisiese usar en producción. En cualquier caso, la aplicación está desplegada en el namespace *seguridad*, que debe estar restringido en todo momento incluso a lectura a sólo aquellos usuarios que tengan permisos de administrador.

4.2.1.8. Auto escalado horizontal de la aplicación

Una funcionalidad muy útil que ofrece kubernetes es la de auto escalado horizontal, que permite crear nuevos pods si el uso que se le está dando a una aplicación sobrepasa los parámetros definidos. Esto es muy útil porque permite tener en horas valle un número menor de replicas (pods) y si el uso de una aplicación se dispara, automáticamente y sin ninguna supervisión, kubernetes crea nuevas réplicas que pueden atender la nueva demanda. De esta forma, los recursos que no son necesarios en un momento dado quedan libres y pueden ser usados para otros usos en caso de ser necesario.

Para esta aplicación, tal como se ha visto en el apartado anterior, se ha definido que haya 2 réplicas creadas. Y seguramente para esta aplicación sea suficiente estas 2 réplicas para toda la demanda que pudiera haber. No obstante, y para mayor seguridad y tranquilidad, se ha definido un auto escalado para esta aplicación que puede incrementar hasta las 4 réplicas en caso de ser necesario. Concretamente, la configuración del auto escalado se hace tal como se muestra en la Figura 4-18:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: oidc
  namespace: seguridad
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: oidc
  minReplicas: 2
  maxReplicas: 4
```

```yaml
  metrics:
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Figura 4-18: configuración de auto escalado para la aplicación OIDC. Fichero 31-uned-oidc-hpa.yaml

Como se puede ver en la Figura 4-18, se ha definido que se empiece a escalar cuando el uso medio de memoria asignada entre todas las réplicas sea del 80% o superior, y en caso de que se dé esa situación se creará una 3ª réplica. Y una vez se haya creado, si se vuelve a dar la situación de un uso superior al 80% se creará una 4ª réplica.

El hecho de haber configurado la memoria al 80% es debido a que hemos comprobado que una réplica nueva se inicia en unos 15 segundos y el uso de memoria aumenta lentamente. Además, el uso que hemos ido siguiendo durante la realización de este trabajo ha sido de unos 30Mi y hemos fijado un límite de 128Mi, por lo que en caso de que se llegase a un uso del 80% sería de esperar que para llegar al 100% habría un tiempo superior a los 15 segundos que tardaría la nueva réplica en estar operativa. En cualquier caso, sería recomendable hacer pruebas de carga en el clúster definitivo con los perfiles de carga esperables y ver cómo evoluciona el uso de recursos, y así poder ajustar mejor estos parámetros del auto escalado en caso de que fuese necesario.

Y en caso de que haya más réplicas que las definidas como el número mínimo, 2 en nuestro caso, si después de 5 minutos hay un uso de memoria inferior al 80% (contando que se eliminaría una réplica), una réplica se eliminaría. Y así sucesivamente hasta llegar al número mínimo de réplicas.

El auto escalado también se puede definir en función del uso de CPU que se haga, pero la aplicación OIDC apenas hace uso de CPU por lo que nunca llegará a ser algo crítico que debamos considerar.

Sólo por aclarar que se ha definido como máximo 4 réplicas porque en el despliegue habíamos definido que las réplicas deben ir a diferentes nodos (anti-afinidad), y tenemos 4 nodos en total, por lo que nunca podría crearse una quinta réplica para una aplicación con

este tipo de anti-afinidad. Y, en cualquier caso, si se llegase a una situación en la que hiciesen falta más de 4 réplicas para esta aplicación, que sólo valida los usuarios nuevos y valida los tokens cuando se realiza alguna acción en el clúster, probablemente querría decir que el clúster está sobredimensionado.

4.2.1.9. Implementación de la aplicación

La aplicación se ha desarrollado en el lenguaje JavaScript para su funcionamiento en NodeJS. Como base para el código hemos cogido la aplicación ya existente node-oidc-provider, que está publicada a modo de ejemplo de implementación de OIDC para NodeJS (node-oidc-provider, 2020).

Sobre dicha aplicación se han desarrollado varios cambios, siendo los principales cambios hechos sobre dicha base:

- Implementación de la autenticación usando el SSO de la UNED.
- Gestión de grupos/roles en función de la dirección email del usuario autenticado.
- Funcionalidad para permitir la autenticación de usuarios locales (o usuarios de aplicación).
- Adaptar la aplicación para poder funcionar sobre una ruta que no sea la raíz. Esto es necesario si todo el clúster se expone sobre un mismo dominio y las aplicaciones se deben organizar en rutas, sin posibilidad de definir un subdominio particular.
- Configuración de un cliente particular para kubernetes.
- Cambio de claves por defecto. Las claves incluidas en la aplicación base no son únicas para nuestra aplicación, por lo que hemos configurado otras propias. Estas claves se han dejado como parte del código en sí en lugar de externalizarlas a un *secret*, por ejemplo, por sencillez en el manejo de la aplicación y teniendo en cuenta que tanto la imagen generada como el código fuente están restringidos y sólo quien está autorizado a ello puede tener acceso de alguna forma a dichas claves.

Una vez que la aplicación estaba lista se ha optado por su despliegue dentro del propio clúster en lugar de externalizarlo a otro sitio. Esta decisión está motivada principalmente por dos motivos:

1. Acotar el ámbito de validez de la aplicación al clúster en cuanto a configuraciones necesarias, definición de clientes, etc. Al tener un ámbito acotado, también tenemos controlados los posibles riesgos debidos a un posible mal funcionamiento.
2. El clúster es completamente autocontenido. Esto hace que no sea necesario preparar o configurar ninguna aplicación o sistema adicional, más allá del posible dominio y el balanceador de carga.

Esta aplicación, tal como hemos visto en el apartado 4.2.1.1, se despliega dentro del clúster como cualquier otra aplicación, encargándose kubernetes de incluirla en los nodos que corresponda.

Si fuese necesario la disponibilidad de esta aplicación en un ámbito más amplio o su acceso público desde internet sería muy recomendable llevar a cabo un exhaustivo análisis de seguridad (hacking ético) así como extraer las claves usadas fuera de la aplicación. Esta implementación está aconsejada únicamente para uso en estos tipos de clústeres y teniendo un despliegue independiente para cada clúster.

4.2.1.10. Otras notas relacionadas con la aplicación

Como nota aclaratoria, si se hace cualquier modificación en cualquiera de los ficheros *ConfigMap* mencionados, es necesario reiniciar todos los pods que haya relacionados con la aplicación para que tome los cambios, y si el cambio no es algo crítico, para no impactar al uso productivo de la aplicación, se deberían reiniciar los pods uno a uno para asegurar que siempre haya alguna instancia levantada y funcionando.

Por último, hay que añadir que esta aplicación se ha desarrollado particularmente para este trabajo fin de máster y que está almacenada como imagen Docker en el *hub* de Docker,

tal como se detalla en el anexo 0, y dentro de un repositorio privado, y por tanto únicamente aquellas personas autorizadas pueden descargarse la imagen.

En el anexo 0 se detalla el proceso de generación de una imagen Docker apta para su uso en Raspberry desde un dispositivo con una arquitectura diferente a ARM, así como lo necesario para poder instalar esta imagen Docker en el clúster.

4.2.2. Cliente OIDC para usuarios

4.2.2.1. Descripción funcional

El proceso de autenticación requiere que el token resultado se incluya en el fichero de configuración, en el sitio que le corresponde (ver anexo C). Además, para iniciar el proceso es necesario enviar una serie de parámetros que deben contener unos contenidos bastante definidos (ver apartado 4.2.1.2).

Si bien ambas acciones no son muy complicadas de hacer, es cierto que no es lo más cómodo para operar con ello y, además, podría ser confuso de usar cuando no se está acostumbrado a ello.

Por estos motivos es por lo que se ha creado una aplicación independiente que se encarga de envolver todo el proceso de uso de la aplicación OIDC para que el usuario sólo tenga que arrancar esta aplicación e introducir su usuario y contraseña cuando así se solicite, nada más. De esta manera, el proceso se vuelve muy sencillo e intuitivo.

La aplicación está disponible para Windows, Linux y MacOS como un ejecutable. Por ejemplo, en Windows sólo tenemos que ejecutar el archivo binario `uned-oidc-client-win.exe` como cualquier otro archivo ejecutable. Al hacerlo, se abrirá también una ventana o pestaña de navegador donde se cargará directamente la aplicación OIDC y que debería mostrarnos directamente la ventana donde introducir el usuario y contraseña (ver Figura 4-9). Una vez introducidas las credenciales, si son correctas, el navegador habrá redirigido a la URL `https://127.0.0.1:8123/` con el marcador *id_token*. Esta URL se corresponde con la máquina local donde se haya ejecutado el programa, y el puerto donde está escuchando la

aplicación ejecutada para recibir la respuesta. Este servidor, al recibir esta petición comprueba que se trata de una petición correcta y actualiza, usando la herramienta de kubernetes de línea de comandos *kubectl*, la configuración local de kubernetes con este token.

En el navegador se habrá quedado una ventana que indica:

Login finalizado correctamente.

Ya puedes cerrar esta pestaña y operar con **kubectl** desde la terminal.

Figura 4-19: confirmación del resultado correcto del proceso de autenticación

Una vez recibido este mensaje ya se puede cerrar esa ventana/pestaña, y se podrá operar con el clúster por medio de *kubectl* de forma usual. Por otro lado, el programa habrá terminado su ejecución y se habrá cerrado.

En caso de que las credenciales introducidas no sean correctas, también se redirigirá a la misma URL, pero sin el token y con la información del error, que se mostrará en pantalla:

access_denied

Invalid username and/or password, or the user is no more active

Figura 4-20: información sobre el resultado incorrecto del proceso de autenticación

El único requisito es el que el usuario tenga disponible la aplicación *kubectl*, que de igual modo debe tener si desea operar con el clúster, objeto último de la aplicación OIDC, por lo que este requisito no debería ser un obstáculo para su uso.

Este proceso es el mismo para cualquier usuario del clúster, bien sea un administrador, un miembro del equipo docente o un estudiante. Por tanto, lo único que es necesario es distribuir el archivo ejecutable del sistema operativo que corresponda para que cualquier usuario pueda iniciar el proceso de autenticación. Poniendo un posible ejemplo de uso, el proyecto eNMoLabs podría incluir estos archivos ejecutables dentro de su página web oficial,

con las instrucciones de uso aquí mencionadas, y redirigir a cualquier usuario potencial a dicha página donde estos usuarios podrán descargarse el ejecutable e iniciar el proceso.

No obstante, antes de realizar la distribución del cliente será necesario que el administrador del clúster configure la URL de acceso a la API de kubernetes en Raspberry máster, y el certificado para el acceso con conexión segura al mismo. Ambos datos hay que configurarlos en el fichero `.env` de la raíz de la aplicación (Figura 4-21), y volver a empaquetar los ejecutables posteriormente.

```
API_URL=https://rpi-master:6443
CERT_AUTH_DATA=LS0tLS1CRUdJTiBDRVJ...
```

Figura 4-21: parámetros a configurar en la aplicación cliente OIDC antes de su distribución

Por último, hay que decir que esta aplicación da por hecho que se está trabajando con el clúster desde el dispositivo en el que se ejecutar este cliente. Se puede acudir al anexo C para entender mejor cómo configurar el acceso remoto al clúster en sí para su operativa.

4.2.2.2. Detalle de la implementación

En el arranque de esta aplicación se ejecutan 2 acciones independientes entre sí:

- Apertura en el navegador de la URL de entrada de la aplicación OIDC
- Preparación y arranque del servidor que recibirá la respuesta, sea la que sea, del proceso de autenticación

Mientras que la parte del navegador se lanza y se da por terminada, la parte del servidor se queda a la escucha de que llegue una respuesta. En caso de que no llegue una respuesta en 3 minutos, la aplicación da el proceso por cancelado y se cierra.

En caso de recibir una respuesta correcta desde el proceso de autenticación, se lanzan 2 comandos *kubectl* para actualizar tanto el token como la fijar el usuario actual como usuario para operar en el clúster. La primera acción sirve para refrescar el token en la configuración, y la segunda sirve para indicar que el usuario que se acaba de logar es el que debe operar.

Esto último solo tiene realmente sentido cuando en el mismo ordenador se trabaje en kubernetes con más de un usuario.

Así, el diagrama de secuencia de esta aplicación queda así:

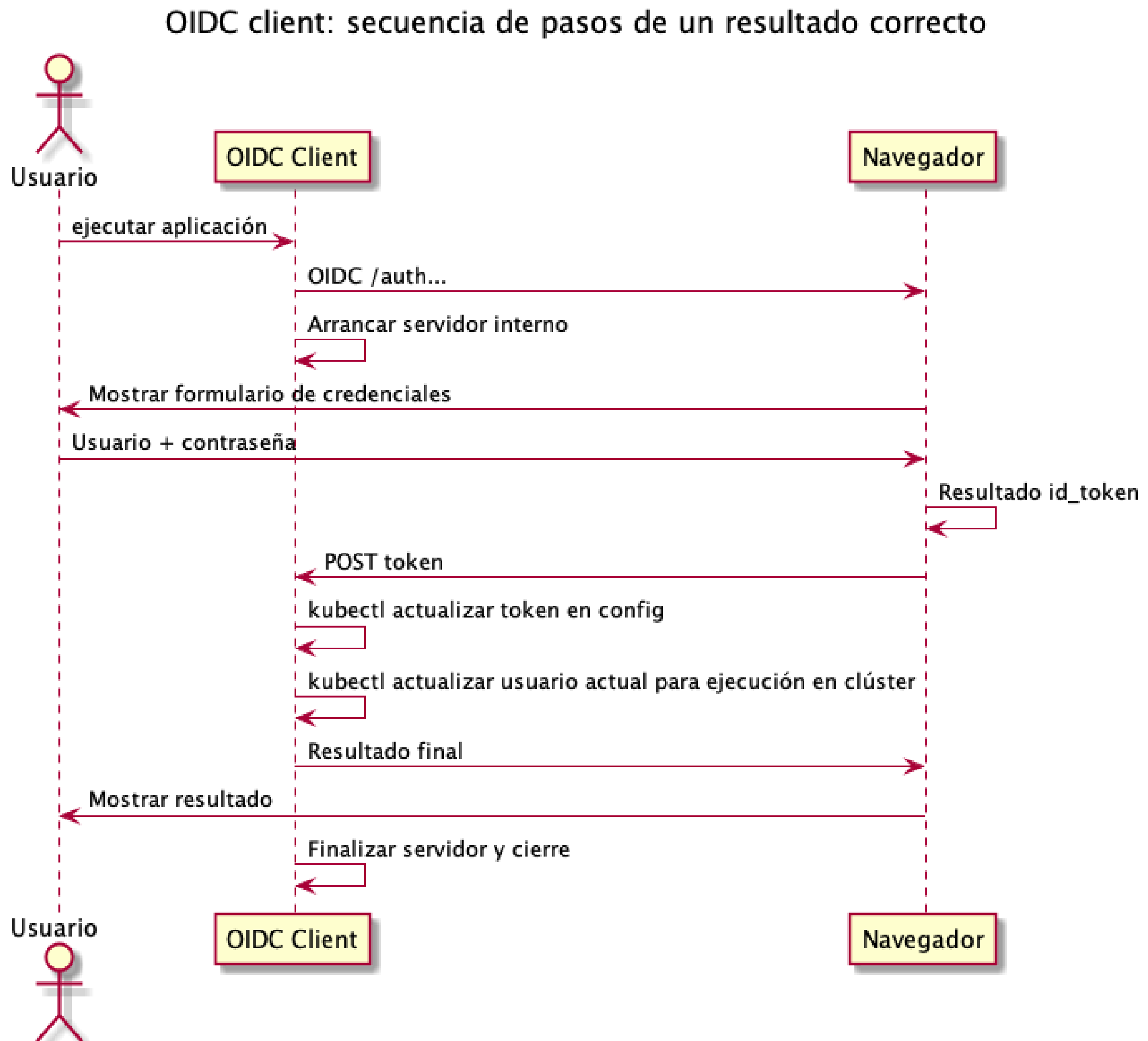


Figura 4-22: diagrama de secuencia de la aplicación cliente

4.3. Monitorización

Para poder gestionar mejor el clúster es necesario establecer un sistema de monitorización que permita poder consultar el estado del clúster en sí en el momento que consideremos. Además, podemos configurar unas alertas que se disparen automáticamente en el momento en que suceda alguna de las situaciones que configuremos, y que estas alertas nos lleguen, por ejemplo, por correo electrónico (que normalmente suele estar configurado en el teléfono móvil que solemos llevar encima) y así poder conocer las situaciones potencialmente anómalas en el momento en el que sucedan, sin necesidad de esperar a revisar manualmente el estado del clúster.

Para tenerlo agrupado y mejor controlado, todo lo relativo a la monitorización se ha desplegado en un *namespace* dedicado a ello: *monitoring*. Se debe asegurar que ninguna aplicación, más allá de las dedicadas a la monitorización, se pueda desplegar en este *namespace* ya que se disponen de permisos especiales dentro del mismo y una aplicación intrusa podría hacer un uso no deseado de dichos permisos.

Tal como iremos viendo en los subapartados de este capítulo, basaremos nuestra monitorización en tres piezas fundamentales: Prometheus, Alert manager y Grafana. La estructura de cómo quedan integradas las piezas se puede ver en la Figura 4-23, donde una pieza se apoya en la que tiene inmediatamente por debajo.

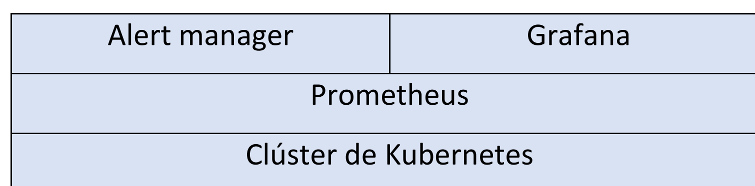


Figura 4-23: estructura sobre cómo se asientan las piezas de monitorización dentro del clúster

4.3.1. Prometheus

Para la monitorización hemos optado por Prometheus (Prometheus, 2020), al tratarse de una herramienta que se integra muy bien con Kubernetes, y que permite operar sobre multitud de métricas. Las métricas no vienen en su mayoría de Prometheus en sí, sino que éste va recopilando información de todos los sitios disponibles que estén configurados para ello, siendo el más interesante para nuestro propósito el sistema que provee información

sobre el clúster de Kubernetes. Pero se pueden configurar más fuentes de información, ampliando las métricas disponibles. Esto se puede hacer programando manualmente algún proceso que genere y exporte métricas o se pueden encontrar multitud de *plugins* que recaban más información y la exponen para que Prometheus la pueda gestionar.

Es decir, con Prometheus lo que conseguimos es tener en un único sitio centralizado toda la información que se pueda recabar, en base a todas las fuentes de información disponibles.

Como se puede ver en la Figura 4-23, Prometheus se apoya sobre el clúster de kubernetes, del que extrae métricas continuamente. Y sirve a Alert manager y a Grafana.

Las fuentes de métricas no están dentro de Prometheus, sino que cada aplicación/sistema puede proveer las métricas y disponerlas para que Prometheus las lea y las procese.

Y, como se ha dicho, la información no tiene por qué ser únicamente relativa al clúster sino también se podría extender a las diferentes aplicaciones que se vayan desplegando en el clúster si estas se configurar para exportar la información en un formato compatible con Prometheus.

Su funcionamiento, de forma resumida, consiste en ver qué fuentes hay disponible en cada uno de los *namespaces*, y conectarse a cada una de las fuentes de información encontradas. Para detectar qué fuentes hay disponibles analiza qué objetos hay desplegados y sobre los que son compatibles y están configurados para ser analizados, intenta conectarse para extraer la información. Y la detección de objetos se realiza a través de unos trabajos programados según una configuración definida en un `ConfigMap` que posteriormente es consumido por la aplicación.

Puesto que necesita analizar los objetos disponibles, es necesario otorgar ciertos permisos a Prometheus ya que de otro modo Kubernetes no le permitiría realizar el escaneo

que le permite descubrir las fuentes. Para ello debemos crear un `ClusterRole`[7] en Kubernetes y vincularlo. Esto lo hacemos tal como se muestra en la Figura 4-24:

```yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring
```

Figura 4-24: configuración del objeto ClusterRole dentro del clúster para otorgar permisos al usuario asociado a Prometheus. Fichero 10-prom-clusterrole.yaml

Como podemos ver en la Figura 4-24, estamos otorgando permisos de lectura (`get`, `list` y `watch`) sobre una serie de recursos Kubernetes, así como sobre una URL, `/metrics`, que es la estándar en la que Prometheus se conecta para extraer la información.

[7] Un objeto de tipo ClusterRole se corresponde con un conjunto de permisos sobre todo el clúster. Por tanto, debemos tener cuidado al crearlos y, sobre todo, al asignarlo a grupos/usuarios, ya que una gestión no cuidadosa podría resultar en más permisos de los deseados para algunos usuarios.

La configuración que mencionábamos relativa a la aplicación Prometheus se realiza por medio del fichero que se ve en la Figura 4-25:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-conf
  labels:
    name: prometheus-server-conf
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
(las alertas se definen aquí)
…
  prometheus.yml: |-
    global:
      scrape_interval: 5s
      evaluation_interval: 5s
    rule_files:
      - /etc/prometheus/prometheus.rules
    alerting:
      alertmanagers:
      - scheme: http
        static_configs:
        - targets:
          - "alertmanager.monitoring.svc:9093"

    scrape_configs:
…
      - job_name: 'prometheus'
        static_configs:
          - targets: ['tfm-atellez.ngrok.io']
…
      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.kube-system.svc.cluster.local:8080']
…
        metrics_path: '/prometheus/metrics'
        scheme: https
…
```

Figura 4-25: esquema de la configuración para Prometheus. Fichero 11-prom-config.yaml

Debido a que el fichero mostrado en la Figura 4-25 es muy extenso, hemos copiado solo una parte de éste.

En cualquier caso, en el `ConfigMap` podemos ver que se definen 2 ficheros:

- **prometheus.rules**. Dentro de este fichero definimos las alertas que queremos usar posteriormente. Hablaremos más en detalle de esto en el punto 4.3.2.
- **prometheus.yml**. Aquí indicamos la configuración que queremos que tenga Prometheus con relación a la extracción de métricas, así como dónde enviar la información de las alertas que se disparen. También se configuran los trabajos encargados de recopilar la información. Hemos dejado únicamente un trabajo en la anterior, Figura 4-25, que se encarga de recopilar información sobre la propia instancia de Prometheus, accediendo vía su dominio externo.

El despliegue y exposición de la aplicación es bastante sencillo, salvo por un par de detalles que hay que tener en cuenta:

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
  labels:
    app: prometheus-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-server
  template:
    metadata:
      labels:
        app: prometheus-server
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
            - "--storage.tsdb.path=/prometheus/"
            - "--web.external-url=https://tfm-atellez.ngrok.io/prometheus"
          ports:
            - containerPort: 9090
          volumeMounts:
```

```yaml
          - name: prometheus-config-volume
            mountPath: /etc/prometheus/
          - name: prometheus-storage-volume
            mountPath: /prometheus/
        livenessProbe:
          httpGet:
            path: /-/healthy
            port: http
            scheme: HTTP
          initialDelaySeconds: 60
          timeoutSeconds: 3
        readinessProbe:
          httpGet:
            path: /-/ready
            port: http
          initialDelaySeconds: 30
          timeoutSeconds: 3
      volumes:
        - name: prometheus-config-volume
          configMap:
            defaultMode: 420
            name: prometheus-server-conf

        - name: prometheus-storage-volume
          emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus-service
  namespace: monitoring
  annotations:
    prometheus.io/scrape: 'true'
    prometheus.io/port:   '9090'
spec:
  ports:
  - port: 9090
    protocol: TCP
    targetPort: 9090
  selector:
    app: prometheus-server
```

Figura 4-26: despliegue y exposición de Prometheus. Fichero 12-prometheus.yaml.

A destacar que hay que ejecutar el contenedor con ciertos argumentos, donde le indicamos cuál es el fichero de configuración -que se corresponde con uno de los ficheros anteriormente comentados almacenados en el `ConfigMap`-, ruta donde almacenar los datos y, en este caso, cuál es la URL externa en la cual estará expuesto Prometheus. Los primeros

argumentos son más bien estándar, pero el último es algo particular hecho para esta configuración de clúster, y debido a ello vamos a detenernos un poco más en el él ya que conviene entender por qué es necesario en este caso y por qué puede que no lo sea en otros casos. En esta configuración de clúster hemos decidido exponer Prometheus con una ruta particular dentro de nuestro *ingress*. Esto hace que cuando accedemos a Prometheus, este no se encuentra en la raíz del dominio por el que accedemos sino dentro de una ruta particular (`/prometheus`). Pues bien, para que la aplicación, en el lado del navegador, funcione correctamente es necesario configurar la ruta por la que se accede a Prometheus, ya que de no hacerlo se producirán errores con todos los ficheros estáticos (scripts, imágenes, etc.) y no podrán ser accesibles.

Prometheus dispone de este argumento, `web.external-url`, para solucionar esta situación, aunque es la única posibilidad que tenemos: si es posible se podría crear un subdominio particular para Prometheus donde se accedería en la raíz, o se podría recubrir el acceso con un servidor web -Nginx por ejemplo- que se encargue de reescribir la ruta de los ficheros estáticos antes de devolver los ficheros al navegador. Si tuviésemos la necesidad de exponer Prometheus bajo 2 o más rutas distintas seguramente debamos plantearnos hacer uso de alguna de estas soluciones. Sin embargo, en nuestro caso, por simplicidad hemos optado directamente por este parámetro para solventar el problema.

Hay que mencionar también que fijamos únicamente 1 réplica para la aplicación debido a que el clúster que tenemos no es especialmente grande, y queremos dejar suficiente espacio para el resto de las aplicaciones que se vayan a desplegar. Y tenemos en cuenta que, si bien es muy importante Prometheus, no es una aplicación crítica. Pero idealmente tendríamos 2 réplicas. De hecho, y como se comenta en el punto 6.2, si hubiese una capacidad suficiente en el clúster se podría destinar un nodo del clúster únicamente a los *namespaces* de seguridad y al de monitorización. Aclarar que si el uso que se le vaya a dar al clúster es algo más crítico (como aplicaciones que deben estar en funcionamiento constantemente y/o realizando tareas críticas), seguramente sí sería indispensable incrementar el número de réplicas.

Respecto al servicio, la exposición de Prometheus, decir que hemos incluido 2 anotaciones algo particulares: `prometheus.io/scrape` y `prometheus.io/port`. Sirven para indicar a Prometheus que sí debe extraer métricas de este servicio, y por tanto de la aplicación, y le indica el puerto sobre el que debe hacerlo. Para que Prometheus, cuando encuentra un servicio disponible, intente extraer información del mismo se debe configurar la anotación indicada haciéndoselo saber y que por tanto pueda intentar el acceso a sus métricas.

Por último, exponemos el servicio hacia el exterior a través de un objeto de tipo `Ingress`. Este tipo de objetos sirven para *gestionar el acceso externo a los servicios en un clúster* (Kubernetes, 2020). Por tanto, lo que queremos hacer es que el servicio con el que podemos acceder a la aplicación Prometheus esté disponible desde fuera del clúster, y que así pueda ser consumida sin necesidad de estar dentro del clúster para ello. Esto lo hacemos con la configuración que se ve en la Figura 4-27:

```yaml
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: monitoring
  name: prometheus-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
spec:
  rules:
  - http:
      paths:
      - path: /prometheus
        backend:
          serviceName: prometheus-service
          servicePort: 9090
```

Figura 4-27: exposición al exterior de Prometheus. Fichero 13-prom-exponer.yaml

Como habíamos mencionado anteriormente, y tal como vemos en la Figura 4-27, lo exponemos al exterior bajo la ruta `/prometheus`.

Como fuente adicional de métricas hemos desplegado una instancia de *kube-state-metrics* y otra de *node-exporter*. *Kube state metrics* es un servicio que está pendiente de las peticiones a la API de Kubernetes para obtener información sobre el estado de los objetos dentro del clúster, sin modificar ningún objeto. Y *Node exporter* se encarga de recopilar y

exponer multitud de datos sobre cada uno de los nodos. Dentro de la configuración de Prometheus hay un *job* para cada uno que se encarga de extraer las métricas de aquí. No vamos a detallar su despliegue y configuración al no tratarse de nada realmente particular para este clúster, y por no aportar una mayor visión sobre lo realmente hecho. En cualquier caso, los ficheros para su despliegue y configuración están incluidos en el mismo repositorio de ficheros, junto con el resto de ficheros YAML usados.

Cuando accedemos a la URL donde Prometheus está accesible, veremos una interfaz gráfica (Figura 4-29) que permite hacer consultas sobre las métricas disponibles.

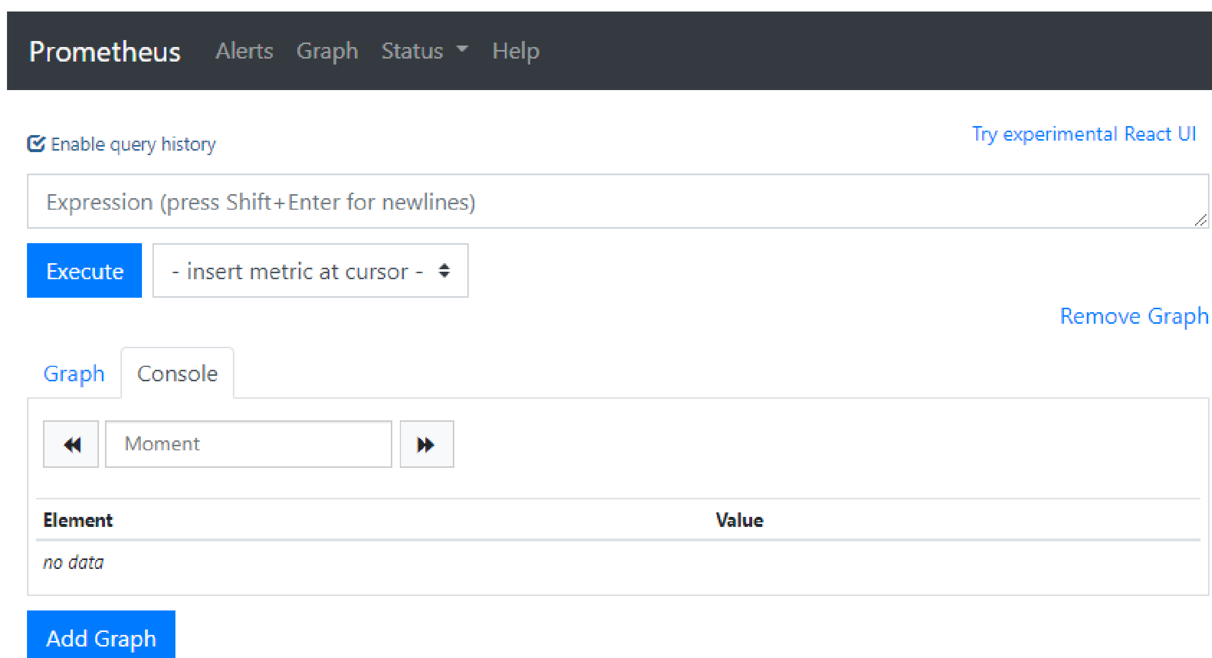


Figura 4-28: pantalla inicial cuando se accede a Prometheus, con la interfaz gráfica que permite realizar consultas sobre las métricas disponibles

Simplemente a modo de ejemplo de esta interfaz gráfica, vamos a obtener el porcentaje de memoria en uso por el clúster en el momento actual, mostrando los resultados tanto en forma de gráfico de evolución del valor (Figura 4-29) como en dato absoluto (Figura 4-30). Como se ve, el resultado que nos da es de un 46% de utilización de memoria, y teniendo en cuenta que nuestro clúster tiene 5GB (ver Tabla 4-5), esto quiere decir que estamos usando aproximadamente 2,3GB.

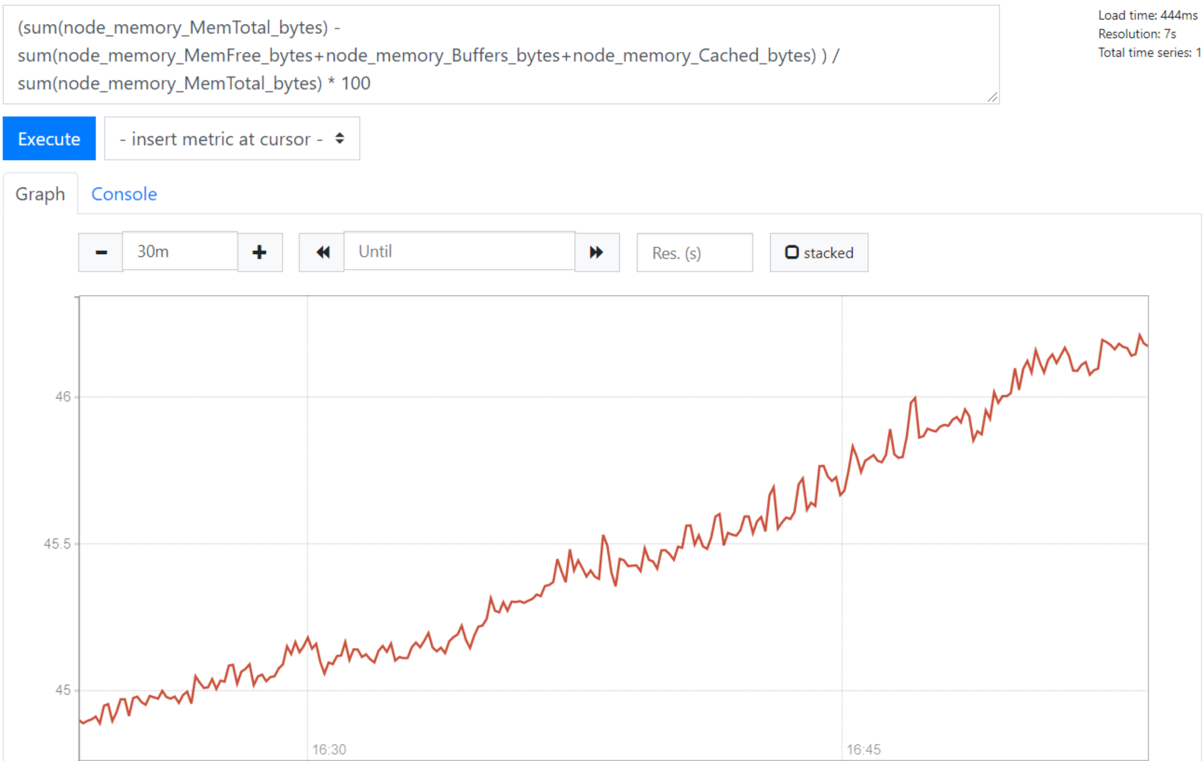


Figura 4-29: consulta en Prometheus del porcentaje de memoria en uso, con gráfica de resultado

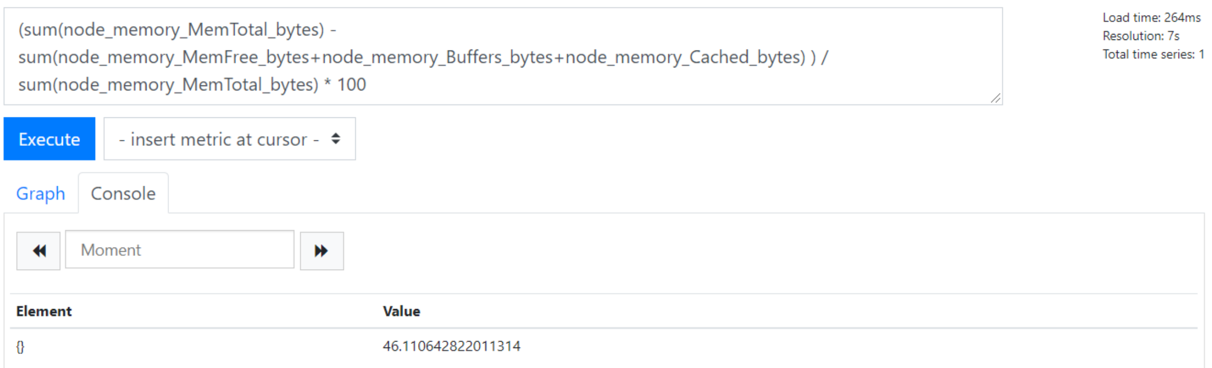


Figura 4-30: consulta en Prometheus del porcentaje de memoria en uso, con tabla de resultado

En el ejemplo de consulta anterior hemos buscado por un porcentaje, lo que nos da un único valor. Pero también podemos generar consultas que nos devuelvan un listado de valores, tal como puede ser la consulta para recuperar el uso de memoria en cada uno de los nodos del clúster. Estas consultas también se pueden recuperar en forma de gráfica de evolución (Figura 4-31) como en forma de tabla de resultados (Figura 4-32).

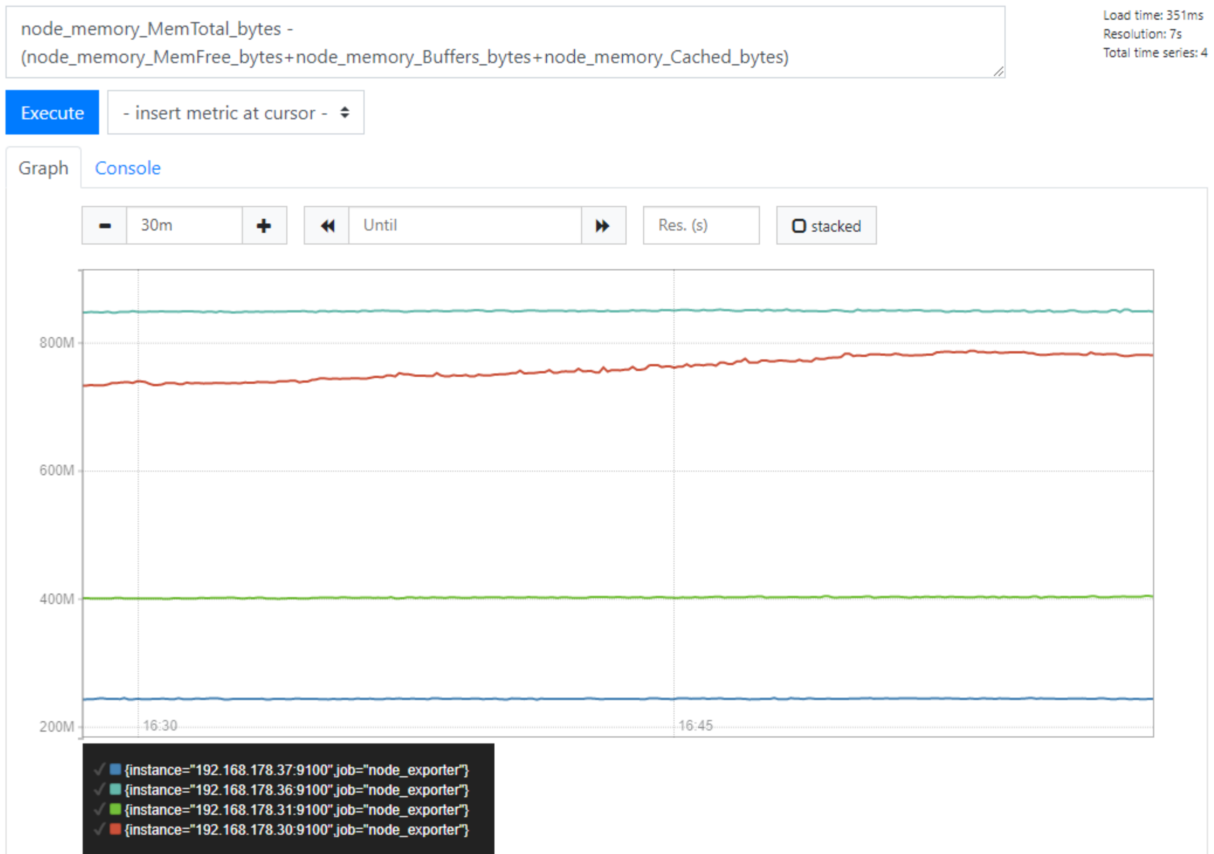
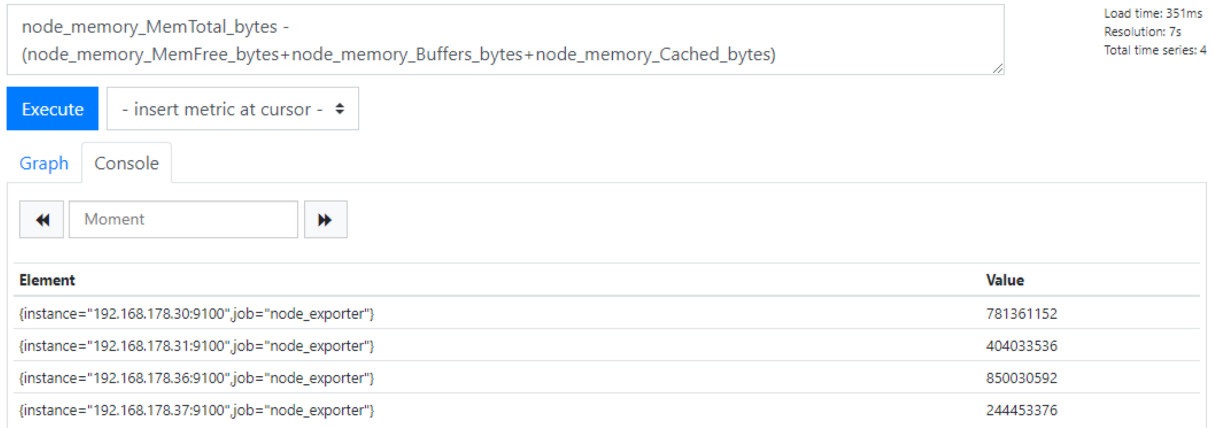


Figura 4-31: consulta en Prometheus sobre el uso de memoria en cada nodo, con gráfico de resultado



Element	Value
{instance="192.168.178.30:9100",job="node_exporter"}	781361152
{instance="192.168.178.31:9100",job="node_exporter"}	404033536
{instance="192.168.178.36:9100",job="node_exporter"}	850030592
{instance="192.168.178.37:9100",job="node_exporter"}	244453376

Figura 4-32: consulta en Prometheus sobre el uso de memoria en cada nodo, con tabla de resultado

En la última consulta realizada, se puede ver la memoria actual usada por cada nodo (Figura 4-32) en bytes, que si la sumamos nos da un valor aproximado de 2,27GB, prácticamente lo mismo que nos daba la consulta del tamaño de memoria usado a nivel de clúster. Hay que tener en cuenta que los valores dependen del momento en el que se realizan las consultas, por lo que pueden variar si dejamos pasar un pequeño intervalo de tiempo, como ha sido el caso.

Las consultas no se pueden guardar, por lo que no resulta muy cómodo tener que guardarlas en algún fichero y tener que ir copiando y pegando. Sin embargo, se pueden guardar de otra forma, y es que cada consulta genera una URL y por tanto podemos guardar las URLs generadas de las consultas que nos interesan. Por ejemplo, en la última consulta realizada, la URL generada es:

```
https://{dominio}/prometheus/graph?g0.range_input=30m&g0.expr=node_memory_MemTotal_bytes%20-%20(node_memory_MemFree_bytes%2Bnode_memory_Buffers_bytes%2Bnode_memory_Cached_bytes)&g0.tab=1
```

De esta forma, podemos guardar estas URLs (por ejemplo, en los favoritos del navegador) y cuando queramos ejecutar dicha consulta, simplemente ir al enlace guardado. En cualquier caso, esta interfaz no suele ser la ideal para visualizar el estado general del clúster, pero sí es útil para realizar consultas específicas.

Otro de los puntos de menú de la aplicación nos lleva al listado de aletas que tenemos configuradas, donde además podemos ver su estado actual de forma muy sencilla: si está en rojo es que la alerta está disparada actualmente, y si está en verde es que los valores de las métricas asociadas a esa alerta están bajo los umbrales definidos. Podemos ver un ejemplo en la Figura 4-33:

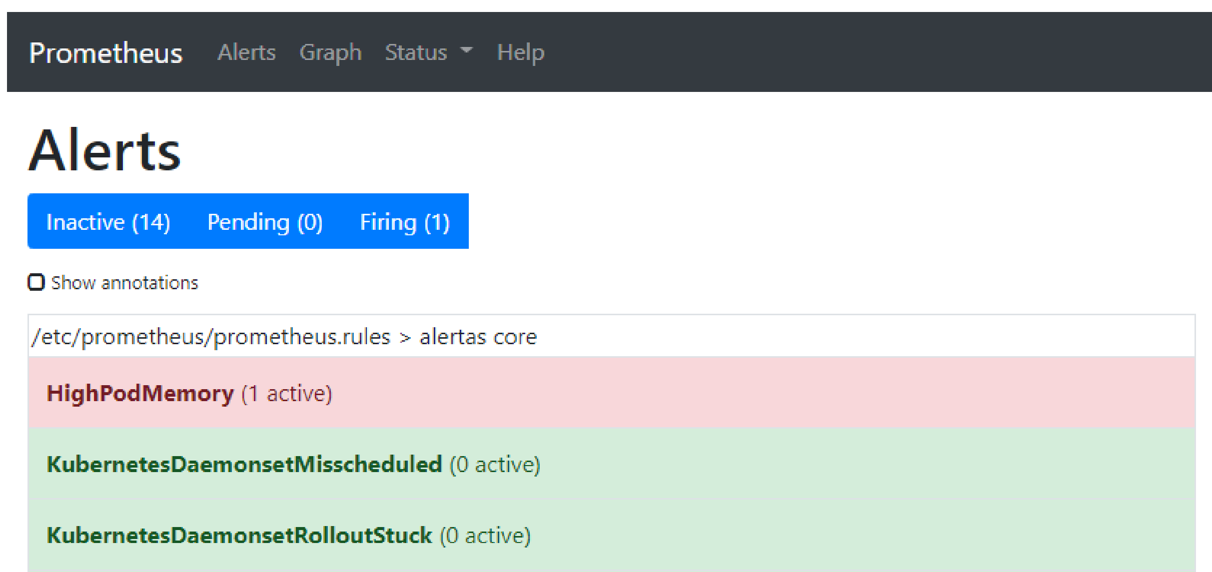


Figura 4-33: pantalla de alertas en la aplicación Prometheus

Y si pulsamos sobre una alerta en concreto podemos ver detalles sobre dicha alerta, tanto su configuración como, en caso de estar disparada, desde cuándo está activa y su valor actual de métrica. Un ejemplo de esto lo podemos ver en la Figura 4-34.

/etc/prometheus/prometheus.rules > alertas core

HighPodMemory (1 active)

```
alert: HighPodMemory
expr: sum(container_memory_usage_bytes) > 1
for: 1m
labels:
  class: resource_usage
  severity: normal
annotations:
  summary: High Memory Usage
```

Labels	State	Active Since	Value
alertname="HighPodMemory" class="resource_usage" severity="normal"	FIRING	2020-09-14 17:58:37.482489795 +0000 UTC	6.380269568e+09

Figura 4-34: detalle de una alerta, vista desde la sección de alertas de la aplicación Prometheus

Otro de los puntos de menú que puede resultar interesante es el relativo a las fuentes de métricas que tenemos configuradas y estado actual. Se puede acceder a este punto desde el menú *Status*, y dentro de él, la opción *Targets*. En la Figura 4-35 vemos las primeras dos entradas, que se corresponden con dos de las fuentes de métricas configuradas (dentro del fichero mostrado en la Figura 4-25). De entre los datos que se muestran, se puede ver el *endpoint* al que se conecta Prometheus para extraer métricas, el estado actual de dicho *endpoint* y hace cuánto tiempo hace de la última vez que se realizó la conexión.

En la aplicación de Prometheus hay otros puntos de menú que muestran otro tipo de información, pero no son relevantes para el entendimiento y alcance de este trabajo.

Prometheus Alerts Graph Status Help

Targets

All Unhealthy

kube-state-metrics (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://kube-state-metrics.kube-system.svc.cluster.local:8080/metrics	UP	instance="kube-state-metrics.kube-system.svc.cluster.local:8080" job="kube-state-metrics"	2.618s ago	78.96ms	

kubernetes-apiservers (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://192.168.178.30:6443/metrics	UP	instance="192.168.178.30:6443" job="kubernetes-apiservers"	4.245s ago	2.174s	

Figura 4-35: primeros elementos del listado de fuentes de métricas configuradas para Prometheus, vista desde la aplicación Prometheus

Como se ha podido ver, desde esta aplicación no es posible modificar nada. Se trata pues de una aplicación meramente consultiva. Aunque se debe tener cuidado con las redes a las que se expone, ya que desde la aplicación muestra datos que dan información sobre el clúster, y un posible atacante podría conseguir información sobre la infraestructura (*endpoints*, servicios, *namespaces*, etc.) o sobre el estado de ciertas aplicaciones (memoria y CPU en uso, etc.) que podría usar para intentar realizar algún ataque.

4.3.2. Alert manager

Por medio de la aplicación Alert manager (Prometheus, 2020) podemos gestionar qué hacer con las alertas una vez que se produzcan y enviar notificaciones si así se considera. Como habíamos dicho en el punto 4.3.1, es Prometheus el que está pendiente de analizar continuamente las métricas del estado de todos los objetos analizados y, en caso de que se produzca una situación definida en alguna de las alertas configuradas en su configuración, enviará una alerta a esta aplicación, *alert manager*, que será quien se encargue de gestionar

esa alerta y, si corresponde, enviará un notificación. Cuando Prometheus detecte que la anomalía se ha resuelto, enviará de nuevo la actualización a esta aplicación, que a su vez la atenderá según corresponda.

Teniendo en cuenta lo anterior, el flujo básico de una alerta es el siguiente:

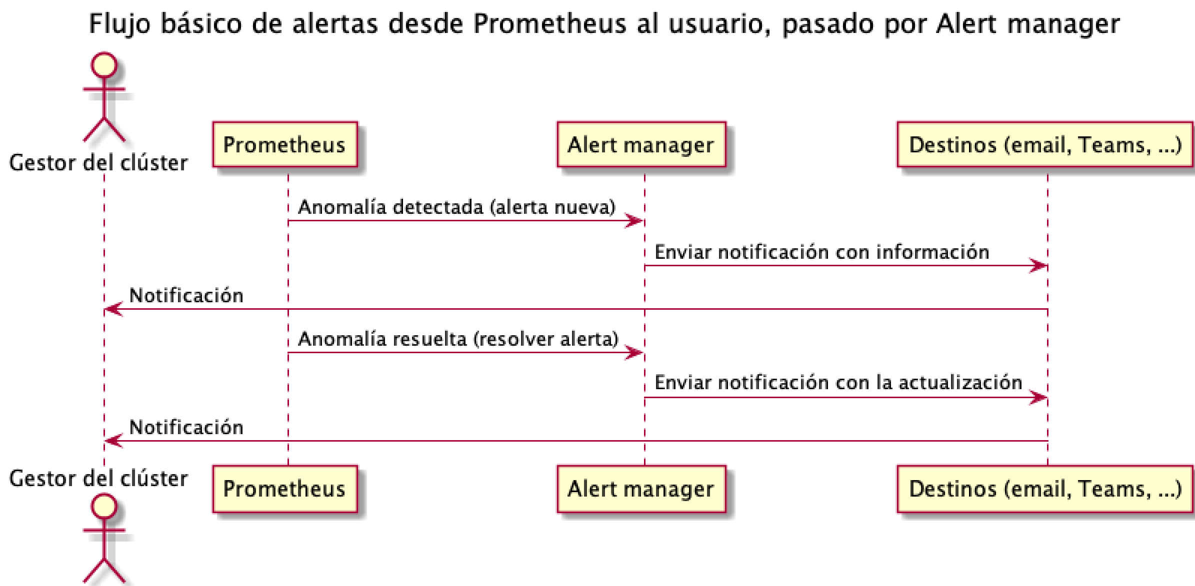


Figura 4-36: flujo básico de las alertas detectadas por Prometheus

Lo más interesante es sin duda la posibilidad de poder enviar notificaciones acordes a unas condiciones que podemos configurar. Para entender mejor el potencial y cómo se puede configurar, pongamos un ejemplo de alerta configurada en Prometheus (en el punto 4.3.1 pusimos el contenido del fichero de configuración de alertas, pero sin poner ninguna alerta. Aquí es donde ponemos un ejemplo de estas):

```
- alert: KubernetesNodeReady
  expr: kube_node_status_condition{condition="Ready",status="true"} == 0
  for: 5m
  labels:
    severity: critical
    class: infra
  annotations:
    summary: "Nodo de Kubernetes no disponible (instancia {{ $labels.instance
}})"
    description: "El node {{ $labels.node }} ha estado no disponible por demas
iado tiempo\n  VALUE = {{ $value }}\n  LABELS: {{ $labels }}"
```

Figura 4-37: ejemplo de configuración de una alerta en Prometheus

En este ejemplo de alerta nos estamos fijando en la métrica llamada `kube_node_status_condition`, filtrándola por los atributos `condition="Ready"` y `status="true"` y comparándola con el valor `0`. Además, esta situación se debe dar durante 5 minutos. Es decir, cuando esa métrica, habiéndola filtrado por esos atributos, tenga el valor 0 durante 5 minutos, la alerta se dispara. Cuando dicha métrica deje de tomar el valor 0, la alerta se tomará como resuelta.

Además, en la alerta se configuran unas etiquetas y unas anotaciones. Las etiquetas sirven para catalogar la alerta según los datos que consideremos, pudiendo añadir cualquier etiqueta con cualquier valor. Estas etiquetas nos servirán posteriormente para poder catalogarlas, así que es recomendable definir un conjunto suficiente de etiquetas distintas y que tomen valores descriptivos. En este caso se han definido 2 etiquetas: `severity: critical` y `class: infra`.

Las anotaciones nos sirven para definir variables que estarán disponible cuando se explote la alerta. Y en su propia definición podemos hacer referencia a varias variables que permiten enriquecer el valor, en formato texto, que tomará cuando la alerta se dispare. En esta alerta hemos definido las variables `summary` y `description`.

La configuración anterior se hace en Prometheus, que es donde se definen las alertas y donde se detecta cuándo deben ser disparadas.

Por otro lado, en alert manager debemos configurar una ruta, que nos sirve para definir un grupo de escucha de alertas y qué hacer cuando nos llegue una alerta a este grupo. Esto es parte del `ConfigMap` de configuración que usa la aplicación, y que podemos ver en la Figura 4-38:

```yaml
kind: ConfigMap
apiVersion: v1
metadata:
  name: alertmanager-config
  namespace: monitoring
data:
  config.yml: |-
    global:
      smtp_auth_username: usuario@dominio.com
      smtp_auth_password: password123
      smtp_from: cluster@dominio.com
      smtp_smarthost: smtp.dominio.com:465
      smtp_require_tls: false
    templates:
    - '/etc/alertmanager/*.tmpl'
    route:
      receiver: alert-notification
      group_by: ['alertname', 'priority']
      group_wait: 10s
      repeat_interval: 30m
      routes:
        - receiver: alert-notification
          match_re:
            severity: ".*"
          group_wait: 10s
          repeat_interval: 1m
    receivers:
    - name: alert-notification
      email_configs:
      - to: atellez9@alumno.uned.es, otrousuario@dominio.com
        send_resolved: true
      webhook_configs:
      - url: "http://alertmgr-msteams:2000/alertmanager"
        send_resolved: true
```

Figura 4-38: configuración de la aplicación alert manager. Fichero 30-alertmgr-configmap.yaml

Vamos a explicar por partes el contenido del fichero declarado `config.yml`:

- `global`: en esta sección definimos datos globales y comunes al resto de la configuración. En nuestro caso definimos los datos necesarios para la utilización del servidor SMTP de envío de correos. Aunque no lo hemos dejado, a falta de un servidor concreto, hemos configurado como servidor SMTP Gmail, usando una cuenta personal. No obstante, idealmente se debería configurar un servidor SMTP perteneciente a la UNED.

- `templates`: listado de rutas de ficheros que contienen las plantillas usadas para los mensajes de notificaciones gestionadas propiamente en la aplicación. Estas plantillas serán usadas para formatear el contenido de los mensajes que se generen y se manden como notificaciones. Se pueden definir varias plantillas y que se use la que corresponda en cada caso.
- `route`: además de unos parámetros generales de configuración, en esta sección se configuran las rutas que mencionábamos anteriormente para gestionar las alertas que vayan llegando. Principalmente, lo que hemos configurado en este caso es que las alertas que encajen con las etiquetas y valores (evaluados como expresiones regulares) definidas dentro de `match_re` se enviarán al destino definido en `receiver`. Al configurar que como grupo de selección únicamente `severity: ".*"` lo que hacemos es aceptar cualquier alerta que llegue que tenga la etiqueta `severity`, independientemente de su valor (ya que como valor aceptamos cualquier cosa)
- `receivers`: aquí podemos declarar el listado de grupos a los que mandar las notificaciones. En nuestro caso sólo hemos definido un grupo, pero dentro del mismo hemos definido 2 destinos para las notificaciones: email y *webhook*. Los email destino son los indicados. Y como *webhook* lo que hacemos es mandar la información a otra aplicación desplegada en el clúster, dentro del mismo namespace, y que se encargará de enviar las notificaciones a un canal creado en Microsoft Teams.

No vamos a entrar en detalle sobre cómo funciona la aplicación que envía notificaciones a Microsoft, sólo aclarar que esta aplicación ya existía y lo que hemos hecho ha sido clonarla y construir una imagen Docker para ARMv7, ya que la imagen original no estaba disponible para nuestra arquitectura. Esta aplicación requiere de una configuración de la URL del *webhook* para el canal de Teams que hayamos creado, y se encargará de ir enviando las notificaciones a dicho canal.

Alert manager tiene implementada la posibilidad de envío de notificaciones a otros destinos, como Slack. Y, en cualquier caso, si queremos otra forma de procesar la notificación podemos hacer de igual modo que lo que hemos hecho con la aplicación de envío de

notificaciones para Teams (clonar una aplicación existente y construirla para nuestra arquitectura), o incluso podemos construir una aplicación propia en caso de que sea necesario.

En cualquier caso, y a modo de ejemplo, con la configuración anterior recibiremos notificaciones tanto al buzón de correo electrónico como a un canal de Microsoft Teams. En caso de que un administrador del clúster reciba alguna alerta, éste debería consultar los detalles de la misma y acudir a resolver la incidencia, lo que podría pasar por consultar más información en Grafana, o directamente operar con *kubectl* sobre el clúster, por mencionar únicamente el par de acciones más comunes que el autor ha debido realizar durante la realización del trabajo.

Para ilustrar con un ejemplo el resultado de las notificaciones hemos forzado una situación anómala en tanto que hemos apagado uno de los nodos, forzando así la situación definida en la alerta descrita en la Figura 4-37, recibiendo las notificaciones de la Figura 4-40 y Figura 4-39.

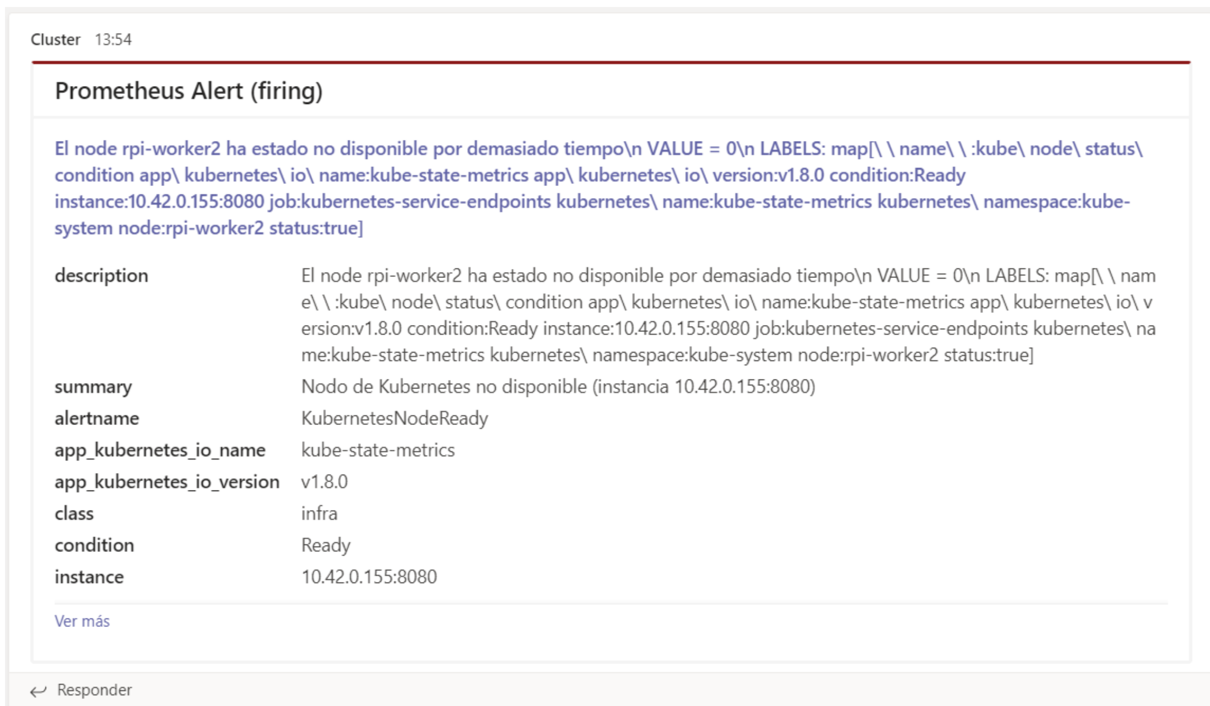


Figura 4-39: notificación recibida al canal de MS Teams tras disparo de una alerta

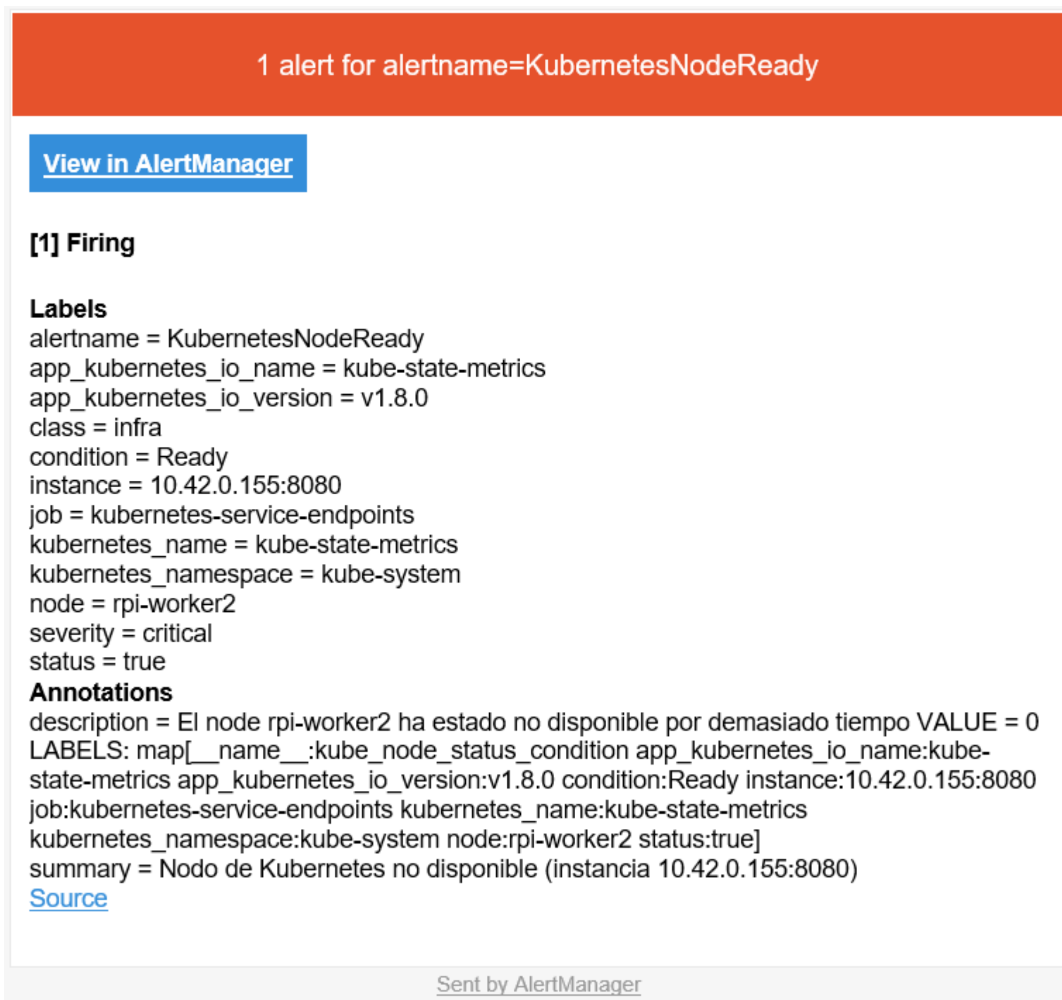


Figura 4-40: notificación recibida al buzón de correo electrónico tras disparo de una alerta

Por otro lado, otro fichero de configuración necesario para la configuración para *alert manager* es otro `ConfigMap` donde definimos las plantillas por defecto que se van a usar, salvo que se indique lo contrario (Figura 4-41).

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: alertmanager-templates
  namespace: monitoring
data:
  default.tmpl: |
…
```

Figura 4-41: plantillas por defecto usadas para las notificaciones en alert manager. Fichero 31-alertmgr-templates.yaml

Hay que tener claro presente que esta plantilla por defecto aplica al envío de correo que habíamos visto antes, ya que se produce y manda dentro de *alert manager*. Sin embargo, en casos de que las notificaciones se manden desde otras instancias, como era caso de las

notificaciones Teams, estas plantillas no tienen aplicación y se usarán las plantillas definidas en dichas instancias.

El despliegue de la aplicación lo hacemos por medio del siguiente fichero (Figura 4-42):

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alertmanager
  namespace: monitoring
spec:
  replicas: 1
…
    spec:
      containers:
      - name: alertmanager
        image: prom/alertmanager:v0.19.0
        args:
          - "--config.file=/etc/alertmanager/config.yml"
          - "--storage.path=/alertmanager"
          - "--log.level=debug"
        ports:
        - name: alertmanager
          containerPort: 9093
        volumeMounts:
        - name: config-volume
          mountPath: /etc/alertmanager
        - name: templates-volume
          mountPath: /etc/alertmanager-templates
        - name: alertmanager
          mountPath: /alertmanager
      volumes:
      - name: config-volume
        configMap:
          name: alertmanager-config
      - name: templates-volume
        configMap:
          name: alertmanager-templates
      - name: alertmanager
        emptyDir: {}
```

Figura 4-42: despliegue de la aplicación alert manager. Fichero 32-alertmgr.yaml

Y, como cualquier otra aplicación en kubernetes que queramos que esté expuesta dentro del clúster, debemos crear un servicio asociado a la aplicación, tal como vemos en la Figura 4-43.

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: alertmanager
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/path:   /
      prometheus.io/port:   '8080'
spec:
  selector:
    app: alertmanager
  ports:
    - port: 9093
      targetPort: 9093
```

Figura 4-43: exposición de la aplicación alert manager. Fichero 32-alertmgr.yaml

Como podemos ver, en el despliegue hacemos referencia a los ficheros de configuración mencionado anteriormente. Y, además, le hemos fijado un nivel de log *debug*, para que muestre más información en el log. Esta configuración puede ser desactivada una vez esté instalado y funcionando correctamente el clúster.

Alert manager no es necesario exponerlo fuera del clúster, ya que únicamente será consumido internamente por Prometheus. De hecho, es aconsejable que no sea accesible desde fuera para que se puedan enviar alertas falsas que podrían generar ruido y falsas alarmas.

4.3.3. Grafana

Hasta ahora hemos preparado Prometheus, que gestiona las métricas de las que dispone sobre el uso y estado del clúster. Y hemos preparado Alert manager para enviar notificaciones sobre las alertas que se van produciendo. Sin embargo, no tenemos ninguna manera de consultar el estado general del clúster de forma sencilla y rápida. Y es aquí donde metemos Grafana.

Grafana es una herramienta ideada para la observación de datos métricos, permitiendo crear gráficos con multitud de datos y de muy diversos formatos, obteniendo los

datos desde las fuentes de datos configuradas, pudiendo ser estas fuentes muy diversas. En nuestro caso, la fuente de datos será la instancia de Prometheus de la que hemos hablado en el punto 4.3.1. Como configuración para Grafana a este respecto tenemos un `ConfigMap` (Figura 4-44).

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-datasources
  namespace: monitoring
data:
  prometheus.yaml: |-
    {
        "apiVersion": 1,
        "datasources": [
            {
               "access":"direct",
                "editable": true,
                "name": "prometheus",
                "orgId": 1,
                "type": "prometheus",
                "url": "https://tfm-atellez.ngrok.io/prometheus",
                "version": 1
            }
        ]
    }
```

Figura 4-44: configuración para Grafana. Fichero 40-grafana-datasource-configmap.yaml

Con la configuración anterior indicamos que la fuente de datos será nuestra instancia de Prometheus, a la que accederemos de forma directa (esto es, desde el propio navegador) por medio de la ruta externa que tenemos configurada.

Por otro lado, y teniendo en cuenta que no disponemos de un almacenamiento permanente para Grafana y por tanto en cada reinicio perderemos cualquier posible dato guardado, configuramos información sobre dónde se encuentran los *dashboards* disponibles en el propio contenedor de Grafana por medio de la configuración de la Figura 4-45:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboardcfg
  namespace: monitoring
data:
  all.yaml: |-
    - name: 'default'
      org_id: 1
      folder: ''
      type: 'file'
      options:
        folder: '/var/lib/grafana/dashboards'
```

Figura 4-45: configuración sobre los dashboards de Grafana. Fichero 41-grafana-dashboard-configmap.yaml

Básicamente, lo que aquí configuramos es que la carpeta donde se encontrarán almacenados los *dashboards* será en la ruta `/var/lib/grafana/dashboards`. Sin embargo, aquí no definimos ningún *dashboard*, sólo la ubicación en la que buscarlos.

Un *dashboard* en Grafana se compone de una serie de paneles previamente configurados. Cada uno de estos paneles ejecuta un conjunto de consultas sobre la fuente de datos configurada, Prometheus en nuestro caso, y muestra los resultados de forma gráfica, tabla o numérica. De esta forma podemos visualizar de forma sencilla toda la información que hayamos configurado. Para nuestro clúster hemos creado un *dashboard*, y para ponerlo a disposición de la aplicación, lo incluimos dentro de un fichero de configuración:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboard
  namespace: monitoring
data:
  dashboard.json: |-
…
```

Figura 4-46: dashboard creado para Grafana. Fichero 42-grafana-dashboard.yaml

No hemos incluido el *dashboard* en sí por tratarse de un fichero de varios miles de líneas, y que no aporta información de interés para efectos de este documento. El *dashboard* usado se ha basado en los *dashboard* mostrado en (Eduardo, 2020) y (prat0318, 2020), sobre los que hemos hecho varias modificaciones para añadir y eliminar ciertos elementos, así como adaptaciones en varios paneles para ajustarlo a nuestro clúster y las métricas de las que disponemos.

Es importante aclarar que el *dashboard* está definido en el fichero de la Figura 4-46 y que cualquier cambio que se pueda hacer en el *dashboard* desde Grafana no se podrá guardar, ya que este tipo de objetos no son editables desde las aplicaciones. Por tanto, si se desease hacer alguna modificación en el *dashboard*, finalmente se deberá actualizar el contenido de este fichero y así, en el siguiente reinicio de la aplicación Grafana, cogerá el nuevo contenido del *dashboard*. Una alternativa a esto podría ser el uso de un volumen persistente en kubernetes, que a efectos prácticos se trata de un almacenamiento persistente, donde sí podrían guardarse los cambios realizados en el *dashboard*. Pero este enfoque supondría una exportación periódica de dicho contenido ya que al estar contenido directamente dentro del clúster no podría ser integrado en otro clúster a menos que se exporte e importe de alguna manera. Además, es de suponer que el *dashboard* de Grafana no se modificará con mucha frecuencia ya que una vez refinado no debería sufrir muchos cambios.

Una vez que tenemos los ficheros de configuración disponibles, podemos desplegar Grafana, usando su correspondiente despliegue (Figura 4-48) y exposición interna (Figura 4-48). Este despliegue se basa en buena medida en (Wilson, How To Setup Grafana On Kubernetes, 2019).

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      name: grafana
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana:latest
        ports:
```

```yaml
  - name: grafana
    containerPort: 3000
  env:
  - name: GF_SERVER_SERVE_FROM_SUB_PATH
    value: "true"
  - name: GF_SERVER_ROOT_URL
    value: "https://tfm-atellez.ngrok.io/grafana/"
  - name: GF_SECURITY_ADMIN_USER
    value: "****"
  - name: GF_SECURITY_ADMIN_PASSWORD
    value: "****"
  volumeMounts:
    - mountPath: /var/lib/grafana
      name: grafana-storage
    - mountPath: /etc/grafana/provisioning/dashboards
      name: grafana-dashboardcfg
    - mountPath: /etc/grafana/provisioning/datasources
      name: grafana-datasources
      readOnly: false
    - mountPath: /var/lib/grafana/dashboards
      name: grafana-dashboard
      readOnly: false
volumes:
  - name: grafana-storage
    emptyDir: {}
  - name: grafana-datasources
    configMap:
        defaultMode: 420
        name: grafana-datasources
  - name: grafana-dashboardcfg
    configMap:
        defaultMode: 420
        name: grafana-dashboardcfg
  - name: grafana-dashboard
    configMap:
        defaultMode: 420
        name: grafana-dashboard
```

Figura 4-47: despliegue de Grafana. Fichero 43-grafana.yaml

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: monitoring
  annotations:
      prometheus.io/scrape: 'true'
      prometheus.io/port:   '3000'
spec:
  selector:
    app: grafana
  ports:
    - port: 3000
      targetPort: 3000
```

Figura 4-48: exposición interna de Grafana. Fichero 43-grafana.yaml

En relación al despliegue, definimos las variables de entorno `GF_SERVER_SERVE_FROM_SUB_PATH` y `GF_SERVER_ROOT_URL` para indicar que la aplicación estará accesible desde una URL específica, y que esta URL tendrá una sub ruta. Esta configuración es importante, ya que sin ella no podríamos tener funcionamiento de forma correcta la aplicación tal como gestionamos la exposición de la aplicación fuera de la red local. Por otro lado, configuramos las variables `GF_SECURITY_ADMIN_USER` y `GF_SECURITY_ADMIN_PASSWORD` como usuario y contraseña administrador de la aplicación. Este usuario será el que tenga permiso para acceder a la aplicación. Otra posibilidad, dependiendo de cómo se exponga la aplicación, podría ser el habilitar el acceso anónimo a Grafana en modo de sólo lectura, lo que permitiría ver el *dashboard* y la información ahí contenida sin necesidad de usar ningún usuario. Esta posibilidad es interesante si Grafana está expuesto únicamente dentro de una intranet o por acceso vía VPN.

También se ha configurado en el despliegue los diferentes `ConfigMap` mencionados anteriormente como volúmenes, en las rutas especificadas. Como mención especial indicar que el fichero relativo al *dashboard* se monta sobre la ruta `/var/lib/grafana/dashboards`, que es la ruta que estaba configurada en otro de los ficheros de configuración que habíamos mencionado. De esta forma aseguramos que nuestro *dashboard* estará disponible cuando se arranque el contenedor.

Por último, exponemos Grafana hacia fuera del clúster por medio de un `Ingress`, definido en la Figura 4-49.

```yaml
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: monitoring
  name: grafana-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
spec:
  rules:
  - http:
      paths:
      - path: /grafana
        backend:
          serviceName: grafana
          servicePort: 3000
```

Figura 4-49: exposición hacia fuera del clúster de Grafana. Fichero 44-grafana-exponer.yaml

Exponemos Grafana bajo la ruta `/grafana`, tal como habíamos definido en una de las variables de entorno del propio despliegue de la aplicación.

Una vez que hemos aplicado todos los ficheros anteriores, y una vez que la réplica esté disponible, podremos acceder a la aplicación y desde ahí, a nuestro *dashboard*, donde veremos algo similar a los siguientes paneles, que se corresponden con la visión completa que se puede ver con el *dashboard* definido, en un momento dado.

Para acceder a la aplicación basta con ir a la URL y aparecerá la pantalla de autenticación (Figura 4-50) donde debemos introducir las credenciales de acceso que habíamos configurado en el despliegue de la aplicación (Figura 4-48).

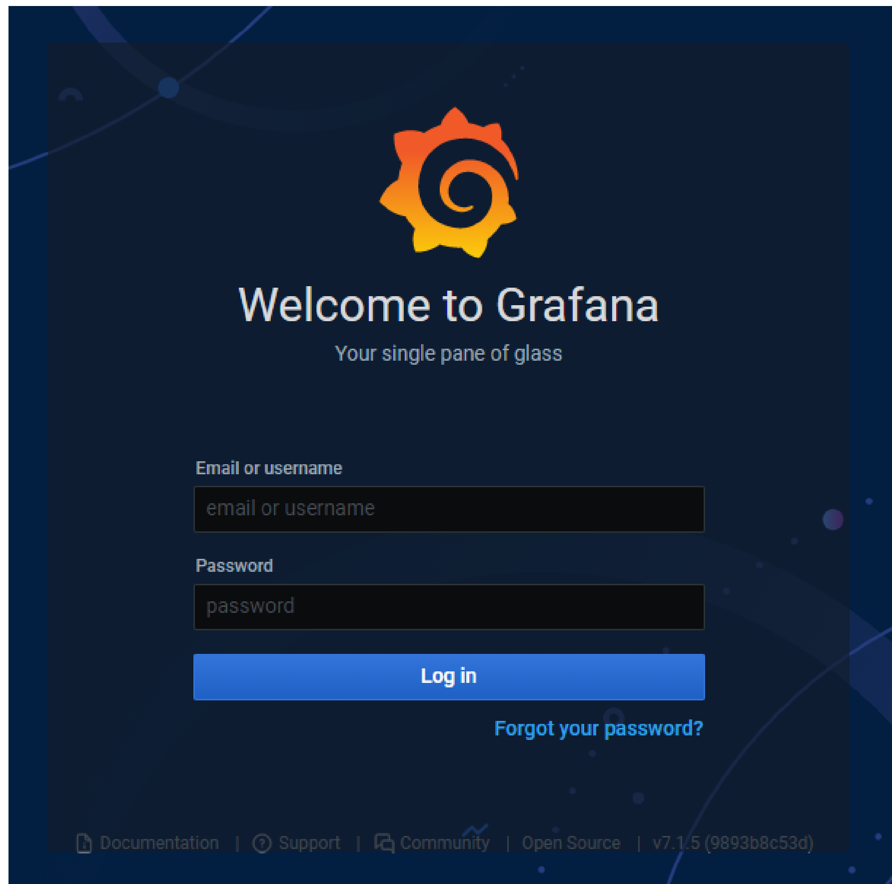


Figura 4-50: pantalla de autenticación de Grafana

Tras autenticarnos correctamente llegamos a la página de inicio, donde ya podemos ver nuestro *dashboard* disponible, tal como se ve en la Figura 4-53.

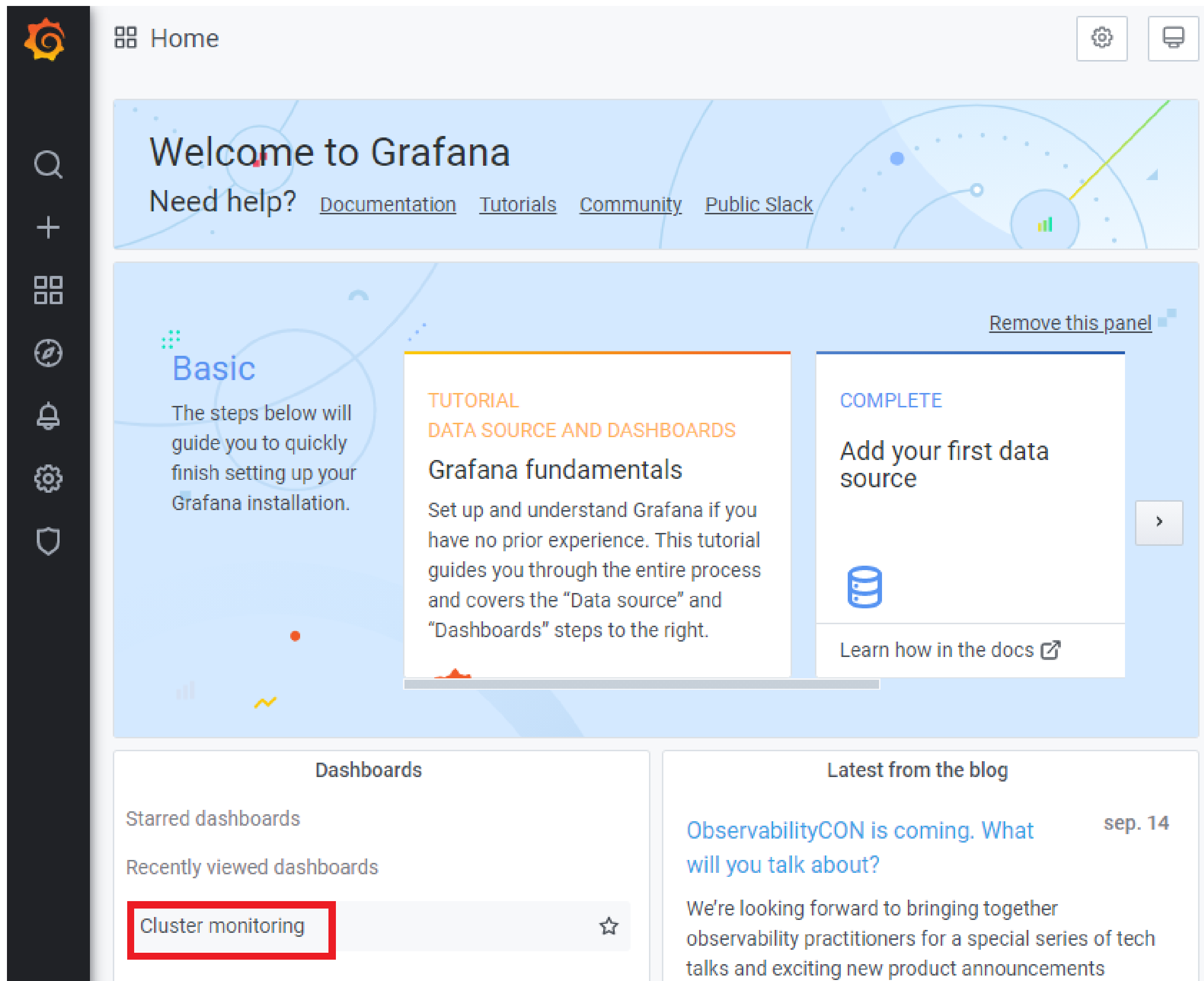


Figura 4-51: pantalla de inicio de Grafana, tras una correcta autenticación. En la parte de abajo a la izquierda se puede ver el dashboard ya creado

Si pinchamos directamente sobre el *dashboard* llegaremos a él, donde veremos toda la información que el dashboard ofrece (Figura 4-52, Figura 4-53, Figura 4-54, Figura 4-55, Figura 4-56 y Figura 4-57).

Cluster monitoring

Last 3 hours | 10s

Nodos

node	kernel_version	kubelet_version	os_image	pod_cidr	app_kubernetes_io_version
rpi-master	4.19.118-v7+	v1.18.6+k3s1	Raspbian GNU/Linux 10 (buster)	10.42.0.0/24	v1.8.0
rpi-worker1	4.19.118-v7+	v1.18.6+k3s1	Raspbian GNU/Linux 10 (buster)	10.42.1.0/24	v1.8.0
rpi-worker2	4.19.118-v7+	v1.18.6+k3s1	Raspbian GNU/Linux 10 (buster)	10.42.2.0/24	v1.8.0
rpi-worker3	4.19.118-v7l+	v1.18.6+k3s1	Raspbian GNU/Linux 10 (buster)	10.42.3.0/24	v1.8.0

Tiempo desde último reinicio de nodo

Time	instance	Uptime
2020-09-06 18:41:21	192.168.178.30:9100	37.55 min
2020-09-06 18:41:21	192.168.178.31:9100	22.81 hour
2020-09-06 18:41:21	192.168.178.36:9100	22.80 hour
2020-09-06 18:41:21	192.168.178.37:9100	4.41 hour

Figura 4-52: sección del dashboard de Grafana que muestra información técnica (versión de número, IP, etc.) de cada uno de los nodos, así como el tiempo que llevan en funcionamiento

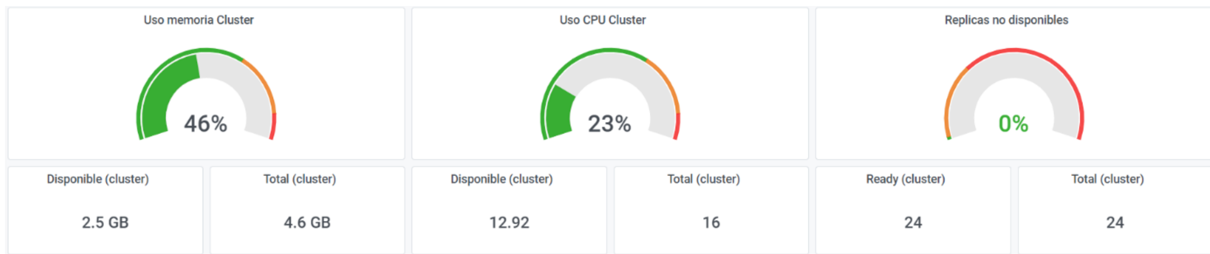


Figura 4-53: sección del dashboard de Grafana que muestra el uso global de memoria y CPU en el clúster, así como el porcentaje de réplicas no disponible

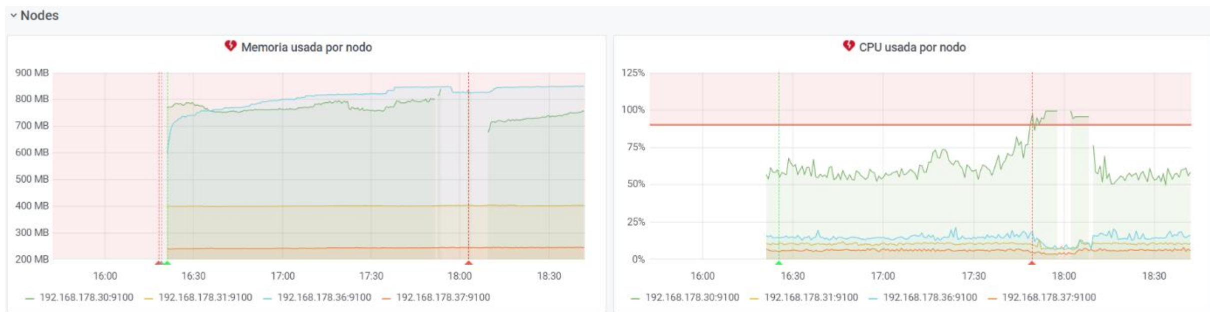


Figura 4-54: sección del dashboard de Grafana que muestra la memoria (MiB) y CPU (%) usada en cada uno de los nodos

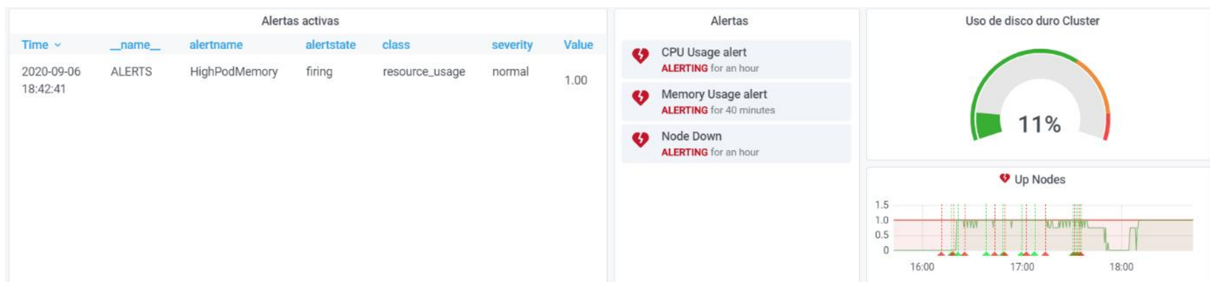


Figura 4-55: sección del dashboard de Grafana que muestra información sobre las alertas activas y sobre el porcentaje de almacenamiento físico usado en el clúster

Figura 4-56: sección del dashboard de Grafana que muestra el uso de memoria y CPU a nivel de pod

Figura 4-57: sección del dashboard de Grafana que muestra información sobre el tráfico de red a nivel de pod

El *dashboard* muestra información sobre:

- Información sobre qué nodos hay en el clúster
- Uso de memoria y CPU en los nodos
- Número de réplicas que no están disponibles
- Información sobre alertas disparadas
- Uso de memoria y CPU a nivel de pod
- Tráfico de red por contenedor y por pod

Se han escogido estas métricas por considerarse las más interesantes para mostrar la situación general del clúster. No obstante, en función del uso que se haga del clúster así como de las necesidades particulares que puedan surgir durante su explotación, es posible que sea necesario crear nuevos paneles o modificar algunos de los aquí incluidos.

Como comentario particular, en los paneles mostrados anteriormente se puede ver una situación extraña entre las 17:50 y las 18:10, que se correspondió a un período en el que las Raspberry master tuvo problemas y estuvo fuera de servicio, hasta que se efectuó un reinicio manual y se pudo restaurar el correcto funcionamiento. Hemos considerado dejar ilustrada específicamente esta situación ya que muestra claramente como todo el clúster se cae cuando la Raspberry master está caída. Esta situación la comentamos anteriormente en el punto 4.1 y aquí se ve reflejado cómo de vulnerable puede ser un clúster basado en una única master. También corresponde decir que esta situación en la que la master se ha visto

tan comprometida ha sucedido una única vez durante todo el desarrollo de este trabajo, y por tanto no es una situación que ocurra a menudo, ni se espera que pueda ser muy frecuente.

Por último, en la Figura 4-32 se había mostrado una consulta en Prometheus sobre el uso de la memoria, en bytes, en cada nodo del clúster. Esta misma consulta se corresponde con uno de los paneles en el *dashboard* y mostrados anteriormente (Figura 4-54). Simplemente por detallar un poco más, podemos ver que el panel en cuestión usa la misma consulta que habíamos usado en Prometheus (Figura 4-32), porque de hecho todas las consultas se realizan sobre Prometheus, pero los datos se muestran en forma de gráfica con la evolución durante el intervalo elegido, con la configuración de presentación que deseemos. Así, el panel relativo al uso de memoria en cada nodo se configura como se ve en la Figura 4-58.

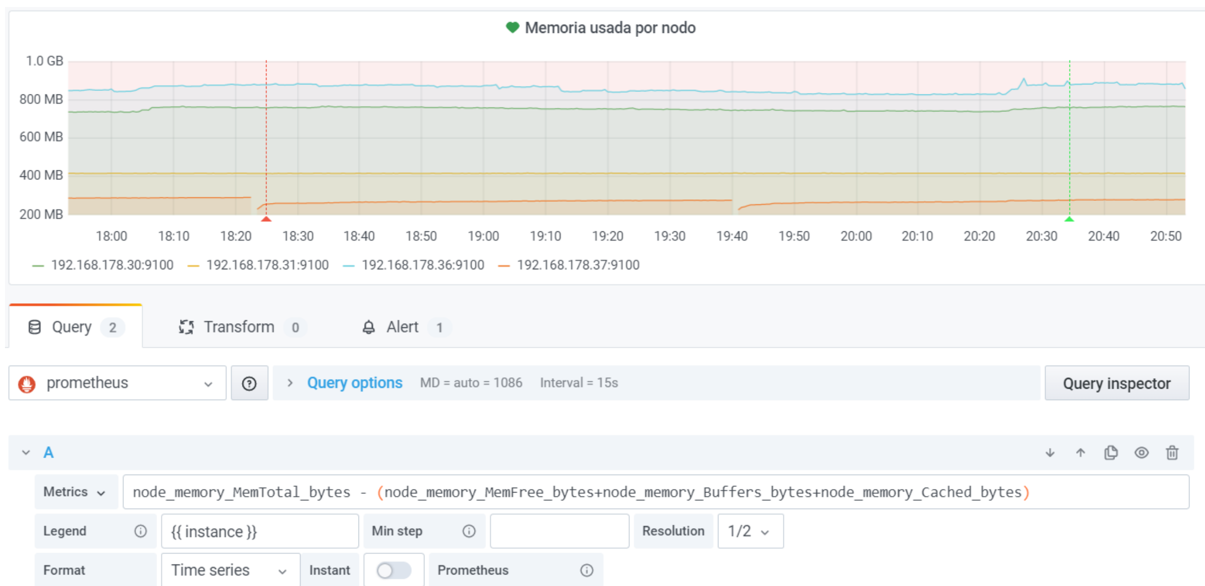


Figura 4-58: detalle de configuración en Grafana del panel relativo al uso de memoria en cada nodo

4.4. Dashboard de Kubernetes

Kubernetes ofrece la posibilidad de instalar un *dashboard* propio, también llamado Web UI, que muestra información sobre el clúster y permite realizar operaciones sobre el mismo.

Desde esta herramienta podemos ver qué objetos existen en el clúster, crear objetos nuevos, modificarlos o borrarlos. También permite reiniciar réplicas de forma muy sencilla.

Además, el acceso de esta herramienta se puede controlar por medio de una variedad de mecanismos, siendo uno de estos mecanismos el uso del token usado para la autenticación en el uso del clúster (el que se genera tras el proceso de autenticación). Se puede encontrar más información al respecto en (Kubernetes, 2020).

Es por estos motivos por lo que esta herramienta es muy útil… y muy peligrosa, precisamente por el hecho de mostrar tanta información y permitir realizar tantas acciones, ya que en malas manos podría causar grandes daños. Así pues, su exposición y acceso debe estar muy controlado y, en caso de pensar en la posibilidad de su acceso no esté correctamente asegurado es mejor no exponer el *dasboard*. En general, suele recomendarse exponer el *dashboard* únicamente en redes internas controladas, y usar algún mecanismo de autenticación para su acceso.

En cualquier caso, a efectos de este proyecto, y teniendo en cuenta que se ha trabajado sobre un clúster fuera de la red de la universidad, se ha expuesto a internet. En el anexo 1.C se explicará cómo hemos hecho la exposición del clúster hacia internet, y por qué hemos optado por esta vía.

Así pues, hemos desplegado el *dashboard* dentro de nuestro clúster y asegurando su acceso vía el mismo token de uso de clúster. Para ello, hemos instalado la aplicación usando el fichero yaml que Kubernetes pone a disposición, y que se puede consultar desde (Kubernetes, 2020), pero con una serie de cambios para adaptarlo a nuestras necesidades. Por motivos de claridad vamos a mencionar únicamente los cambios particulares hechos en nuestro despliegue, ya que el resto del fichero no aporta mucho valor en relación al trabajo aquí hecho. Así pues, en la parte relativa al `Deployment`, hemos dejado estos argumentos de arranque de la aplicación:

```
- --namespace=kubernetes-dashboard
- --insecure-bind-address=0.0.0.0
- --insecure-port=9090
- --enable-insecure-login=true
```

Figura 4-59: argumentos de arranque del contenedor de kubernetes dashboard. Parte del fichero dashboard.yaml

El primero de los argumentos venía en el fichero original y sirve para indicar el *namespace* donde se instalará la aplicación. El hecho de definir un *namespace* único para la

aplicación tiene sentido en relación con motivos de seguridad, ya que puede controlarse el acceso al *namespace* por medio de roles. Sin embargo, en nuestro clúster, en el que asumimos un conjunto reducido de roles, seguramente aquel que se encargue del mantenimiento del clúster tendrá permisos de administrador sobre el clúster, y quién no tenga dichos permisos no podrá acceder a este *namespace*, por lo que puede que nos diese igual usar un *namespace* único o el de seguridad creado anteriormente. No obstante, dejamos que se despliegue en un *namespace* independiente ya que, como poco, lo tenemos más ordenado e independiente.

Los argumentos que comienzan con la palabra insecure, `insecure-bind-address` y `insecure-port`, hacen referencia a que queremos habilitar el acceso vía HTTP y no únicamente por HTTPS. Esto será necesario para poder exponer la aplicación, tal cual hemos gestionado la exposición. De otra forma, tendríamos que instalar el certificado del dominio con el que se accede a la aplicación dentro del *namespace*, y en nuestro no es posible al no tener acceso a la clave privada del certificado usado (ver detalles en el anexo 1.C). En cualquier caso, lo que habilitamos es el acceso vía HTTP, pero dicho acceso únicamente será posible por medio del servicio que expone la aplicación, y dicho servicio, salvo dentro del clúster, solo estará disponible vía la URL externa que queremos configurar y que ya tiene configurada el acceso vía HTTPS.

El último argumento, `enable-insecure-login`, activa la pantalla de *login* según se accede al *dashboard* aun cuando se haga vía HTTP. Sin esta opción, el *dashboard* estaría desprotegido en cuanto a que cualquier usuario podría acceder.

De cara a una instalación real del clúster, es recomendable no configurar ninguno de los 3 últimos argumentos, aunque en realidad lo que realmente importa es no activar el acceso no seguro y dejar por tanto que sólo se pueda acceder vía HTTPS. Aunque esto supone poder disponer de un certificado (clave pública y privada) de la DNS donde se vaya a exponer el *dashboard*.

Además del despliegue de la aplicación, es necesario crear un usuario particular para su uso, con permisos suficientes para poder tener acceso a todo el clúster, tal como queremos.

Para ello debemos crear un usuario, `ServiceAccount`, y asociarlo a un rol ya existente dentro del clúster, el rol de administrador:

```yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Figura 4-60: creación de usuario para kubernetes dashboard y su asociación al rol correspondiente. Fichero dashboard-user.yaml

Por último, exponemos el *dashboard* hacia fuera del clúster:

```yaml
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: kubernetes-dashboard
  name: dashboard-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
    traefik.frontend.rule.type: PathPrefixStrip
spec:
  rules:
  - http:
      paths:
      - path: /dashboard/
        backend:
          serviceName: kubernetes-dashboard
          servicePort: 9090
```

Figura 4-61: exposición hacia fuera del clúster de kubernetes dashboard. Fichero exponer-dashboard.yaml

Así, el *dashboard* estará disponible bajo la ruta `/dashboard` de nuestro dominio de exposición.

Si bien el *dashboard* es el propio de Kubernetes, únicamente a modo de ejemplo mostramos la pantalla para la autenticación, a la que se llega según se accede, y la vista general de uno de los *namespaces* que componen el clúster:

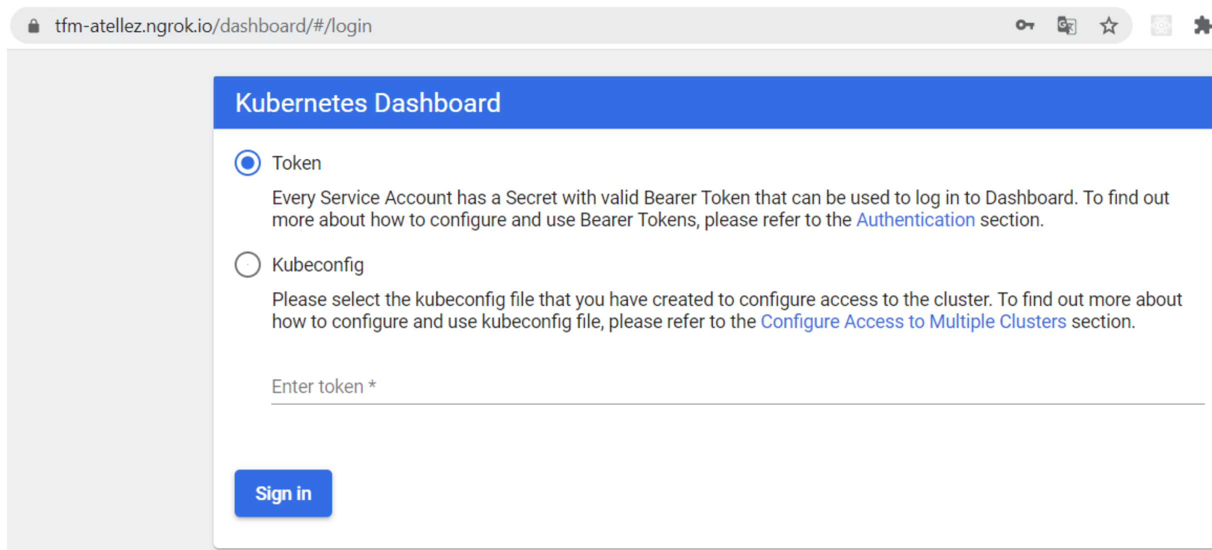


Figura 4-62: pantalla de autenticación para acceder al dashboard de Kubernetes

Figura 4-63: parte de la pantalla resumen del namespace seguridad en el dashboard de kubernetes

4.5. Aspecto final del clúster a nivel de kubernetes

Con todo lo mencionado en los puntos anteriores, y que básicamente se corresponde a objetos que son parte de la instalación que hemos hecho del clúster en sí, nos queda un clúster operativo y suficiente para poder empezar a trabajar en él.

Para tener una visión más clara del aspecto que nos queda el clúster una vez instalados todos los componentes que hemos usado para la configuración inicial, recopilamos los objetos e información más relevantes:

4.5.1. Namespaces

La estructura de *namespaces* es la siguiente:

```
λ kubectl get namespaces
NAME                   STATUS   AGE
default                Active   52d
kube-system            Active   52d
kube-public            Active   52d
kube-node-lease        Active   52d
seguridad              Active   52d
kubernetes-dashboard   Active   51d
monitoring             Active   50d
```

Figura 4-64: listado de namespaces tras la configuración del clúster

A modo resumen, la información contenida en cada uno de ellos:

- default: este es el espacio por defecto en kubernetes cuando se trabaja sin seleccionar ningún *namespace*. Probablemente no debería usarse ya que suele ser recomendable usar *namespaces* particulares para cada equipo y/o sistema que puedan ser fácilmente configurables en cuando a su acceso y permisos. Este *namespace* no contiene nada tras la instalación inicial.
- kube-system: espacio reservado a objetos propios de la instalación y funcionamiento del clúster en sí. Altamente recomendable no realizar ninguna acción en ninguno de los objetos aquí contenidos salvo que se esté completamente seguro de lo que se está haciendo.
- kube-public: es un espacio creado con la intención de que cualquier usuario, aun sin estar autenticado, pueda tener acceso al mismo. En caso de querer almacenar algo público en el clúster, puede ser este un buen espacio para hacerlo. Está gestionado por el propio kubernetes y general no necesitamos hacer nada sobre él.
- kube-node-lease: espacio usado por el controlador de nodos de kubernetes. Contiene únicamente un token que es usado para obtener información sobre los nodos. A efectos prácticos debería considerarse parte de la arquitectura básica de kubernetes y no realizar ninguna acción en este espacio.

- <u>seguridad</u>: espacio dedicado a la aplicación OIDC y lo relacionado con ella.
- <u>kubernetes-dashboard</u>: espacio para el *dashboard* de kubernetes.
- <u>monitoring</u>: espacio dedicado a la monitorización. Es aquí donde hemos instalado todo lo relacionado con Prometheus y Grafana.

En principio, y suponiendo una configuración de roles sencilla, seguramente se deba reservar el acceso únicamente al rol administrador a todos los *namespaces* anteriores salvo a `default` y `kube-public`, si así se considera. También se podrían dejar accesibles a otros roles, pero únicamente con permisos de lectura.

4.5.2. Despliegues y réplicas

Los despliegues y réplicas que hay en el clúster son los siguientes:

```
λ kubectl get deployments --all-namespaces
NAMESPACE                NAME                        READY
UP-TO-DATE   AVAILABLE   AGE
kubernetes-dashboard   dashboard-metrics-scraper   1/1    1
1          51d
kubernetes-dashboard   kubernetes-dashboard        1/1    1
1          51d
kube-system            metrics-server              1/1    1
1          52d
kube-system            coredns                     1/1    1
1          52d
kube-system            local-path-provisioner      1/1    1
1          52d
kube-system            kube-state-metrics          1/1    1
1          47d
kube-system            traefik                     1/1    1
1          52d
seguridad              oidc                        2/2    2
2          52d
```

```
    seguridad            redis                      1/1     1
1          52d
    monitoring           alertmanager               1/1     1
1          47d
    monitoring           alertmgr-msteams           1/1     1
1          9d
    monitoring           grafana                    1/1     1
1          47d
    monitoring           prometheus                 1/1     1
1          48d
```

Figura 4-65: listado de despliegues tras la configuración del clúster

Para tener claro a qué se corresponden, hagamos un pequeño repaso de los despliegues por cada *namespace*:

- kubernetes-dashboard: aquí tenemos:
  - `kubernetes-dashboard`: la aplicación del dashboard de kubernetes
  - `dashboard-metrics-scraper`: se encarga de recuperar algunas métricas desde el servidor de métricas de kubernetes para poder explotarlas desde el dashboard.
- kube-system: aquí podemos encontrar:
  - `metrics-server`: el servidor de métricas de kubernetes
  - `coredns`: gestiona los nombres de dominio dentro del clúster
  - `local-path-provisioner`: gestiona el uso de almacenamiento local en cada nodo del clúster.
  - `kube-state-metrics`: service que está pendiente de las operaciones sobre la API de kubernetes para generar métricas sobre su uso.
  - `traefik`: es el enrutador que expone que el clúster hacia fuera del mismo, conectándolo con los servicios internos que se hayan configurado para ello.
- seguridad: básicamente aquí tenemos lo que habíamos mencionado en el punto 4.2.1.
  - `oidc`: la aplicación OIDC para la autenticación y autorización.

- `redis`: base de datos usada por la aplicación OIDC para almacenar las sesiones.

- monitoring: aquí tenemos las aplicaciones que habíamos detallado en la sección 0:
  - alertmanager
  - alertmgr-msteams
  - grafana
  - prometheus

Se puede ver que todos los despliegues tienen 1 única réplica, a excepción de la aplicación OIDC, que tiene 2. Recordemos que idealmente debería haber un par de réplicas, al menos, para cada despliegue, pero que, debido a las capacidades limitadas de nuestro clúster, lo dejamos en 1 réplica para todo excepto para OIDC.

4.5.3. Servicios expuestos fuera del clúster

Los servicios que tenemos expuestos hacia fuera del clúster y, en nuestra implementación, a internet, son:

```
λ kubectl get ingresses --all-namespaces
NAMESPACE              NAME                       CLASS
HOSTS    ADDRESS           PORTS    AGE
kubernetes-dashboard   dashboard-ingress          <none>    *
192.168.178.31   80       51d
monitoring             grafana-ingress            <none>    *
192.168.178.31   80       47d
monitoring             prometheus-ingress         <none>    *
192.168.178.31   80       50d
seguridad              seguridad-oidc-ingress     <none>    *
192.168.178.31   80       52d
```

Figura 4-66: listado de servicios expuestos hacia fuera del clúster

Las aplicaciones que estamos exponiendo son las que habíamos mencionado con anterioridad a lo largo del documento:

- El dashboard de kubernetes
- Grafana
- Prometheus
- Aplicación OIDC

Estas 4 aplicaciones, y sólo estas 4 aplicaciones son las que se exponen hacia fuera, tal como tenemos el clúster montado.

Si bien en nuestra implementación lo que hacemos es exponerlas a internet, una exposición a una red interna requeriría igualmente de esta exposición ya que en realidad lo que hacemos con esto es únicamente disponerlos a cualquiera que pueda acceder a la URL del clúster desde fuera del mismo, ya sea este una red interna o internet.

4.5.4. Uso de memoria y CPU

Así con todo, los recursos utilizados por el clúster tras la instalación de todos los recursos mencionados anteriormente son los siguientes:

```
λ kubectl top nodes
NAME          CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
rpi-master    726m         18%    706Mi           76%
rpi-worker1   142m         3%     513Mi           55%
rpi-worker2   180m         4%     271Mi           29%
rpi-worker3   197m         4%     1034Mi          53%
```

Figura 4-67: recursos usados por el clúster una

En términos generales se está usando considerablemente más memoria que CPU, en porcentaje, y es que mientras que ninguno de los nodos supera el 20% de uso de CPU, todos los nodos superan el 30% de uso de memoria y, salvo uno de ellos, con creces. De hecho, si

combinamos los datos de todos los nodos tenemos que se está usando en torno a un 8% de CPU y un 50% de memoria.

4.6. Resumen del capítulo

En este capítulo hemos empezado mostrando la arquitectura de las piezas instaladas y cómo se relacionan unas con otras.

Una vez que tenemos la idea a alto nivel, entramos en detalle en cada unas de las piezas en las que se ha trabajado, tanto a nivel del desarrollo como de configuración de cada una de las piezas, así como mostrar las posibilidades que pueden ofrecer funcionalmente para los usuarios y administradores del clúster.

Por último, hemos visto cómo queda el clúster, a nivel de objetos creados y de recursos consumidos/libres, una vez que se ha realizado la instalación de todos los elementos que mencionados anteriormente.

5. Pruebas y resultados obtenidos

En este capítulo vamos a poner a prueba lo realizado durante este trabajo. Nos enfocaremos principalmente en comprobaciones de seguridad en relación con la autenticación y autorización, mostrando cuando aplique los datos generados en las diferentes aplicaciones de monitorización.

Todos los objetos que vayamos a crear se harán a partir de las plantillas que iremos detallando a continuación. En estas plantillas podremos ver hasta 2 variables, que se identifican por ir entre llaves dobles. Estas son:

- {{namespace}}. Nombre del *namespace*.
- {{name}}. Nombre del objeto que estemos creando, salvo cuando se trata de un *namespace*.

5.1. Preparación previa a las pruebas: detalle de operaciones que vamos a realizar durante las pruebas

Antes de comenzar con las pruebas, vamos a mostrar cuáles van a ser las diferentes operaciones que vamos a realizar sobre el clúster a lo largo de las siguientes pruebas, y que cuando referenciemos a las mismas, el lector pueda entender a qué nos estamos refiriendo.

5.1.1. Creación de un *namespace*

Como resultado de la instalación y preparación inicial del clúster nos queda un conjunto de objetos repartidos en diferentes *namespaces* (ver apartado 4.5). Estos objetos y *namespaces* deberíamos considerarlos como inmutables y no trabajar con ellos salvo que específicamente queramos hacerlo y seamos administradores del clúster.

Por tanto, para nuestras pruebas crearemos un *namespace* específico, así como diferentes objetos que sean necesarios para cada una de las pruebas.

Así pues, para la creación de un nuevo *namespace*, lo haremos con un fichero de configuración generado a partir de la plantilla de la Figura 5-1.

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: {{namespace}}
```

Figura 5-1: fichero con la plantilla de configuración de un namespace nuevo

5.1.2. Creación de rol por defecto con permisos de modificación para trabajar con cada *namespace*

Pero si tan solo creamos el *namespace* nadie que no tenga permisos sobre el clúster en sí (permisos gestionados con objetos de tipo `ClusterRole` y asociados por medio de objetos de tipo `ClusterRoleBinding`) podrá realizar ninguna acción, y realmente deseamos poder asignar grupos a usuarios (ver apartado 4.2.1.5) que se corresponden directamente con permisos sobre *namespaces* del mismo nombre. Es por esto por lo que además de crear un *namespace* debemos crear un `Role`[8] y un `RoleBinding` para dar permisos sobre un *namespace* a un grupo con el mismo nombre que el *namespace*.

En este apartado vamos a ver cómo vamos a crear los roles por defecto, con permisos de modificación sobre el *namespace*.

Debemos crear por un lado el rol en sí. En este objeto es donde definimos qué acciones y sobre qué objetos puede operar aquel que esté asociado a este rol. Esto lo tenemos en Figura 5-2. Como podemos ver en dicha figura, el rol nuevo se llamará *permisos-estandar* y estará contenido dentro del *namespace* recién creado.

[8] Un objeto de tipo Role define un conjunto de permisos sobre un *namespace* en particular. Todo lo aquí definido, una vez que se asocie con grupos/usuarios (vía un objeto de tipo RoleBinding) tendrá validez únicamente sobre el *namespace* donde se haya definido el Role.

```yaml
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: {{name}}
  name: permisos-estandar
rules:
- apiGroups: ["", "extensions", "apps"]
  resources: ["deployments", "configmaps", "services", "ingress", "horizontalp
odautoscalers"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Figura 5-2: fichero con la plantilla de configuración de un rol para un namespace nuevo

Por otro lado, debemos asociar el rol creado con el fichero de la Figura 5-2 al grupo con el mismo nombre que el *namespace*, y esto lo hacemos con el fichero definido en la Figura 5-3. El nombre del grupo no es únicamente el mismo nombre del *namespace* sino que tiene el prefijo `oidc:`, tal como habíamos configurado en el clúster y que podemos ver en el anexo A.

```yaml
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: permisos-estandar-rel
  namespace: {{name}}
subjects:
- kind: Group
  name: oidc:{{name}}
roleRef:
  kind: Role
  name: permisos-estandar
  apiGroup: rbac.authorization.k8s.io
```

Figura 5-3: fichero con la plantilla de configuración de una asignación de un rol a un grupo

5.1.3. Creación de rol con permisos de solo lectura para un *namespace*

Otro rol interesante es aquel que permite consultar, pero no realizar ninguna modificación. Este tipo de rol puede interesar cuando vamos a asignar grupos asociados al dominio de email del usuario logado, por ejemplo, ya que en estos casos estamos hablando de otorgar los permisos definidos en grupo a todo aquel usuario que pueda logarse cuyo dominio de email coincida, y estos usuarios pueden variar en el tiempo, con la posibilidad de que usuarios inesperados de repente puedan estar asignados al rol.

En nuestro caso, cualquier usuario cuyo dominio sea @alumno.uned.es estará asociado al grupo alumno (ver Figura 4-14), por lo que vamos a crear un rol de solo lectura y asignarlo a ese grupo. Y esta asociación solo tendrá validez para el namespace donde se cree el rol y la asociación. El fichero de creación del rol lo podemos ver en la Figura 5-4, y la asociación con el grupo alumno lo podemos ver en la Figura 5-5.

```
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: {{name}}
  name: permisos-solo-lectura
rules:
- apiGroups: ["", "extensions", "apps", "autoscaling", "batch", "networking.k8
s.io"]
  resources: ["deployments", "pods", "configmaps", "services", "ingresses",
    "horizontalpodautoscalers", "replicationcontrollers", "daemonsets",
    "replicasets", "statefulsets", "jobs", "cronjobs"]
  verbs: ["get", "list"]
```

Figura 5-4: fichero con la plantilla de configuración del rol de solo lectura

```
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: permisos-solo-lectura-rel
  namespace: {{name}}
subjects:
- kind: Group
  name: oidc:alumno
roleRef:
  kind: Role
  name: permisos-solo-lectura
  apiGroup: rbac.authorization.k8s.io
```

Figura 5-5: fichero con la plantilla de configuración de una asignación del rol de solo lectura al grupo alumno

5.1.4. Creación de una aplicación de pruebas

Una vez que hayamos creado un *namespace*, necesitaremos instalar algo dentro, y una forma sencilla de comprobar los permisos es instalar una aplicación y exponerla hacia fuera del clúster, probando tanto su instalación como la exposición. Para esta aplicación de pruebas hemos optado por la aplicación Node-RED, ya existente y disponible para la arquitectura usada por las Raspberry.

La creación de la aplicación dentro del *namespace* que corresponda y su exposición al clúster la hacemos con el fichero de la Figura 5-6. Y la exposición de la aplicación hacia fuera del clúster lo hacemos con el fichero de la .

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{name}}
  labels:
    app: {{name}}
  namespace: {{namespace}}
spec:
  replicas: 1
  selector:
    matchLabels:
      app: {{name}}
  template:
    metadata:
      labels:
        app: {{name}}
    spec:
      containers:
      - name: nodered
        image: nodered/node-red:1.0.5-12
        ports:
        - containerPort: 1880
          name: web
---
kind: Service
apiVersion: v1
metadata:
  labels:
    app: {{name}}
  name: {{name}}
  namespace: {{namespace}}
spec:
  ports:
    - port: 8080
      targetPort: web
      name: http
  selector:
    app: {{name}}
```

Figura 5-6: fichero con la plantilla de creación de una aplicación y su exposición al clúster

```yaml
---
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  namespace: {{namespace}}
  name: nred-{{name}}-ingress
  annotations:
    kubernetes.io/ingress.class: "traefik"
    traefik.frontend.rule.type: PathPrefixStrip
spec:
  rules:
  - http:
      paths:
      - path: /{{namespace}}/{{name}}/
        backend:
          serviceName: {{name}}
          servicePort: http
```

Figura 5-7: fichero con la plantilla de exposición de la aplicación creada hacia fuera del clúster

La exposición se hará bajo la ruta `/{namespace}/{nombre_aplicación}/` asegurándonos que así que la ruta será única, ya que el propio kubernetes no dejaría tener 2 aplicaciones distintas con el mismo nombre de un mismo *namespace*.

5.1.5. Aplicación de automatización de creación de objetos en kubernetes

Para simplificar el trabajo con los puntos anteriores sobre la creación de ciertos objetos, hemos creado una aplicación que funciona por línea de comandos y que recubre la creación de objetos relacionados en un solo comando, encargándose de lanzar los diferentes comandos de *kubectl* que correspondan. Además, precisamente al usar comandos de *kubectl* para trabajar con el clúster, se hace uso de las credenciales que haya en la configuración y por tanto, si el usuario que realiza la acción no está autorizado, recibirá un error. Estas situaciones se podrán ver más adelante en las pruebas realizadas del apartado 5.2.

Esta aplicación trabaja con las plantillas mostradas en los puntos anteriores (5.1.1, 5.1.2, 5.1.3 y 5.1.4) y con los parámetros introducidos al ejecutar el comando correspondiente. Así, formatea la plantilla con los datos introducidos y el contenido resultante se manda a kubernetes vía *kubectl*.

La aplicación se ha empaquetado como un ejecutable para Windows, Linux y Mac. Y aunque en la implementación actual se han incluido únicamente dos comandos (Figura 5-8) nada más (creación de un *namespace* y creación de una aplicación), es posible añadir más comandos en función de las necesidades que puedan surgir a futuro.

```
λ k8s-enmolabs-cmds-win.exe -h
Usage: bin [options]

Options:
  -V, --version              output the version number
  -c, --context <id>         Nombre del contexto a usar
  -ns, --namespace <id>       Crear un namespace con el nombre <id> y con roles especificos para el mismo
  -nred, --node-red <name>   Crear una instancia de Node-RED con el nombre <name> y bajo el namespace indicado
  -h, --help                 display help for command
```

Figura 5-8: información sobre la aplicación creada para la automación de creación de objetos en kubernetes

Para ejecutar la aplicación basta con ejecutar indicando el *namespace* únicamente si la operación que se quiere hacer es crear un nuevo *namespace*, junto con su rol de permisos escritura sobre el mismo, o indicando el *namespace* y el nombre de la aplicación Node-RED si lo que se quiere es crear una aplicación dentro de un *namespace* que se da por supuesto que ya existe. La información de ayuda de la aplicación se puede ver en la Figura 5-8.

A modo de ejemplo de su ejecución se puede ver la creación de un namespace con su rol de escritura (Figura 5-9) y la creación de una aplicación con su exposición interna y externa al clúster (Figura 5-10).

```
λ k8s-enmolabs-cmds-win.exe -ns nstest
[1600614943660] INFO  (k8s): Switched to context "rpi".
[1600614943681] DEBUG (k8s): Se va a cargar el fichero temporal "C:\Users\LVAROT~1\AppData\Local\Temp\tmp-15008-E7LKnaQyEi8r-.yaml"
[1600614945939] INFO  (k8s): namespace/nstest created
```

```
[1600614945941]            INFO                       (k8s):
role.rbac.authorization.k8s.io/permisos-estandar created
[1600614945943]            INFO                       (k8s):
rolebinding.rbac.authorization.k8s.io/permisos-estandar-rel
created
[1600614945943]            INFO                       (k8s):
role.rbac.authorization.k8s.io/permisos-solo-lectura created
[1600614945944]            INFO                       (k8s):
rolebinding.rbac.authorization.k8s.io/permisos-solo-lectura-
rel created
```

Figura 5-9: creación de un namespace con la aplicación creada para la automación de creación de objetos en kubernetes

```
λ k8s-enmolabs-cmds-win.exe -ns nstest -nred aplicacion1
[1600614993737] INFO  (k8s): Switched to context "rpi".
[1600614993757] DEBUG (k8s): Se va a cargar el fichero temporal
"C:\Users\LVAROT~1\AppData\Local\Temp\tmp-19380-djhMV9Foe2Ol-
.yaml"
[1600614996054] INFO  (k8s): deployment.apps/aplicacion1 created
[1600614996055] INFO  (k8s): service/aplicacion1 created
[1600614996056]  INFO   (k8s):  ingress.networking.k8s.io/nred-
aplicacion1-ingress created
```

Figura 5-10: creación de una aplicación con la aplicación creada para la automación de creación de objetos en kubernetes

5.2. Acceso administrador al clúster y crear un *namespace* nuevo y objetos en él

Para esta prueba,

- usaremos un usuario con permisos de administrador (usuario **atellez9**),
- crearemos un *namespace* nuevo, al que llamaremos `nstest1`,
- crearemos una aplicación (compuesta de un objeto `Deployment`, otro `Service` y otro `Ingress`).

Como resultado de este conjunto de operaciones esperamos que este usuario realice correctamente todas las operaciones, y que la aplicación instalada esté disponible desde fuera del clúster.

Antes de la realización de la prueba debemos autenticarnos en el clúster con el usuario que queremos usar, **atellez9**, y para ello haremos uso de la aplicación detallada en el punto 4.2.2. Tras la autenticación podemos ver que los grupos a los que pertenece el usuario[9] son: `alumno` y `cluster_admin`. Y tal como vimos en la Figura 4-5, el grupo `cluster_admin` está asociado con permisos de administrador sobre el clúster.

Una vez que nos hemos autenticado, hacemos algunas comprobaciones previas del usuario en activo y sus permisos sobre el futuro *namespace* (Figura 5-11).

```
λ REM consultamos cuál es el usuario actual
λ kubectl config view --minify -o jsonpath='{.contexts[0].context.user}'
'atellez9'
λ REM comprobación general de permisos del usuario
λ kubectl auth can-i create namespace
Warning: resource 'namespaces' is not namespace scoped
yes
λ kubectl auth can-i create role -n nstest1
yes
λ kubectl auth can-i create rolebinding -n nstest1
yes
λ kubectl auth can-i create deployment -n nstest1
yes
λ kubectl auth can-i create service -n nstest1
```

[9] ver punto 4.2.1.6 para más información sobre cómo extraer dichos grupos

```
yes
λ kubectl auth can-i create configmap -n nstest1
yes
λ kubectl auth can-i create ingress -n nstest1
yes
```

Figura 5-11: comprobación del usuario actual atellez9 y sus permisos sobre el namespace nstest1

Una vez que hemos comprobado los permisos que tiene el usuario actual, procedemos a la creación del namespace `nstest1` (Figura 5-12) y la instalación de la aplicación (Figura 5-13).

```
λ kubectl apply -f namespace_y_rol.yaml
namespace/nstest1 created
role.rbac.authorization.k8s.io/permisos-estandar created
rolebinding.rbac.authorization.k8s.io/permisos-estandar-rel
created
```

Figura 5-12: resultado de la creación del nuevo namespace nstest1 y un rol con permisos sobre los objectos de este

```
λ kubectl apply -f aplicacion.yaml
deployment.apps/nred1 created
service/nred1 created
ingress.networking.k8s.io/nred-nred1-ingress created
```

Figura 5-13: resultado del despliegue y exposición de la aplicación de prueba en el namespace nstest1

Ambas operaciones han resultado satisfactorias, pero comprobamos igualmente que se han creado los objetos que esperábamos dentro del *namespace*, tal como vemos en la Figura 5-14.

```
λ kubectl get all -n nstest1
NAME                            READY   STATUS    RESTARTS   AGE
pod/nred1-7544d7f7c6-9hrc5      1/1     Running   0          4m4s
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/nred1   ClusterIP   10.43.95.165   <none>        8080/TCP
4m4s
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nred1   1/1     1            1           4m4s
NAME                                DESIRED   CURRENT   READY   AGE
```

```
replicaset.apps/nred1-7544d7f7c6   1         1         1       4m4s
λ kubectl get roles -n nstest1
NAME                 CREATED AT
permisos-estandar    2020-09-17T10:00:42Z
λ kubectl get rolebindings -n nstest1
NAME                     ROLE                       AGE
permisos-estandar-rel    Role/permisos-estandar     6m42s
```

Figura 5-14: comprobación de que todos los objetos de la aplicación nred1 se han instalado dentro del namespace nstest1

Por último, vamos a comprobar que la aplicación se ha expuesto correctamente fuera del clúster, que es lo que habíamos configurado con el objeto de tipo `Ingress`, y para ello debemos ir a la URL formada por el dominio de exposición con el path `/nstest1/nred1/` y ver que la aplicación se carga correctamente, lo que efectivamente es así (Figura 5-15).

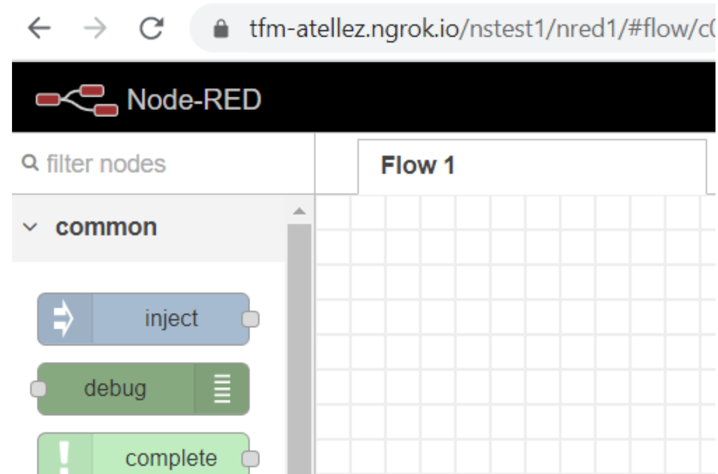


Figura 5-15: imagen de la aplicación instalada en el namespace nstest1 accediendo desde fuera del clúster

Así, podemos considerar esta prueba como satisfactoria al comprobar que un usuario al que le hemos asignado permisos de administrador puede crear un *namespace* nuevo y una aplicación en dicho *namespace*.

5.3. Acceso con usuario limitado a un *namespace* donde está autorizado y crear otra aplicación ahí

Para esta prueba,

- usaremos un usuario con permisos limitados (usuario **local_u1**) asociado a los grupos `alumno`, `nstest1` y `nstest2` (cuyo *namespace* aún no existe),
- crearemos una aplicación (compuesta de un objeto `Deployment`, otro `Service` y otro `Ingress`) en `nstest1`.

Como resultado de este conjunto de operaciones esperamos que este usuario pueda instalar correctamente la nueva aplicación y que esta esté accesible desde fuera del clúster

Tras realizar la autenticación en el clúster con el usuario que queremos usar, **local_u1**, vemos los grupos a los que pertenece el usuario son: `alumno`, `nstest1` y `nstest2`.

Una vez que nos hemos autenticado, hacemos algunas comprobaciones previas del usuario en activo y sus permisos sobre el futuro *namespace* (Figura 5-16).

```
REM consultamos cual es el usuario actual
kubectl config view --minify -o
jsonpath='{.contexts[0].context.user}'
'local_u1'
REM comprobacion general de permisos del usuario
kubectl auth can-i create namespace
Warning: resource 'namespaces' is not namespace scoped
no
kubectl auth can-i delete namespace
Warning: resource 'namespaces' is not namespace scoped
no
kubectl auth can-i create role -n nstest1
no
kubectl auth can-i create rolebinding -n nstest1
no
kubectl auth can-i create deployment -n nstest1
```

```
yes
kubectl auth can-i create service -n nstest1
yes
kubectl auth can-i create configmap -n nstest1
yes
kubectl auth can-i create ingress -n nstest1
yes
```

Figura 5-16: comprobación del usuario actual local_u1 y sus permisos sobre el namespace nstest1

Una vez que hemos comprobado los permisos que tiene el usuario actual, y comprobando que encajan con las operaciones que queremos hacer, procedemos a la instalación de la aplicación (Figura 5-17).

```
λ kubectl apply -f aplicacion.yaml
deployment.apps/nred2 created
service/nred2 created
ingress.networking.k8s.io/nred-nred2-ingress created
```

Figura 5-17: resultado del despliegue y exposición de la aplicación de prueba nred2 en el namespace nstest1

La operación ha resultado satisfactoria, pero comprobamos igualmente que se han creado los objetos que esperábamos dentro del *namespace*, tal como vemos en la Figura 5-18.

```
λ kubectl get all -n nstest1
NAME                            READY   STATUS    RESTARTS   AGE
pod/nred1-7544d7f7c6-5sd5g      1/1     Running   0          35m
pod/nred2-765bc4566f-fftqw      1/1     Running   0          25m
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
service/nred1   ClusterIP   10.43.125.248   <none>        8080/TCP
35m
service/nred2   ClusterIP   10.43.244.34    <none>        8080/TCP
25m
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nred1   1/1     1            1           35m
deployment.apps/nred2   1/1     1            1           25m
```

```
NAME                                   DESIRED   CURRENT   READY
AGE
replicaset.apps/nred1-7544d7f7c6   1         1         1         35m
replicaset.apps/nred2-765bc4566f   1         1         1         25m
```

Figura 5-18: comprobación de que todos los objetos relacionados con la aplicación nred2 se han instalado dentro del namespace nstest1

Podemos ver en la Figura 5-18 que además de los objetos que ya existían anteriormente, se han creado los nuevos objetos relativos a la nueva aplicación.

Vamos a comprobar que la aplicación se ha expuesto correctamente fuera del clúster, y para ello debemos ir a la URL formada por el dominio de exposición con el path `/nstest1/nred2/` y ver que la aplicación se carga correctamente, lo que efectivamente es así (Figura 5-19).

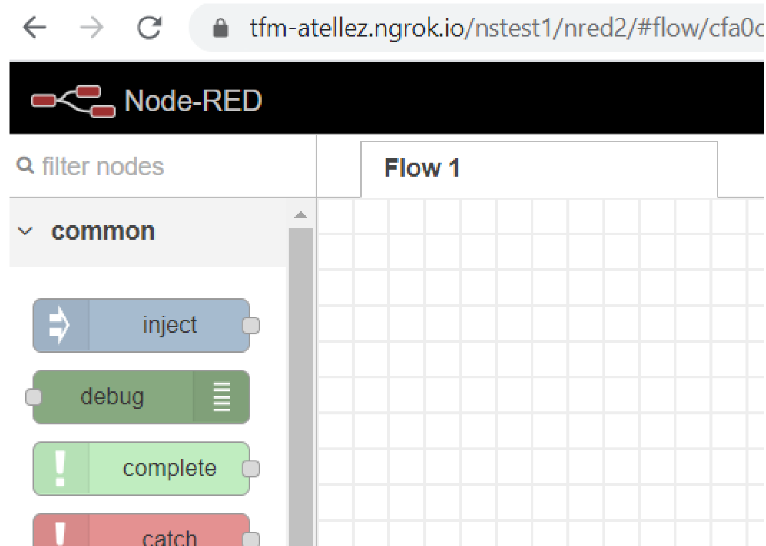


Figura 5-19: imagen de la aplicación nred2 instalada en el namespace nstest1 accediendo desde fuera del clúster

Así, podemos considerar esta prueba como satisfactoria al comprobar que un usuario con permisos sobre un *namespace* puede crear aplicaciones en dicho *namespace*.

5.4. Acceso con usuario limitado y forzar error al intentar crear un *namespace*

Para esta prueba,

- usaremos un usuario con permisos limitados (usuario **local_u1**) asociado a los grupos `alumno`, `nstest1` y `nstest2` (*namespace* aún no existente),
- forzaremos un error al intentar crear el *namespace* `nstest2`, ya que el usuario no tiene permisos para ello.

Como resultado esperamos que el usuario reciba un error al intentar crear el nuevo *namespace*.

Hay que mencionar que el *namespace* que se va a intentar crear no uno cualquiera sino uno sobre el que el usuario tiene permisos. Es decir, una vez que el *namespace* exista, el usuario será capaz de trabajar sobre él, pero puesto que no tiene permisos para crear *namespaces* nuevos debe recibir un error cuando lo intente crear.

Ya habíamos visto en la Figura 5-16 que el usuario no tiene permisos para crear un *namespace*, y al intentar crear uno vemos que se obtiene un error (Figura 5-20).

```
λ kubectl apply -f namespace_y_rol.yaml
Error from server (Forbidden): error when retrieving current configuration of:
…
Resource: "/v1, Resource=namespaces", GroupVersionKind: "/v1, Kind=Namespace"
Name: "nstest2", Namespace: ""
from server for: "yaml": namespaces "nstest2" is forbidden: User "oidc:local_u1" cannot get resource "namespaces" in API group "" in the namespace "nstest2"
Error from server (Forbidden): error when retrieving current configuration of:
…
Resource: "rbac.authorization.k8s.io/v1, Resource=roles", GroupVersionKind: "rbac.authorization.k8s.io/v1, Kind=Role"
```

```
Name: "permisos-estandar", Namespace: "nstest2"
from server for: "yaml": roles.rbac.authorization.k8s.io "permisos-estandar" is forbidden: User "oidc:local_u1" cannot get resource "roles" in API group "rbac.authorization.k8s.io" in the namespace "nstest2"
Error from server (Forbidden): error when retrieving current configuration of:
…
Resource: "rbac.authorization.k8s.io/v1, Resource=rolebindings", GroupVersionKind: "rbac.authorization.k8s.io/v1, Kind=RoleBinding"
Name: "permisos-estandar-rel", Namespace: "nstest2"
…
from server for: "yaml": rolebindings.rbac.authorization.k8s.io "permisos-estandar-rel" is forbidden: User "oidc:local_u1" cannot get resource "rolebindings" in API group "rbac.authorization.k8s.io" in the namespace "nstest2"
```

Figura 5-20: error producido al intentar crear un namespace cuando no se tiene permisos para ello

Puesto que esto era justo lo que esperábamos, esta prueba también es correcta.

5.5. Acceso con usuario limitado a un *namespace* al que no está autorizado y forzar un error

Para esta prueba,

- usaremos un usuario con permisos limitados (usuario **local_u2**) asociado a los grupos `alumno` y `nstest2`,
- forzaremos un error al intentar listar y crear una aplicación en el *namespace* `nstest1`, ya que el usuario no tiene permisos para ello.

Como resultado esperamos que el usuario reciba un error cuando intente listar las aplicaciones disponibles y cuando intente crear una nueva dentro de un *namespace* para el que no está autorizado.

Vamos a comprobar primero los permisos que tiene el usuario sobre el *namespace* sobre que el querremos trabajar, lo que podemos ver en la Figura 5-21.

```
REM consultamos cual es el usuario actual
λ kubectl config view --minify -o jsonpath='{.contexts[0].context.user}'
'local_u2'
REM comprobacion general de permisos del usuario
λ kubectl auth can-i get deployment -n nstest1
no
λ kubectl auth can-i list deployment -n nstest1
no
λ kubectl auth can-i create deployment -n nstest1
no
λ kubectl auth can-i delete deployment -n nstest1
no
λ kubectl auth can-i create service -n nstest1
no
λ kubectl auth can-i create configmap -n nstest1
no
λ kubectl auth can-i create ingress -n nstest1
no
```

Figura 5-21: comprobación del usuario actual local_u2 y sus permisos sobre el namespace nstest1

Como hemos visto en la Figura 5-21, el usuario no tiene ningún permiso sobre el *namespace* `nstest1`, por lo que cualquier cosa que queramos hacer sobre dicho *namespace* resultará en un error. Algo que podemos comprobar al intentar listar los `deployments` (Figura 5-22), pods (Figura 5-23) o directamente todos los objetos del *namespace* (Figura 5-24). O también al intentar crear una aplicación nueva.

```
λ kubectl get deployments -n nstest1
Error from server (Forbidden): deployments.apps is forbidden: User "oidc:local_u2" cannot list resource "deployments" in API group "apps" in the namespace "nstest1"
```

Figura 5-22: error obtenido por el usuario local_u2 al intentar listar los deployments del namespace nstest1, al no tener permisos

```
λ kubectl get pods -n nstest1
Error from server (Forbidden): pods is forbidden: User "oidc:local_u2" cannot list resource "pods" in API group "" in the namespace "nstest1"
```

Figura 5-23: error obtenido por el usuario local_u2 al intentar listar los pods del namespace nstest1, al no tener permisos

```
λ kubectl get all -n nstest1
Error from server (Forbidden): pods is forbidden: User "oidc:local_u2" cannot list resource "pods" in API group "" in the namespace "nstest1"
Error from server (Forbidden): replicationcontrollers is forbidden: User "oidc:local_u2" cannot list resource "replicationcontrollers" in API group "" in the namespace "nstest1"
Error from server (Forbidden): services is forbidden: User "oidc:local_u2" cannot list resource "services" in API group "" in the namespace "nstest1"
Error from server (Forbidden): daemonsets.apps is forbidden: User "oidc:local_u2" cannot list resource "daemonsets" in API group "apps" in the namespace "nstest1"
Error from server (Forbidden): deployments.apps is forbidden: User "oidc:local_u2" cannot list resource "deployments" in API group "apps" in the namespace "nstest1"
Error from server (Forbidden): replicasets.apps is forbidden: User "oidc:local_u2" cannot list resource "replicasets" in API group "apps" in the namespace "nstest1"
Error from server (Forbidden): statefulsets.apps is forbidden: User "oidc:local_u2" cannot list resource "statefulsets" in API group "apps" in the namespace "nstest1"
```

```
Error from server (Forbidden): horizontalpodautoscalers.autoscaling is forbidden: User "oidc:local_u2" cannot list resource "horizontalpodautoscalers" in API group "autoscaling" in the namespace "nstest1"
Error from server (Forbidden): jobs.batch is forbidden: User "oidc:local_u2" cannot list resource "jobs" in API group "batch" in the namespace "nstest1"
Error from server (Forbidden): cronjobs.batch is forbidden: User "oidc:local_u2" cannot list resource "cronjobs" in API group "batch" in the namespace "nstest1"
```

Figura 5-24: error obtenido por el usuario local_u2 al intentar listar todos los objetos del namespace nstest1, al no tener permisos

Por otro lado, forzamos otro error cuando el usuario intenta crear una aplicación (Figura 5-25).

```
λ kubectl apply -f aplicacion.yaml
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "apps/v1, Resource=deployments", GroupVersionKind: "apps/v1, Kind=Deployment"
Name: "nred3", Namespace: "nstest1"
…
from server for: "yaml": deployments.apps "nred3" is forbidden: User "oidc:local_u2" cannot get resource "deployments" in API group "apps" in the namespace "nstest1"
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "/v1, Resource=services", GroupVersionKind: "/v1, Kind=Service"
Name: "nred3", Namespace: "nstest1"
…
from server for: "yaml": services "nred3" is forbidden: User "oidc:local_u2" cannot get resource "services" in API group "" in the namespace "nstest1"
```

```
Error from server (Forbidden): error when retrieving current
configuration of:
Resource: "networking.k8s.io/v1beta1, Resource=ingresses",
GroupVersionKind: "networking.k8s.io/v1beta1, Kind=Ingress"
Name: "nred-nred3-ingress", Namespace: "nstest1"
…
from server for: "yaml": ingresses.networking.k8s.io "nred-
nred3-ingress" is forbidden: User "oidc:local_u2" cannot get
resource "ingresses" in API group "networking.k8s.io" in the
namespace "nstest1"
```

Figura 5-25: error obtenido por el usuario local_u2 al intentar crear una aplicación en el namespace nstest1, al no tener permisos

Con esta prueba hemos comprobado que un usuario que no está asociado al grupo que tiene permisos sobre un *namespace* en particular, no puede realizar ninguna operación sobre el mismo.

5.6. Acceso con usuario no administrador y forzar un error al intentar listar recursos de *namespaces* del sistema o de seguridad

Para esta prueba,

- usaremos un usuario con permisos limitados (usuario **local_u1**) asociado a los grupos `alumno`, `nstest1` y `nstest2`,
- forzaremos un error al intentar listar recursos de un *namespace* de sistema y del *namespace* de seguridad, ya que el usuario no tiene permisos para ello.

Como resultado esperamos que el usuario reciba un error en ambos escenarios.

Lo primero que vamos a hacer es confirmar que el usuario actual no puede realizar ninguna acción consultiva sobre ambos *namespaces* (Figura 5-26).

```
REM consultamos cual es el usuario actual
λ kubectl config view --minify -o
jsonpath='{.contexts[0].context.user}'
```

```
'local_u1'
REM comprobacion general de permisos del usuario
λ kubectl auth can-i get deployment -n kube-system
no
λ kubectl auth can-i list deployment -n kube-system
no
λ kubectl auth can-i get pod -n kube-system
no
λ kubectl auth can-i list pod -n kube-system
no
λ kubectl auth can-i get secret -n kube-system
no
λ kubectl auth can-i list secret -n kube-system
no
λ kubectl auth can-i get deployment -n seguridad
no
λ kubectl auth can-i list deployment -n seguridad
no
λ kubectl auth can-i get pod -n seguridad
no
λ kubectl auth can-i list pod -n seguridad
no
λ kubectl auth can-i get secret -n seguridad
no
λ kubectl auth can-i list secret -n seguridad
no
```

Figura 5-26: comprobación del usuario actual local_u1 y sus permisos sobre los namespaces kube-system y seguridad

Con lo visto en la Figura 5-26 ya tenemos la certeza de que el usuario no puede realizar acciones sobre ninguno de los *namespaces*, pero aún así vamos a intentar obtener algunos de los recursos que están almacenados en dichos *namespaces* y comprobar que se producen los correspondientes errores.

```
λ kubectl get pods -n kube-system
Error from server (Forbidden): pods is forbidden: User "oidc:local_u1" cannot list resource "pods" in API group "" in the namespace "kube-system"
λ kubectl get deployments -n kube-system
Error from server (Forbidden): deployments.apps is forbidden: User "oidc:local_u1" cannot list resource "deployments" in API group "apps" in the namespace "kube-system"
λ kubectl get secrets -n kube-system
Error from server (Forbidden): secrets is forbidden: User "oidc:local_u1" cannot list resource "secrets" in API group "" in the namespace "kube-system"
λ kubectl get pods -n seguridad
Error from server (Forbidden): pods is forbidden: User "oidc:local_u1" cannot list resource "pods" in API group "" in the namespace "seguridad"
λ kubectl get deployments -n seguridad
Error from server (Forbidden): deployments.apps is forbidden: User "oidc:local_u1" cannot list resource "deployments" in API group "apps" in the namespace "seguridad"
λ kubectl get secrets -n seguridad
Error from server (Forbidden): secrets is forbidden: User "oidc:local_u1" cannot list resource "secrets" in API group "" in the namespace "seguridad"
```

Figura 5-27: errores obtenidos por el usuario local_u1 al intentar recuperar objetos de los namespaces kube-system y seguridad

Se puede ver en la Figura 5-27 que el usuario recibe errores cuando intenta recuperar el listado de objetos de los *namespaces* considerados protegidos. En realidad, esta prueba es equivalente a la realizada en el apartado 5.5, ya que en ambos casos se trata de un usuario sin acceso a ninguna operación dentro de un *namespace* intentando realizar alguna operación en el mismo.

5.7. Acceso con un usuario con acceso al grupo general asociado a su dominio de email, que se corresponde con un grupo de solo lectura sobre un *namespace*

Para esta prueba,

- usaremos un usuario (usuario **local_u3**) que no está asociado a ningún grupo por su id de usuario, pero sí lo está asociado al grupo genérico `alumno`, que se deriva del dominio de su dirección de email,
- listaremos los objetos creado del *namespace* `nstest2`, sobre el que el grupo `alumno` tiene permisos de lectura,
- intentaremos crear un objeto en dicho *namespace*.

Como resultado esperamos que el usuario pueda listar los objetos correctamente, ya que el único grupo al que tiene acceso tiene permisos de lectura, pero reciba un error cuando intente crear algo en el *namespace*.

En la Figura 5-28 podemos ver algunos de los permisos que tiene el usuario, y podemos ver que el usuario puede listar objetos, pero no puede crear nada dentro del *namespace* `nstest2`.

```
REM consultamos cual es el usuario actual
λ kubectl config view --minify -o
jsonpath='{.contexts[0].context.user}'
'local_u3'
REM comprobacion general de permisos del usuario
λ kubectl auth can-i get deployment -n nstest2
yes
λ kubectl auth can-i list deployment -n nstest2
yes
λ kubectl auth can-i create deployment -n nstest2
no
λ kubectl auth can-i delete deployment -n nstest2
no
λ kubectl auth can-i create service -n nstest2
no
```

```
λ kubectl auth can-i create configmap -n nstest2
no
λ kubectl auth can-i create ingress -n nstest2
no
```

Figura 5-28: comprobación del usuario actual local_u3 y sus permisos sobre el namespace nstest2

Tras la comprobación de permisos, vamos a ver que efectivamente podemos listar los elementos del namespace (Figura 5-29), pero no podemos crear ningún objeto dentro del mismo (Figura 5-30).

```
λ kubectl get all -n nstest2
NAME                              READY   STATUS    RESTARTS   AGE
pod/nred-2-1-5694978684-fzq5l     1/1     Running   2          57m
NAME                  TYPE         CLUSTER-IP      EXTERNAL-IP
PORT(S)    AGE
service/nred-2-1      ClusterIP    10.43.88.255      <none>
8080/TCP   57m
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nred-2-1      1/1     1            1           57m
NAME                                         DESIRED   CURRENT   READY
AGE
replicaset.apps/nred-2-1-5694978684    1          1          1
57m
```

Figura 5-29: listado de objetos del namespace nstest2, recuperados correctamente por el usuario local_u3

```
λ kubectl create secret generic misecreto --from-literal=dato1=secreto1 --from-literal=dato2=secreto2 -n nstest2
Error from server (Forbidden): secrets is forbidden: User "oidc:local_u3" cannot create resource "secrets" in API group "" in the namespace "nstest2"
```

Figura 5-30: error recibido al intentar crear un objeto en el namespace nstest2, por parte del usuario local_u3

5.8. Monitorización en las pruebas relacionadas con los accesos al clúster

De cara a la monitorización de las pruebas descritas en los apartados anteriores hay que decir que han sido pruebas que no han generado ningún problema y por tanto no se han

producido alertas. Sin embargo, podemos encontrar información sobre las nuevas aplicaciones en Prometheus y, por tanto, en Grafana. El *dashboard* que tenemos ahí (ver apartado 4.3.3) se enfoca al clúster en sí, pero podemos ver cierta información a nivel de pod, y esta información, relativo a las aplicaciones que hemos idea creando a lo largo de las pruebas anteriores se puede observar en la Figura 5-31 y Figura 5-32.

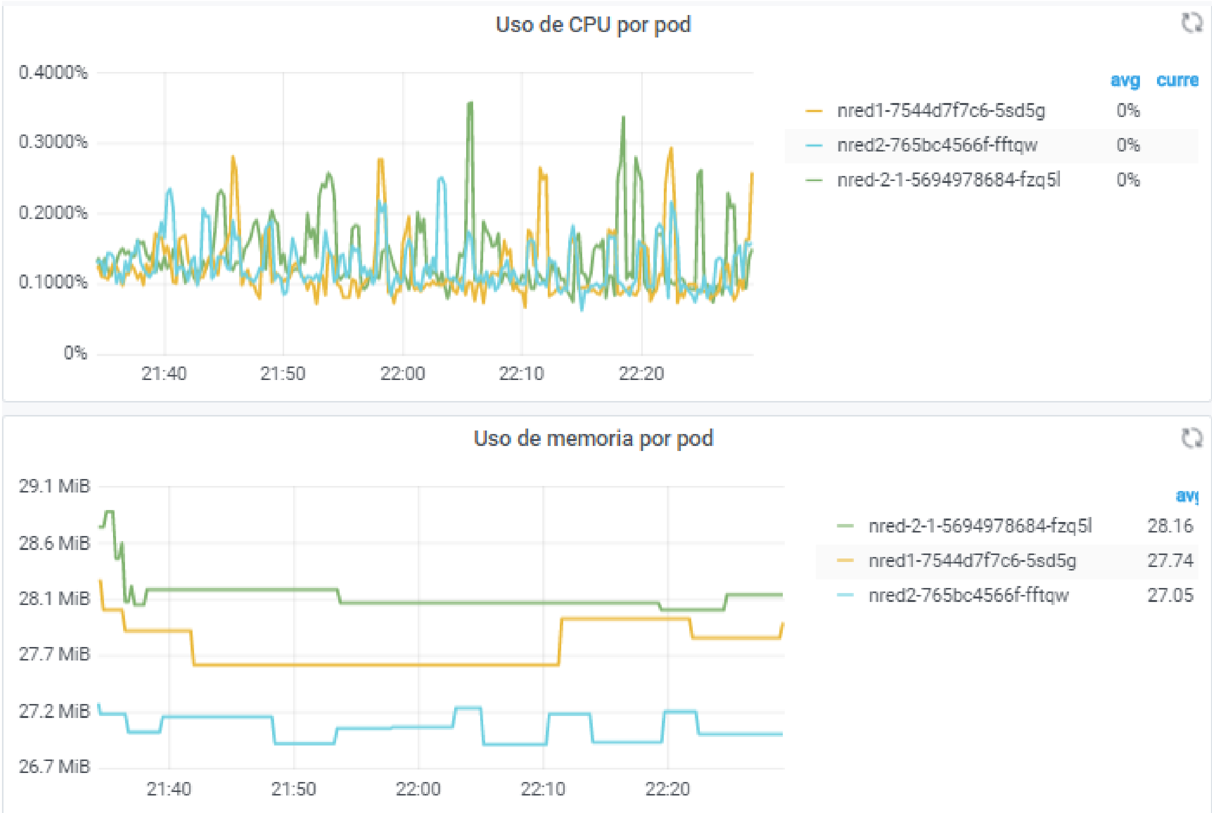


Figura 5-31: paneles del dashboard de Grafana que muestran el uso de CPU y memoria por parte de los pods de las aplicaciones creadas para las pruebas realizadas

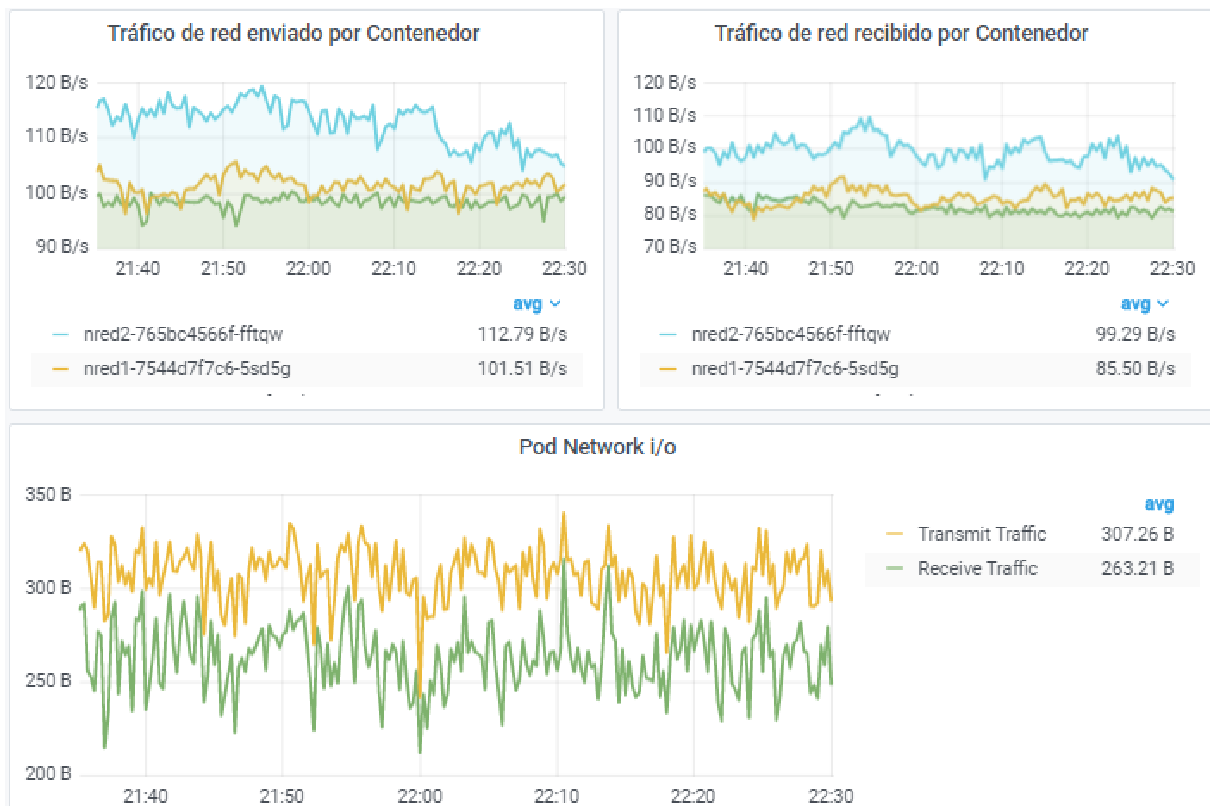


Figura 5-32: paneles del dashboard de Grafana que muestran el tráfico de red por parte de los pods y sus contenedores de las aplicaciones creadas para las pruebas realizadas

5.9. Resumen del capítulo

En este capítulo hemos detallado las operaciones que hemos realizamos en el clúster, mostrando cómo creamos los diferentes objetos que usaremos de cara a las pruebas. Estas plantillas mostradas que se han usado para las pruebas también pueden ser utilizadas a futuro para la creación de los diferentes objetos. También se ha explicado la relación entre los diferentes objetos creados y para qué son necesarios.

Después hemos visto diferentes pruebas realizadas en relación con la seguridad con respecto a los accesos por parte de diferentes usuarios que tienen diferentes roles, y comprobar que las operaciones a las que están autorizados son las únicas que realmente puede realizar. Como parte adicional a estas pruebas, también hemos comprobado que la creación de una aplicación configurada con exposición hacia fuera del clúster funciona correctamente sin necesidad de hacer nada más, por lo que es posible incluir esta configuración en las aplicaciones futuras que se consideren y dar la posibilidad de un

autoservicio completo a los usuarios del clúster una vez que estén autorizados en los *namespaces* que correspondan.

Por último, hemos visto algunas de las métricas relacionadas con las aplicaciones creadas, a través de Grafana, demostrando que desde el momento en que realizamos las acciones que sean podemos acceder a los datos métricos que se vayan generando.

6. Conclusiones

6.1. Logros alcanzados

Durante el desarrollo de este trabajo se ha buscado implementar mecanismos de seguridad que permitan una gestión sencilla de la autenticación y autorización, así como de la instalación y configuración de mecanismos que permitan monitorizar el clúster. Así pues, estas dos han sido las líneas principales de trabajo, y en las que mayores resultados se han obtenido. Sin embargo, durante el trabajo en estas líneas han surgido otros objetivos secundarios que han culminado como logros dignos a mencionar, ya que pueden ser de utilidad en el futuro de este tipo de clústeres o en futuras líneas de trabajo.

El primer logro conseguido ha sido la preparación del clúster de Raspberry. Partiendo de trabajos anteriores del equipo del proyecto eNMoLabs, hemos extendido la preparación inicial para incluir la exposición del clúster y con la posibilidad de trabajar con el clúster desde fuera del mismo. De esta forma, el clúster en sí ha quedado como servidor remoto al que no es necesario acceder para nada, ni para el mantenimiento de objetos ni para acceder a las herramientas instaladas.

Tras ello, uno de los logros principales ha sido el desarrollo de la aplicación OIDC con la que hemos conseguido poder gestionar la autenticación con las credenciales de la universidad y poder configurar roles de manera sencilla, con lo que conseguimos poder gestionar el clúster de kubernetes con las credenciales de la universidad y además poder controlar los permisos que queramos dar a los diferentes usuarios de estos. El sistema ideado de roles se basa en grupos con nombres coincidentes a los *namespaces* creados o por crear, haciendo más intuitivo la labor de asignar y detectar sobre qué *namespaces* tiene permisos un usuario dado.

Además, con el desarrollo de la aplicación cliente OIDC conseguimos que el proceso de autenticación sea muy sencillo y transparente para quien se está autenticando, ya que tan solo será necesario lanzar un ejecutable e introducir las credenciales de la universidad una vez que se requieran, y será el cliente OIDC desarrollado el que se encargue de envolver la invocación de la aplicación OIDC así como su respuesta y actualización de configuración del clúster en el ordenador en el que se esté realizando el proceso.

De esta forma, aquel que desee y esté autorizado para ello, podrá trabajar con el clúster y empezar a hacerlo de forma muy sencilla. Y desde el lado del administrador del clúster, la gestión de quién hace qué se ve resumida a un fichero de configuración dentro del clúster, y al conjunto de roles definidos, y que pueden ser extendidos en caso de que surjan nuevos casos en el futuro.

Una vez logrado la autenticación con credenciales de UNED y su integración en el clúster, nos enfocamos en preparar una monitorización de aspectos técnicos principalmente para el clúster y así hemos preparado los ficheros de instalación y configuración de Prometheus, su *alert manager* y de Grafana. En los dos últimos casos dedicamos mayor tiempo en la confección de alertas y del *dashboard* que pudieran servir de utilidad teniendo en cuenta las características del clúster creado.

Estas aplicaciones, junto con el *dashboard* de kubernetes y la aplicación OIDC, se han expuesto hacia fuera del clúster como prueba de concepto de cómo podrían quedar en una instalación de clúster real que se quisiera llevar a cabo. Durante el desarrollo del presente trabajo hemos accedido a estas herramientas desde fuera, sin acceder al clúster de forma directa para nada en la correcta operación del mismo (para resolución de problemas técnicos aparecidos sí ha sido necesario acceder a algunos nodos directamente), lo que es esperable en el funcionamiento futuro del clúster.

Cabe destacar también que la imagen Docker de la aplicación OIDC se ha dispuesto para que sea usada en arquitectura ARM aun cuando la aplicación se ha desarrollado en otras arquitecturas. Esto es muy interesante ya que puede permitir realizar cualquier desarrollo que sea necesario y construir una imagen Docker que pueda ser usada directamente en el clúster, o recompilar imágenes ya existentes, como el caso de las alertas para MS Teams, para otras arquitecturas para que sea posible usarlas en ARM.

Por último, hemos puesto a prueba el clúster resultante obteniendo los resultados previstos y comprobando que el clúster está preparado para su uso y aprovechando las ventajas que ofrece kubernetes.

6.2. Trabajos futuros

Los trabajos futuros que pueden surgir como continuación de este trabajo o a partir del mismo se pueden agrupar en diferentes líneas de trabajo, que por claridad vamos a detallar por separado.

6.2.1. Integración del trabajo realizado en el proyecto eNMoLabs

Todo el trabajo realizado se ha hecho sobre un clúster local fuera del proyecto eNMoLabs, por lo que debe integrarse dentro de un clúster del proyecto.

Pensando en simplificar al máximo la integración se ha preparado un conjunto de ficheros YAML que definen todos los objetos y sus configuraciones necesarias, por lo que la integración de cada una de partes aquí hecha se resume a una carga de los ficheros YAML correspondiente a cada pieza, dentro del clúster final. Pero antes de esa carga de ficheros, es necesario adaptar ciertos parámetros de varias de las configuraciones. Parámetros como la URL en la que las herramientas estarán expuestas, o la configuración de grupos asignados a nivel de usuario y de email, etc. Por tanto, aunque la integración dentro del proyecto debería ser sencilla, requerirá de una adaptación de lo realizado al clúster destino.

También deben ser consideradas las adaptaciones relacionadas con la monitorización, como detallaremos en el punto 6.2.3, pero estas adaptaciones no son imprescindibles para poder realizar la instalación e integración en el proyecto eNMoLabs.

6.2.2. Mejoras en la aplicación OIDC

La aplicación OIDC aquí desarrollada se ha hecho pensando en un uso exclusivo para el clúster donde va a ser instalado y con un control de acceso a los objetos kubernetes asociados muy restringido. Con eso en mente, hay algunos puntos que no se han desarrollado como parte de la aplicación debido al coste en tiempo que requieren y el poco beneficio que ofrecen en el contexto indicado. Sin embargo, vamos a comentar estos aspectos potencialmente mejorables por si una vez integrado en el clúster final se viese la necesidad de llevar a cabo alguna de estas mejoras.

Externalizar las claves usadas para la generación de credenciales a un almacén seguro. Si la aplicación OIDC se va a instalar una sola vez y ésta está suficientemente protegida, el disponer de las claves dentro de la aplicación en sí podría ser suficiente. Sin embargo, esto no es recomendable ya que hacerlo así conlleva unos riesgos de seguridad como puede ser el que no permite una rotación sencilla de claves, o que un acceso no deseado al repositorio del código o a la imagen Docker podría exponer dichas claves. Es por este motivo por lo que es altamente recomendable usar algún almacén seguro que pueda estar disponible dentro del proyecto eNMoLabs, como una instancia de Vault, y al que la aplicación OIDC se pueda conectar en el arranque de la aplicación para recuperar las claves que correspondan. Además, este tipo de almacenes suelen dar la opción a rotar las claves cada cierto tiempo, lo que también es recomendable por motivos de seguridad (si alguien no autorizado se hace con las claves en un momento dado, sólo podrá hacer un uso ilícito de las mismas hasta que estas se renueven).

La aplicación desarrollada genera logs con información de lo que va ocurriendo, con diferentes niveles de log que pueden ser configurados. Pero no expone ningún tipo de métrica que pueda ser explotada por Prometheus. Las métricas relativas al consumo de memoria o CPU no son necesario que se expongan porque kubernetes las puede deducir por otras vías, pero sí podría ser interesante publicar métricas relativas a su uso, como podría ser métricas relacionadas con los accesos correctos e incorrectos. En caso de que estas métricas se expusiesen, podrían ser recogidas por Prometheus y ser explotadas por Grafana y Alert manager según se considerase. La aplicación ha sido desarrollada en JavaScript sobre NodeJS y ya existen librerías como (SimenB, 2020) o (Yeh, 2019) que podrían ser integradas dentro de la aplicación y exportar las métricas que se puedan consideran. Antes de considerar esta opción, o a la vez, se debe considerar también la posibilidad de explotar los logs ya generados por medio de una solución que permita una presentación y gestión de alertas basadas en la información de los logs, como Kibana. Si se optase por este otro enfoque, no se podría integrar ninguna información finalmente en Grafana, pero es posible que ya se pudiesen cubrir las posibilidades de detección de autenticaciones correctas/incorrectas y la generación de alertas cuando pudiese corresponder.

Otro detalle menor que podría ser mejorable en la aplicación es la posibilidad de hacer la aplicación complemente *stateless*. En la implementación actual, tras la finalización de un proceso correcto, la sesión recién creada es válida por un tiempo de 24 horas. Durante este período, si el mismo usuario accede de nuevo a la aplicación OIDC, debe ir a la misma instancia de la aplicación para recuperar correctamente los grupos a los que pertenece el usuario, ya que esta información se guarda en la memoria del pod. En realidad, esto no es un problema ya que lo hacemos por medio de la afinidad que es configurable en la aplicación, y así nos aseguramos de que siempre sea así. Pero podría mejorarse la aplicación y evitar guardar nada en memoria, haciendo uso de la instancia de base de datos ya usadas para las sesiones en sí. Este seguramente sea un detalle menor ya que es un problema que está cubierto actualmente y para una casuística que no debería ser muy frecuente, pero aún así, se podría considerar a futuro con el fin de intentar mejorar la aplicación.

6.2.3. Monitorización

En relación con la monitorización se debe realizar un refinado de las alertas configuradas en Prometheus y la información mostrada en Grafana. Este trabajo se debe hacer cuando el clúster ya esté en uso, y debe ser un trabajo iterativo y evolutivo en base a las necesidades que haya y vayan surgiendo durante la explotación del clúster, y que por tanto no puede definirse en este momento.

Por otro lado, y como se ha mencionado para la aplicación OIDC, se puede considerar el forzar que las aplicaciones que se desplieguen dentro del clúster deban exportar información en forma de métricas que Prometheus pueda explotar. Sin embargo, esto hay que analizarlo con detenimiento, ya que las métricas exportadas deben aportar valor y es más interesante tener calidad en las métricas disponibles, aunque haya menos cantidad de estas.

También podría ser interesante integrar otras fuentes de métricas relacionadas con las Raspberry en sí, por ejemplo. O si se instalan sensores y/o actuadores, también podría ser interesante contar con métricas sobre el uso que se haga de ellos.

En general, cuantas más métricas haya cubriendo el máximo de información, mejor. Pero también hay que tener cuidado en el sentido de que un mayor volumen de fuentes supone un mayor tráfico de red, y un mayor volumen de datos supone una sobrecarga de memoria y CPU.

Según se vayan definiendo las fuentes de métricas para Prometheus se deberá ir añadiendo/modificando/eliminando paneles en Grafana. También hay que tener cuidado con añadir demasiado paneles ya que podríamos llegar a una sobreinformación que deje de ser útil. En el caso de que se llegue a un caso de sobreinformación se puede optar por crear más *dashboards*, dejando cada uno de ellos para un uso en concreto (por ejemplo, uno solo para el clúster, otro para sensores/actuadores, otro para el estado de Raspberry, etc.).

6.2.4. Logging

Una de las mayores carencias que el clúster generado tiene es el no disponer de herramientas que mejoren la gestión de los logs generados en los diferentes contenedores. Esto queda por tanto como un posible, aunque muy recomendable, trabajo futuro.

A continuación, se menciona una posible configuración de solución para cubrir esta necesidad, aunque si se desea llevar a cabo seguramente sea necesario disponer de Raspberry con mayor capacidad de memoria que las utilizadas en este trabajo, ya que para un correcto funcionamiento de esta solución consume bastantes recursos y requiere de pods de un mayor consumo de memoria que hacía inviable su uso con los dispositivos de los que hemos dispuesto.

La posible solución ya se había mencionado en el apartado 3.1, y puede entender más en detalle en (Jetha, 2020). La idea sería usar FluentD para capturar los logs generados en cada contenedor y enviarlos a Elasticsearch, donde se indexarían y quedarían a disposición de que una aplicación de presentación, como Kibana, realice consultas y muestre los resultados en el formato visual que se considere. Con esta solución, tendríamos todos los logs de todas las aplicaciones disponibles en una misma ubicación. Y además se podría dividir dicha ubicación en espacios virtuales de tal manera que algunos usuarios puedan tener acceso a todos los

espacios (usuarios administradores) y otros usuarios solo puedan tener acceso a un conjunto reducido de espacios.

Además, si se disponen de estas herramientas, se deberían extender con una herramienta como Kibana alerting, para ofrecer la posibilidad de generación de alertas en base a información contenida en los logs.

Llevar a cabo este trabajo aumentaría mucho las capacidades de análisis de problemas y detección de anomalías, así como permitiría una gestión de alertas refinada a nivel de log, que, junto con las alertas basadas en métricas ya tratadas en el este trabajo, otorgaría grandes capacidades de detección de anomalías de todo tipo.

6.2.5. Ideas generales

Como idea general ya habíamos comentado en el apartado 4.1 la posibilidad de preparar un auto escalado de aplicaciones basado en criterio de tiempo, es decir, en una planificación que a ciertas horas de ciertos días levantase o incrementase las réplicas que correspondan, y que fuera de ese tiempo reduzca las réplicas o incluso que inhabilite la aplicación por completo, evitando así que consuman recursos. Esto puede ser muy útil para clústeres de reducidas dimensiones, como el nuestro, y algo factible si pensamos en aplicaciones como prácticas para alumnos que puedan planificarse. Esto permitiría ahorrar recursos a coste de disponibilidad y, potencialmente, podría permitir disponer de más aplicaciones configuradas a la vez en el clúster, aunque solo en ejecución las que correspondan a cada momento. Puede haber varias maneras de enfocar esta idea, pero una que puede ser relativamente sencilla de implementar y de configurar es la que está disponible en (AliyunContainerService, 2020).

Por otro lado, también habíamos comentado en el apartado 4.1 que k3s no permite actualmente disponer de más de un nodo máster. Esto es un potencial riesgo ya que en caso de que dicho nodo se estropee o simplemente tenga un problema que haga que no funcione temporalmente, el clúster completo estará inaccesible e inoperable. Mientras esta situación se mantenga y el clúster cuente con un único nodo máster sería muy aconsejable que se

preparase otra herramienta completamente ajena al clúster en sí que chequease cada poco tiempo que el clúster está accesible y el nodo máster, disponible. Una posible solución podría pasar por un proceso sencillo desplegado en todos los nodos esclavos que cada poco tiempo (por ejemplo, un minuto) hiciesen una petición a un proceso interno corriendo en la Raspberry máster, pero fuera del clúster, con el fin de verificar que está accesible. En caso de que alguno de los nodos detecte un error podría elevar una alerta en forma de email o lo que se configure para ello. Esto procesos deben estar completamente fuera del clúster, pero podrían correr como procesos sencillos directamente en los nodos, y deberían ser procesos ligeros y no accesibles fuera de los propios nodos con una comunicación entre ellos por las IPs definidas dentro de la red local. De esta forma, si el nodo máster se cae, en un máximo del tiempo configurado debería ser detectado por el primero nodo esclavo que realice la comprobación, que a su vez elevaría la alerta como se define y el administrador del clúster podría analizar el problema.

Referencias y bibliografía

AliyunContainerService. (2020). *kubernetes-cronhpa-controller*. Recuperado el 18 de septiembre de 2020, de https://github.com/AliyunContainerService/kubernetes-cronhpa-controller

Al-Zoubi, A. H. (2014). Remote laboratories for renewable energy courses at Jordan universities. *IEEE Frontiers in Education Conference*, (págs. 1-4).

askubuntu.com. (05 de octubre de 2012). *How to install updates via command line?* Recuperado el 18 de septiembre de 2020, de https://askubuntu.com/questions/196768/how-to-install-updates-via-command-line

autenticacion, k. (08 de 2020). *Autenticación*. Obtenido de https://kubernetes.io/docs/reference/access-authn-authz/authentication/

Data, T. S. (2020). *Raft, Understandable Distributed Consensus*. Recuperado el 13 de septiembre de 2020, de http://thesecretlivesofdata.com/raft/

Docker. (30 de abril de 2019). *Building Multi-Arch Images for Arm and x86 with Docker Desktop*. Recuperado el 11 de septiembre de 2020, de https://www.docker.com/blog/multi-arch-images/

Docker. (2020). *Get Docker*. Recuperado el 19 de septiembre de 2020, de https://docs.docker.com/get-docker/

Docker. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://www.docker.com/

Docker. (2020). *Swarm mode overview*. Recuperado el 13 de septiembre de 2020, de https://docs.docker.com/engine/swarm/

Docker. (2020). *What is a Container?* Recuperado el 13 de septiembre de 2020, de https://www.docker.com/resources/what-container

Eduardo, C. (2020). *kubernetes-cluster-dashboard.json*. Recuperado el 14 de septiembre de 2020, de https://github.com/carlosedp/cluster-monitoring/blob/7d2b7473d8aa10219a7843c5e7305a25f76cddcf/grafana-dashboards/kubernetes-cluster-dashboard.json

Elastic. (2020). *Página web oficial*. Recuperado el 17 de septiembre de 2020, de https://www.elastic.co/es/elastic-stack

eNMoLabs. (2020). *Proyecto eNMoLabs*. Recuperado el 11 de septiembre de 2020, de https://blogs.uned.es/cibergid/enmolabs/

eNMoLabs, P. (julio de 2019). *Entregable 2.1 (E2.1)*. Recuperado el 11 de septiembre de 2020, de http://blogs.uned.es/cibergid/wp-content/uploads/sites/233/2020/08/Entregable_2.1.pdf

eNMoLabs, P. (enero de 2020). *Entregable 2.2*. Recuperado el 12 de septiembre de 2020, de http://blogs.uned.es/cibergid/wp-content/uploads/sites/233/2020/08/Entregable_2.2.pdf

eNMoLabs, P. (septiembre de 2020). *Entregable 3.1*. Recuperado el 12 de septiembre de 2020, de http://blogs.uned.es/cibergid/wp-content/uploads/sites/233/2020/09/Entregable-3.1-Objetivo-4-trimestre-4-It1.pdf

Fluentd. (2020). *Página web oficial*. Recuperado el 17 de septiembre de 2020, de https://www.fluentd.org/

Grafana. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://grafana.com/

Herman, M. (20 de 01 de 2019). *Logging in Kubernetes with Elasticsearch, Kibana, and Fluentd*. Recuperado el 17 de septiembre de 2020, de https://mherman.org/blog/logging-in-kubernetes-with-elasticsearch-Kibana-fluentd/

IETF. (08 de 2020). *RFC7519*. Obtenido de https://tools.ietf.org/html/rfc7519

Jetha, H. (30 de marzo de 2020). *How To Set Up an Elasticsearch, Fluentd and Kibana (EFK) Logging Stack on Kubernetes*. Recuperado el 17 de septiembre de 2020, de https://www.digitalocean.com/community/tutorials/how-to-set-up-an-elasticsearch-fluentd-and-kibana-efk-logging-stack-on-kubernetes

k3s. (diciembre de 2019). *K3s nodes Status:NotReady after rebooting*. Recuperado el 19 de septiembre de 2020, de https://github.com/rancher/k3s/issues/1170

k3s. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de Lightweight Kubernetes: https://k3s.io/

Kubernetes. (2020). *Autenticación en kubernetes*. Recuperado el 11 de septiembre de 2020, de https://kubernetes.io/docs/reference/access-authn-authz/authentication/

Kubernetes. (2020). *Ingress*. Recuperado el 14 de septiembre de 2020, de https://kubernetes.io/docs/concepts/services-networking/ingress/

Kubernetes. (2020). *JSONPath Support*. Recuperado el 18 de septiembre de 2020, de https://kubernetes.io/docs/reference/kubectl/jsonpath/

Kubernetes. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://kubernetes.io/es/

Kubernetes. (2020). *Web UI (Dashboard)*. Recuperado el 11 de septiembre de 2020, de https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

Mackenzie. (10 de febrero de 2020). *What is K3s?* Recuperado el 13 de septiembre de 2020, de https://rootleveltech.com/what-is-k3s/

Mangat, M. (04 de 11 de 2019). *Kubernetes vs Docker Swarm: What are the Differences?* Recuperado el 13 de septiembre de 2020, de https://phoenixnap.com/blog/kubernetes-vs-docker-swarm

Mathias Renner, L. K. (11 de enero de 2017). *Setup Kubernetes on a Raspberry Pi Cluster easily the official way!* Recuperado el 13 de septiembre de 2020, de https://blog.hypriot.com/post/setup-kubernetes-raspberry-pi-cluster/

Michael Hausenblas, L. R. (2018). *Kubernetes Security.* O'Reilly Media, Inc.

Ngrok. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://ngrok.com/

node-oidc-provider. (2020). *node-oidc-provider*. Recuperado el 13 de septiembre de 2020, de https://github.com/panva/node-oidc-provider

Node-RED. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://nodered.org/

OIDC. (2020). *OIDC*. Recuperado el 11 de septiembre de 2020, de The Internet Identity Layer: https://openid.net/connect/

Osnat, R. (15 de enero de 2019). *Protecting Kubernetes Secrets: A Practical Guide*. Recuperado el 18 de septiembre de 2020, de https://blog.aquasec.com/managing-kubernetes-secrets

Pastor, R. T.-G. (05 de julio de 2020). *A WoT Platform for Supporting Full-Cycle IoT Solutions from Edge to Cloud Infrastructures: A Practical Case*. Recuperado el 12 de septiembre de 2020, de https://www.mdpi.com/1424-8220/20/13/3770

Piltch, A. (25 de junio de 2020). *How to Set Up a Headless Raspberry Pi, Without Ever Attaching a Monitor*. Recuperado el 18 de septiembre de 2020, de https://www.tomshardware.com/reviews/raspberry-pi-headless-setup-how-to,6028.html

prat0318. (2020). *1. Kubernetes Deployment Statefulset Daemonset metrics*. Recuperado el 14 de septiembre de 2020, de https://grafana.com/grafana/dashboards/8588

Prometheus. (2020). *Alertmanager, página web oficial*. Recuperado el 11 de septiembre de 2020, de https://prometheus.io/docs/alerting/latest/alertmanager/

Prometheus. (2020). *Página web oficial*. Recuperado el 11 de septiembre de 2020, de https://prometheus.io/

Raspberry Pi. (2020). *Downloads*. Recuperado el 18 de septiembre de 2020, de https://www.raspberrypi.org/downloads/

Redis. (2020). *Página web oficial*. Recuperado el 13 de septiembre de 2020, de https://redis.io/

SimenB. (2020). *prom-client*. Recuperado el 18 de septiembre de 2020, de https://github.com/siimon/prom-client

Strittmatter, B. (30 de mayo de 2018). *Understanding Etcd Consensus and How to Recover from Failure*. Recuperado el 13 de septiembre de 2020, de https://blog.containership.io/etcd/

Tobarra, L. R.-G. (2019). *Web of Things Platforms for Distance Learning Scenarios in Computer Science Disciplines: A Practical Approach*. Recuperado el 12 de septiembre de 2020, de Technologies, vol. 7, no 1, pp. 1-20. Special Issue "Technology Advances on IoT Learning and Teaching": https://www.mdpi.com/2227-7080/7/1/17/htm

Vallim, M. (26 de junio de 2019). *Kubernetes Journey — Up and running out of the cloud — etcd*. Recuperado el 13 de septiembre de 2020, de https://itnext.io/kubernetes-journey-up-and-running-out-of-the-cloud-etcd-b332d1be474c

W3C. (09 de abril de 2020). *Web of Things (WoT) Architecture*. Recuperado el 12 de septiembre de 2020, de https://www.w3.org/TR/wot-architecture/

Wilson, B. (04 de noviembre de 2019). *How To Setup Grafana On Kubernetes*. Recuperado el 14 de septiembre de 2020, de https://devopscube.com/setup-grafana-kubernetes/

Wilson, B. (02 de octubre de 2019). *How to Setup Prometheus Monitoring On Kubernetes Cluster*. Obtenido de Prometheus Monitoring on Kubernetes: https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/

Yeh, J. (14 de diciembre de 2019). *Node.js Monitoring with Prometheus+Grafana*. Recuperado el 18 de septiembre de 2020, de https://medium.com/teamzerolabs/node-js-monitoring-with-prometheus-grafana-3056362ccb80

Anexo

A. Instalación del clúster

En este anexo vamos a detallar los pasos que hemos seguido para la instalación inicial del clúster, desde la preparación de las tarjetas de memoria hasta la instalación de k3s tanto en la Raspberry máster como en las *worker*. Aunque inicialmente nos basamos en un documento interno del Proyecto eNMoLabs, hemos hecho varias cosas diferentes y consideramos que merece la pena indicar los pasos seguidos en este trabajo, por si pueden ser de utilidad a futuro en la preparación de otros clústeres.

Primeramente, debemos preparar las tarjetas de memoria microSD que usaremos en las Raspberry. Para ello, debemos formatearlas y después instalar el sistema operativo *Raspberry Pi OS* (anteriormente llamado *Raspbian*) en cada una de ellas. Seguramente la forma más sencilla de hacer ambos pasos sea por medio de la herramienta *Raspberry Pi Imager* que Raspberry ha puesto a disposición y que puede ser descargada desde (Raspberry Pi, 2020). Esta herramienta es muy sencilla de usar y muy intuitiva (Figura A-1). Basta con seleccionar el sistema operativo (Figura A-2), la tarjeta de memoria que queremos usar (Figura A-3), iniciar y esperar a que el proceso termine (Figura A-4) y asegurarnos de que el proceso ha terminado correctamente (Figura A-5).

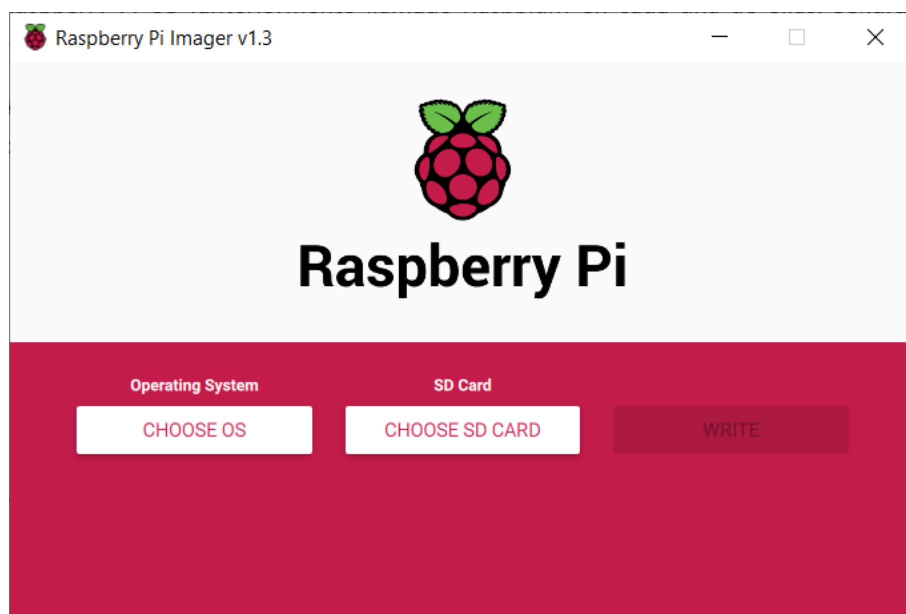


Figura A-1: Imagen inicial de la herramienta Raspberry Pi Imager para Windows

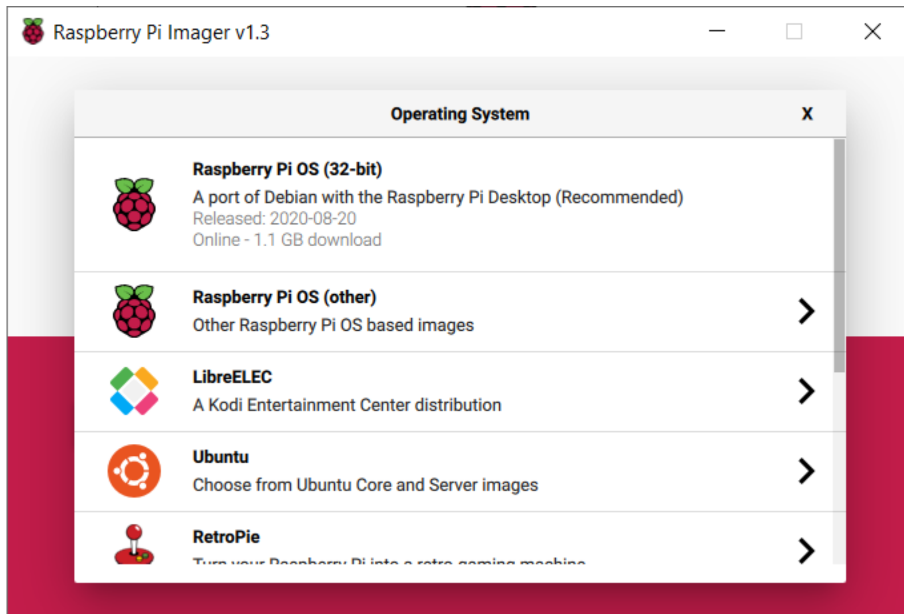


Figura A-2: algunos de los sistemas operativos que podemos instalar en la tarjeta SD con la herramienta Raspberry Pi Imager

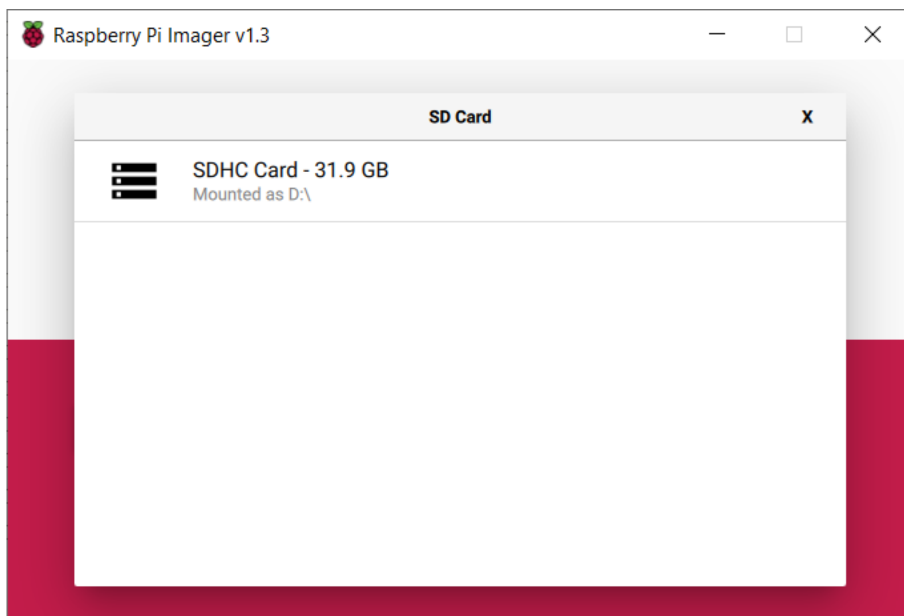


Figura A-3: selección de tarjeta SD que queremos formatear y donde instalaremos el sistema operativo elegido con la herramienta Raspberry Pi Imager

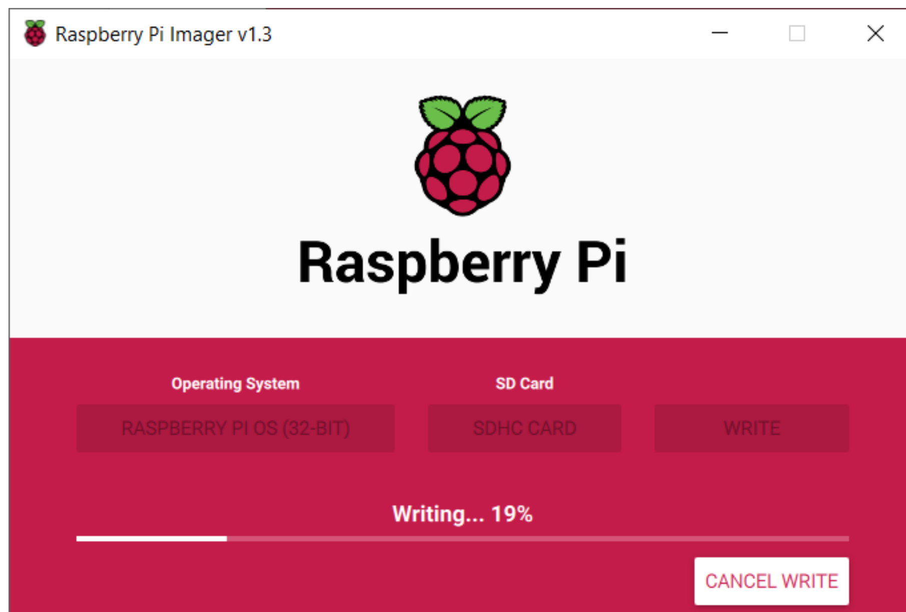


Figura A-4: proceso de formateo e instalación en curso con la herramienta Raspberry Pi Imager

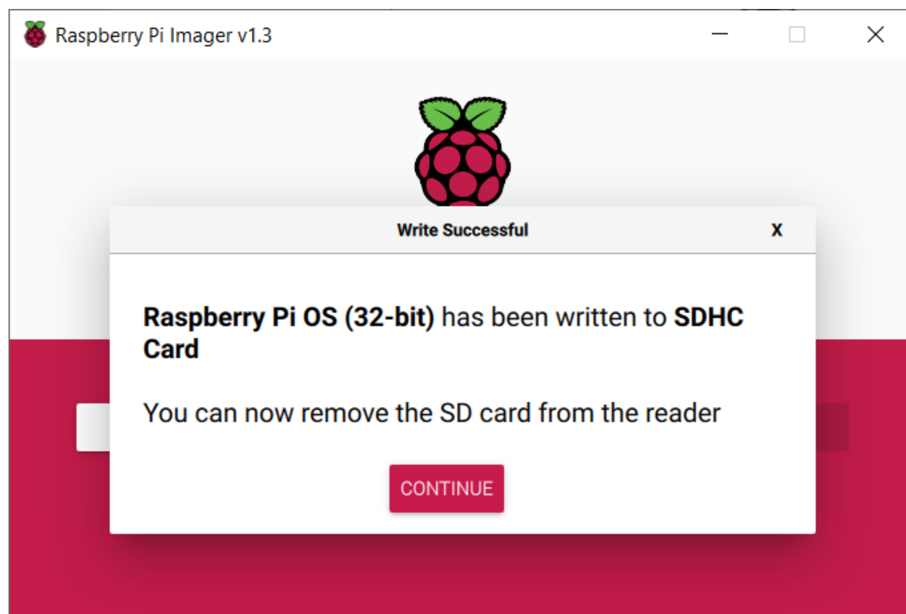


Figura A-5: confirmación de la herramienta Raspberry Pi Imager de escritura satisfactoria del sistema operativo en la tarjeta de memoria

Una vez que tenemos el sistema operativo listo, debemos volver a meter la tarjeta en el ordenador porque hay un par de acciones que debemos realizar para configurar el acceso por SSH para no necesitar acceder físicamente a la Raspberry, y la red WiFi (Piltch, 2020), en caso de que queramos usar una red inalámbrica en lugar de cable de red:

- Para habilitar el acceso por SSH es necesario crear un fichero en la raíz de la tarjeta con el nombre **ssh** y sin ningún contenido.

- Para configurar la conexión WiFi es necesario crear otro fichero en la raíz de la tarjeta con el nombre **wpa_supplicant.conf**, y con el contenido mostrado en la Figura F-2. Hay que poner el nombre de la conexión WiFi y la contraseña correspondiente.

```
country=ES
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
	scan_ssid=1
	ssid="WIFI_NAME"
	psk="WIFI_PASSWORD"
}
```

Figura A-6: configuración de conexión WiFi a guardar en la tarjeta de memoria

Ahora ya podemos introducir la tarjeta en la Raspberry e iniciarla.

Tras un tiempo prudencial, quizás un par minutos, podemos conectarnos a la Raspberry por SSH con el usuario **pi**, cuya contraseña inicial es **raspberry** (Figura A-7).

```
λ ssh pi@raspberrypi
pi@raspberrypi's password:
Linux raspberrypi 4.19.118-v7+ #1311 SMP Mon Apr 27 14:21:24 BST
2020 armv7l

The programs included with the Debian GNU/Linux system are free
software;
the exact distribution terms for each program are described in
the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed May 27 08:31:31 2020
```

```
SSH is enabled and the default password for the 'pi' user has
not been changed.
This is a security risk - please login as the 'pi' user and type
'passwd' to set a new password.
```

Figura A-7: conexión SSH a la Raspberry

Lo primero que debemos hacer una vez que estamos conectados es cambiar la contraseña inicial del usuario *pi* (Figura A-8), ya que de no hacerlo cualquier persona con acceso de red a la Raspberry podría potencialmente acceder a la misma.

```
pi@raspberrypi:~ $ passwd
Changing password for pi.
Current password:
New password:
Retype new password:
passwd: password updated successfully
```

Figura A-8: cambio de contraseña por defecto del usuario por defecto de la Raspberry

Cambiar el nombre de host de la Raspberry, usando para ello la utilidad `raspi-config` (Figura A-9 y Figura A-10) que viene incluida dentro de la instalación del sistema operativo, para que cada Raspberry tenga un nombre único. El nombre que le demos a la Raspberry puede ser cualquiera, pero seguramente será más sencillo posteriormente si elegimos nombres intuitivos en función del rol que le vayamos a dar a la Raspberry.

```
$ sudo raspi-config
```

Figura A-9: ejecución de la utilidad raspi-config que nos permite realizar varias tareas de configuración sobre la Raspberry

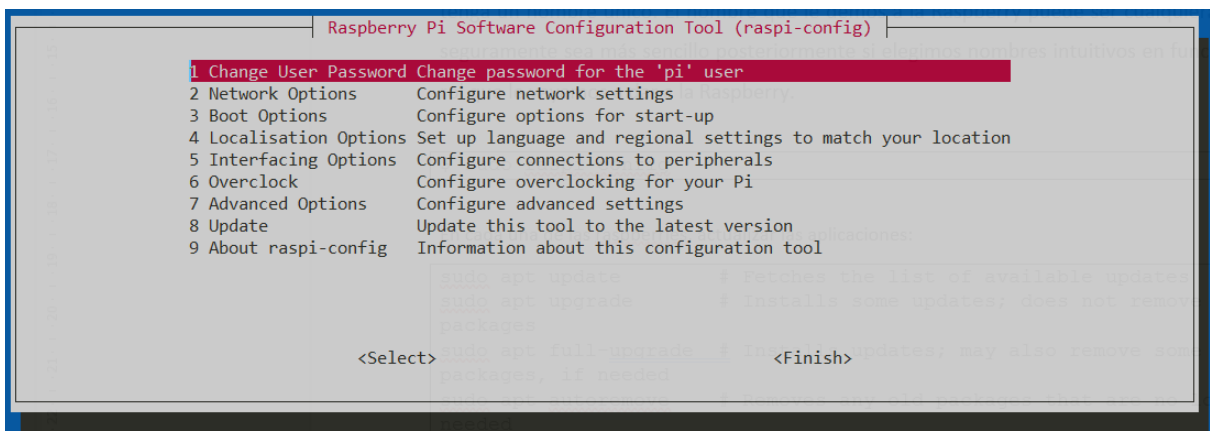


Figura A-10: menú inicial de la utilidad raspi-config

Para el cambio del nombre de host debemos ir al menú *2 Network Options* (Figura A-11) y después a la opción *N1 Hostname* (Figura A-12) y finalmente escribir el nombre de host que queramos (Figura A-13).

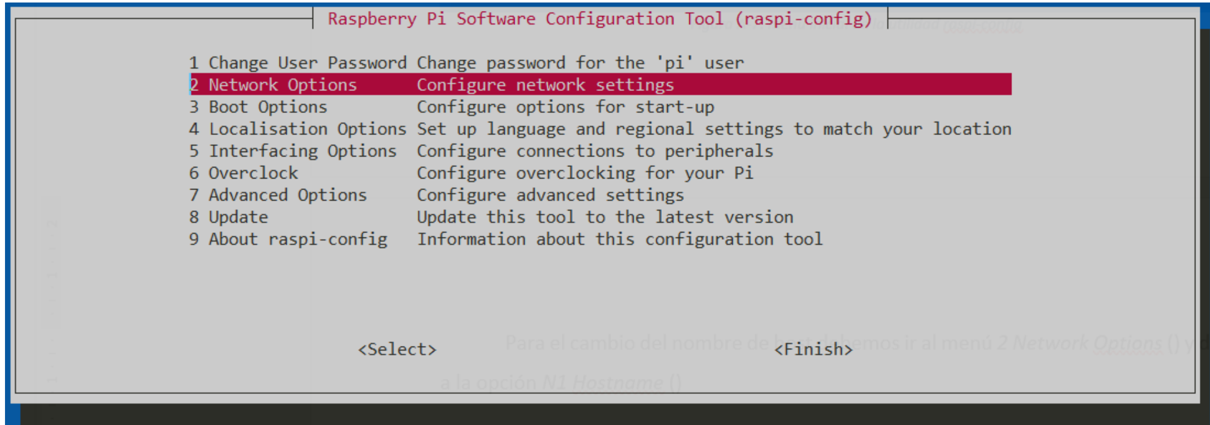


Figura A-11: selección del menú de opciones de red en la utilidad raspi-config

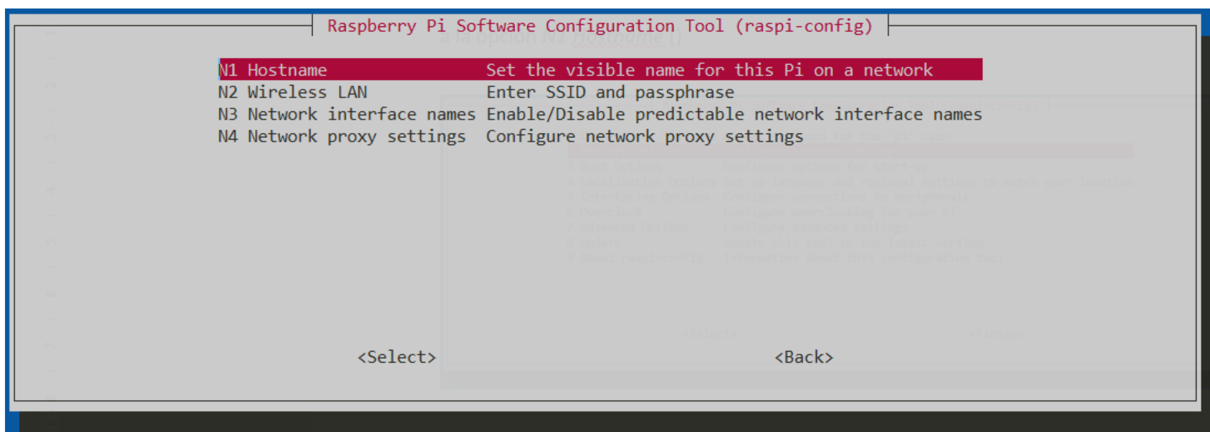


Figura A-12: selección del submenú de host en la utilidad raspi-config

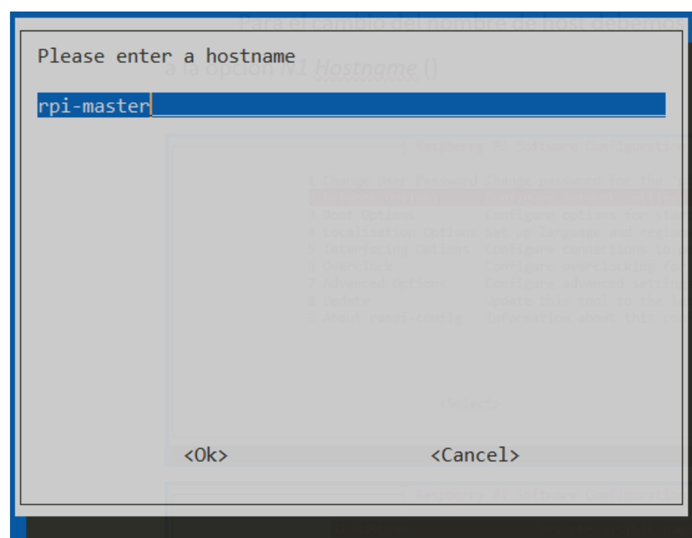


Figura A-13: configuración del nombre de host en raspi-config

Tras fijar el nombre de host debemos reiniciar la Raspberry.

Debemos conectarnos de nuevo a la Raspberry, esta vez para actualizar todos los paquetes instalados. Este paso no es imprescindible pero sí recomendable, y no sólo en este momento inicial sino hacerlo regularmente (por ejemplo, una vez al mes) para tener siempre las últimas versiones, que deberían tener menos errores y vulnerabilidades. Para realizar esta actualización debemos ejecutar cuatro comandos (askubuntu.com, 2012), pero podemos ejecutarlos a la vez, tal como se muestra en la Figura A-14. La ejecución de estos comandos puede tardar hasta veinte minutos más o menos.

```
$ sudo apt update && sudo apt upgrade && sudo apt full-upgrade && sudo apt autoremove
```

Figura A-14: comandos para actualizar los paquetes instalados en la Raspberry

También configurar una fijar una IP fija dentro de la red que vayamos a usar para las Raspberry. La razón para esto es asegurar que, aunque una Raspberry salga de la red o se reinicie, siempre volverá con la misma dirección IP, evitando posibles problemas durante la operación del clúster. Este proceso sin embargo depende de cada router, por lo que no podemos indicar los pasos a seguir.

Hasta ahora todos los pasos seguidos eran comunes a todas las Raspberry, independientemente del rol que fuesen a tomar. Pero ahora eso cambia, ya que vamos a instalar k3s en las Raspberry, y la instalación es ligeramente diferente en función del rol.

Para la Raspberry máster debemos instalar k3s con el comando mostrado en la Figura A-15.

```
$ curl -sfL https://get.k3s.io | sh -
```

Figura A-15: comando para la instalación de k3s en la Raspberry master

Cuando la instalación haya terminado, podemos recuperar los nodos del clúster recién creado para comprobar que se ha creado correctamente el clúster, con un único nodo (Figura A-16).

```
pi@rpi-master:~ $ sudo k3s kubectl get nodes
NAME         STATUS   ROLES    AGE   VERSION
rpi-master   Ready    master   51s   v1.18.4+k3s1
```

Figura A-16: ejecución del comando para la obtención del nodo recién instalado en k3s, en la Raspberry master

El nodo máster ya está listo y ya podemos decir que tenemos un clúster listo. Pero antes de hacer nada en el clúster vamos a instalar el resto de los nodos. Para ello primero debemos recuperar un token de unión al clúster que está dentro de la Raspberry máster recién configurada. Para ello debemos obtener el contenido del fichero `/var/lib/rancher/k3s/server/node-token`, tal como vemos en la Figura A-17.

```
pi@rpi-master:~ $ sudo cat /var/lib/rancher/k3s/server/node-
token
K10111a8250687953846e01436a7d06d6cfda9a80a5a2496e6d7a000101aaa
e1211::server:84cf2ac2555575df7777d0b8296a256e4
```

Figura A-17: obtención del token desde la Raspberry master para que otros nodos puedan unirse al clúster

Ahora ya podemos conectar los otros nodos, que tendrán el rol *worker*, usando el token recuperado en la Figura A-17. Para ello, en cada una de las otras Raspberry debemos ejecutar el comando de la Figura A-18.

```
$ curl -sfL https://get.k3s.io | K3S_URL=https://rpi-master:6443
K3S_TOKEN=K10111a8250687953846e01436a7d06d6cfda9a80a5a2496e6d7
a000101aaae1211::server:84cf2ac2555575df7777d0b8296a256e4 sh -
```

Figura A-18: comando para la instalación de k3s en las Raspberry worker

Cuando hemos instalado k3s en cada una de las otras Raspberry, si recuperamos los nodos podremos ver todos los nodos que se han unido al clúster (Figura A-19). Todos los comandos sobre el clúster se deben ejecutar desde la Raspberry máster, así que a partir de ahora todos los comandos se ejecutarán en dicha Raspberry.

```
pi@rpi-master:~ $ sudo k3s kubectl get nodes
NAME          STATUS   ROLES    AGE     VERSION
rpi-worker1   Ready    <none>   23m     v1.18.4+k3s1
rpi-worker3   Ready    <none>   23m     v1.18.4+k3s1
rpi-worker2   Ready    <none>   2m17s   v1.18.4+k3s1
rpi-master    Ready    master   145m    v1.18.4+k3s1
```

Figura A-19: listado de nodos unidos al clúster. Comando ejecutado desde la Raspberry master, donde se ve que inicialmente los nodos worker no tienen un rol asignado

Sin embargo, tal como vemos en la Figura A-19, los nodos *worker* no tienen un rol asignado, así vamos a asignárselo (Figura A-20).

```
pi@rpi-master:~ $ sudo k3s kubectl label nodes rpi-worker1 node-role.kubernetes.io/worker=true
node/rpi-worker1 labeled
pi@rpi-master:~ $ sudo k3s kubectl label nodes rpi-worker2 node-role.kubernetes.io/worker=true
node/rpi-worker2 labeled
pi@rpi-master:~ $ sudo k3s kubectl label nodes rpi-worker3 node-role.kubernetes.io/worker=true
node/rpi-worker3 labeled
```

Figura A-20: asignación del rol worker a cada una de las Raspberry worker

Y ahora si volvemos a recuperar el listado de nodos del clúster veremos que cada nodo tiene un rol asignado (Figura A-21).

```
pi@rpi-master:~ $ sudo k3s kubectl get nodes
NAME          STATUS   ROLES    AGE    VERSION
rpi-master    Ready    master   158m   v1.18.4+k3s1
rpi-worker2   Ready    worker   15m    v1.18.4+k3s1
rpi-worker1   Ready    worker   36m    v1.18.4+k3s1
rpi-worker3   Ready    worker   36m    v1.18.4+k3s1
```

Figura A-21: listado de nodos unidos al clúster, ahora ya con todos los nodos con un rol asignado

Ahora ya sí tenemos el clúster preparado, pero antes de dar la instalación por finalizada vamos a configurar otros parámetros que nos serán necesarios para trabajar con la aplicación OIDC. Esta configuración se puede hacer ya y hacer la instalación de la aplicación OIDC

directamente desde la Raspberry *master*, o después de haber instalado la aplicación OIDC. Para ello debemos editar el fichero `/etc/systemd/system/k3s.service` dentro de la Raspberry *master* y cambiar la línea donde se define la variable `ExecStart` y poner el contenido de la Figura A-22.

```
ExecStart=/usr/local/bin/k3s \
    --debug \
    server \
    --kube-apiserver-arg          "oidc-issuer-url=https://tfm-
atellez.ngrok.io/oidc" \
    --kube-apiserver-arg "oidc-client-id=kubernetes" \
    --kube-apiserver-arg "oidc-username-claim=sub" \
    --kube-apiserver-arg "oidc-username-prefix=oidc:" \
    --kube-apiserver-arg "oidc-groups-claim=clustergroups" \
    --kube-apiserver-arg "oidc-groups-prefix=oidc:" \
```

Figura A-22: parámetros usados para la ejecución del clúster

Los parámetros que se deben añadir son los relativos a la configuración de la autenticación vía OIDC para el clúster. Dichos parámetros son:

- oidc-issuer-url: URL donde la aplicación OIDC está disponible.
- oidc-client-id: siempre el valor kubernetes.
- oidc-username-claim: campo del *id_token* donde estará el id del usuario autenticado.
- oidc-username-prefix: prefijo que se añadirá al nombre del usuario recuperado desde el *id_token*, en el lado de kubernetes. En nuestra instalación el prefijo es *oidc:*. Aunque no estamos definiendo roles a nivel de id de usuario, hay que tenerlo en cuenta en caso de que así lo hiciésemos.
- oidc-groups-claim: nombre del campo donde se encontrarán los grupos dentro del *id_token*. En este caso será el campo *clustergroups* (ver apartado 4.2.1.6).
- oidc-groups-prefix: prefijo que se añadirá a los grupos recuperados desde el *id_token*, en el lado de kubernetes. En nuestra instalación el prefijo es *oidc:*. Este se entiende mejor en los apartados 5.1.2 y 5.1.3.

Para más información sobre la aplicación OIDC, que es a la que van destinadas estas parametrizaciones, el lector puede acudir a los apartados 4.2.1.1 y 4.2.1.6.

B. Exposición del clúster hacia fuera del mismo

Cuando hablamos de exposición del clúster nos referimos a poder acceder a aplicaciones desplegadas en el mismo desde fuera del propio clúster.

Además, las aplicaciones que serán accedidas desde fuera serán únicamente las que así se hayan configurado (por medio de un `Ingress`), por lo que podemos dejar algunas aplicaciones que nos interesen sólo para su acceso entro del clúster, y exponer las que queramos.

Antes de hablar de algunas posibilidades que se podrían contemplar como configuración dentro de la universidad, veamos cómo se ha configurado para este trabajo:

Puesto que no se ha podido disponer de ninguna infraestructura de la universidad, y debido a que tampoco contábamos con ninguna otra posible infraestructura que pudiéramos usar, hemos optado por usar un servicio que crea un túnel entre la máquina y puerto que se indique y una DNS que es accesible desde fuera. Este servicio es ngrok, y tal como describen, es *un servicio que expone servidores locales detrás de NATs y firewalls a la internet pública sobre túneles seguros* (Ngrok, 2020).

En nuestro caso, hemos optado por exponer únicamente por HTTPS y bajo el dominio tfm-atellez.ngrok.io. Por tanto, cuando accedemos a dicho dominio lo que acaba ocurriendo es que estamos accediendo a nuestro clúster. En el clúster exponemos todas las aplicaciones vía HTTP, y es ngrok el que se encarga de la negociación de la conexión segura antes de redirigir el tráfico hacia el clúster. Esto nos simplifica el trabajo ya que no debemos preocuparnos de ningún certificado local, así como de una exposición del clúster a otras redes, ya que de todo eso se encarga ngrok por nosotros.

Esto, sin embargo, no es la mejor solución y es algo que sólo debería usarse en entornos locales, tal como ha sido el nuestro. Por un lado, al usar alguna herramienta externa que no podemos controlar, dependemos de dicha herramienta y, en caso de que haya algún problema en la misma, no tendremos acceso a nuestro clúster sin poder hacer nada al respecto. Además, el no poder controlar los certificados del dominio no es nada recomendable y sin duda, algo que no debería hacerse en un entorno productivo.

Una alternativa que podríamos haber hecho, sin hacer uso de aplicaciones externa, es montar un servidor web, Nginx por ejemplo, dentro del propio clúster y que éste actuase como proxy inverso. El hecho de no haber escogido esta alternativa ha sido por disponer finalmente de un clúster limpio tal y como se espera que sea una vez se monte dentro de la universidad donde corresponda.

En cualquier caso, la exposición del clúster supone simplemente conectar el dominio que deseemos usar con la IP principal del clúster, que se corresponde con la IP de la Raspberry master, y hacer el mapeo de puertos que corresponda (el servidor web externo debería ser accedido vía el puerto 443 mientras que en el clúster podría ser el puerto 80). Y por último, es muy aconsejable crear las medidas oportunas para eliminar cualquier acceso al clúster que no sea desde estos, porque pueden ser más de uno si así se desea (por ejemplo, uno interno para aplicaciones sensibles y otro externo para aplicaciones públicas), para así evitar posibles accesos no deseados.

Por último, hay que mencionar que la URL aquí mostrada sólo es válida de cara al trabajo, y que dejará de estar en uso una vez termine el trabajo.

C. Configuración del clúster en sí para su acceso remoto

Cuando queramos trabajar con el clúster en sí tenemos que lanzar comandos sobre el propio clúster por medio de la interfaz de línea de comandos *kubectl*. Estos comandos se pueden ejecutar directamente entrando en la Raspberry máster y lanzarlos desde ahí, pero esta opción seguramente no sea cómoda, ya que requiere estar trabajando en otro dispositivo en remoto, y desde luego no es la más segura, ya que el hecho de permitir accesos remotos vía SSH requiere una gestión de usuarios de accesos a la Raspberry y control de permisos sobre la misma, y al tratarse de la Raspberry máster podría suponer un impacto directo sobre el clúster.

Es por ello por lo que es muy recomendable trabajar con el clúster directamente desde el ordenador personal de cada usuario potencial, y no habilitar el acceso SSH a la Raspberry salvo para los administradores del clúster, que a su vez sólo necesitarán acceder por esta vía para realizar tareas propias del mantenimiento de la Raspberry o para solucionar problemas, pero no para realizar la operativa normal del clúster.

Como prerrequisito para poder configurar el clúster de forma local es necesario contar con una instalación de kubernetes en el dispositivo sobre el que vayamos a trabajar. El autor recomienda instalar Docker (Docker, 2020), ya que actualmente incluye también kubernetes, y si se trabaja con un clúster de kubernetes es muy posible que trabajar con Docker sea necesario en algún momento. No obstante, la forma de instalar kubernetes queda a discreción del lector.

Antes de comenzar con la explicación de cómo hacer esta configuración hay que aclarar que si se opta por usar la aplicación descrita en el apartado 4.2.2 no es necesario hacer nada de esto, ya que la propia aplicación será la encargada de hacerlo. Aún así, puede ser recomendable seguir leyendo para entender qué es lo que hace dicha aplicación.

La configuración del clúster se puede en el fichero `/etc/rancher/k3s/k3s.yaml` de la Raspberry máster (Figura C-1). Los apartados más destacables de la configuración son:

- Sección *clusters*: listado con los clústeres que tenemos configurados. Tendremos sólo uno configurado, que se compone del servidor de acceso a la

API de kubernetes. Como la configuración está en la Rasberry máster y es para su clúster local, la URL apuntará a la IP 127.0.0.1 y el puerto que corresponda. También incluye el certificado necesario para realizar una conexión segura y un nombre, que será *default*.

- Sección *users*: listado de usuarios, que en nuestro caso sólo tendremos uno definido. Cada usuario se identifica con su nombre. Las credenciales definidas en este caso serán un nombre de usuario (*admin*) y una contraseña. Este usuario definido tiene permisos de administrador.
- Sección *contexts*: esta sección conecta un clúster con un usuario de acceso al mismo, y a esta asociación se le identifica con un nombre.
- Campo *current-context*: indica cual es el contexto de trabajo actual, lo que indica qué usuario y qué clúster se usarán.

La configuración vista en la Figura C-1 solo nos sirve tal cual para trabajar desde la Raspberry máster, pero nos sirve para ver la URL de acceso al clúster y el certificado de acceso, que son los datos que necesitaremos para configurar el clúster en otro dispositivo[10].

```
pi@rpi-master:~ $ sudo cat /etc/rancher/k3s/k3s.yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTi…
    server: https://127.0.0.1:6443
  name: default
contexts:
- context:
    cluster: default
    user: default
  name: default
```

[10] Estamos dando por hecho que se usará el acceso al clúster vía OIDC (ver apartado 4.2.1) y que por tanto no necesitamos el usuario administrador definido. Si no fuese el caso y se decidiese a trabajar únicamente con el usuario administrador, algo muy desaconsejable, entonces también será necesario copiar el usuario y contraseña definido

```
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    password: 3a932da0f9ab4f664cf9812400000000
    username: admin
```

Figura C-1: configuración del acceso al clúster en la Raspberry máster

En realidad, la URL de la Figura C-1 no nos vale tal cual, ya que apunta a sí misma, por lo que cuando lo configuremos en otro dispositivo, si no la cambiamos, apuntará al propio dispositivo, y como el clúster no se encuentra allí, fallará. Para adaptar la URL necesitamos saber cuál es el nombre del host de la Raspberry o una URL que nos permita llegar a ella, y en nuestro caso, puesto que estamos trabajando en la misma red local, nos vale con indicar el nombre de máquina para llegar a la Raspberry máster. Así, en este caso, la URL de acceso al clúster será, tras sustituir `127.0.0.1` por el nombre del host de la Raspberry master, `https://rpi-master:6443`.

Por otro lado, hay hacer una pequeña preparación sobre el certificado de conexión con el clúster. Para ello, debemos coger todo el valor contenido en el campo `certificate-authority-data`, y guardarlo en un fichero de texto, pero después de haber decodificado base64 dicho valor. Así, nos debería quedar un fichero parecido al de la Figura C-2.


```
-----BEGIN CERTIFICATE-----
MIIBWDCB/qADAgECAgEAMAoGCCqGSM49BAMCMCMxITAfBgNVBAMMGGszcy1zZXJ2
ZXItY2FAMTU5NTA3NDQ4MTAeFw0yMDA3MTgxMjE0NDFaFw0zMDA3MTYxMjE0NDFa
MCMxITAfBgNVBAMMGGszcy1zZXJ2ZXItY2FAMTU5NTA3NDQ4MTBZMBMGByqGSM49
AgEGCCqGSM49AwEHA0IABK3HV8x1WH9LdzAEsL6gUYCem1Iwi8aXhID9ZeXFHM3S
LwzGePcS5BGFIVLybPyl9lt4QUXSCkQfWprskRfq9cejIzAhMA4GA1UdDwEB/wQE
AwICpDAPBgNVHRMBAf8EBTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQCfeIHGnDnV
ilNo3sYUyukOA98qoQfRfHcpQTIu9nnqpwIhANPV3dN5aoM/NagfPaS3WFivTJLT
uNbHl6Q0m9xfC5uO
-----END CERTIFICATE-----
```


Figura C-2: certificado para conectarse al clúster con una conexión segura

Ahora ya tenemos los elementos que necesitamos para poder configurar el clúster en el dispositivo local sobre el que estemos trabajando. Esta configuración podemos hacerla

directamente modificando el fichero de configuración usado por kubernetes[11], pero seguramente sea más sencillo e intuitivo si lo hacemos siguiendo una serie de comandos que podemos lanzar usando la interfaz *kubectl*.

El primer paso que debemos hacer es crear la configuración clúster. Para ello hay que indicar cuál es la URL del servidor, la que habíamos visto anteriormente, y la ruta hasta el fichero que contiene el certificado que habíamos preparado para la Figura C-2. Además, debemos darle un nombre al clúster. El comando que hay que ejecutar podemos verlo en la Figura C-3.

```
λ kubectl config set-cluster enmolabs --server=https:// rpi-master:6443 --embed-certs=true --certificate-authority=clusterca.pem
Cluster "enmolabs" set.
```

Figura C-3: comando para crear un clúster dentro de la configuración general de kubernetes en el dispositivo local

Lo siguiente que debemos hacer es crear un contexto que relacionará el usuario que vayamos a usar con el clúster recién creado, tal como vemos en la Figura C-4.

```
λ kubectl config set-context unedenmolabs --cluster=enmolabs --user=baldomero
Context "unedenmolabs" created.
```

Figura C-4: comando para crear el contexto que relaciona el clúster con el usuario de ejecución, dentro de la configuración general de kubernetes en el dispositivo local

Cada vez que se quiera trabajar con un contexto específico, tanto si es la primera vez que se quiere trabajar con dicho contexto como si se había cambiado a otro contexto, hay que indicarlo para que kubernetes sepa hacia dónde debe mandar las órdenes que se vayan a dar. Esto lo podemos con el comando de la Figura C-5.

```
λ kubectl config use-context unedenmolabs
Switched to context "unedenmolabs".
```

Figura C-5: comando para seleccionar el contexto con el queramos trabajar, dentro de la configuración general de kubernetes en el dispositivo local

---

[11] Este fichero normalmente se encuentra en la ruta `C:\Users\{usuario}\.kube\config` para Windows, o en `$HOME/.kube/config` para Linux y Mac

Finalmente, hay que crear el usuario con sus credenciales. El comando necesario para ello varía en función del mecanismo de autenticación que tenga configurado el clúster, pero por sintonía con lo realizado en este trabajo (apartado 4.2.1) y con la configuración que hemos mostrado en el anexo A, vamos a suponer que el mecanismo de autenticación es vía OIDC. El comando a usar se puede ver en la Figura C-6, teniendo en cuenta que el token a usar debe venir desde el *identity provider* (proveedor de identidad) correspondiente, que en nuestro caso será el detallado en el apartado 4.2.1.

```
λ kubectl config set-credentials baldomero --auth-provider=oidc
--auth-provider-arg=idp-issuer-url=https://tfm-
atellez.ngrok.io/oidc  --auth-provider-arg=client-id=kubernetes
--auth-provider-arg=id-token=eyJhbGciOiJSUz…
User "baldomero" set.
```

Figura C-6: comando para crear/actualizar las credenciales de un usuario de kubernetes, dentro de la configuración general de kubernetes en el dispositivo local

Una vez seguidos estos pasos ya se podrá trabajar con el clúster de forma remota desde el dispositivo local donde se haya realizado la configuración.

D. Problemas y utilidades

En este anexo se ha pensado como un pequeño manual de consulta sobre algunos de los problemas que han surgido durante el uso del clúster en el transcurso del desarrollo de este trabajo y las soluciones llevadas a cabo, así como algunos comandos que podrían ser de utilidad para el mantenimiento de otros futuros clústeres.

Ver el estado de k3s

Para ver el estado actual de k3s, podemos ejecutar el comando de la Figura D-1 desde la Raspberry máster. El comando dará como salida veremos, entre otras cosas, el estado actual, y desde cuando está en funcionamiento el clúster. Esta fecha se resetea cuando se reinicia el clúster.

```
pi@rpi-master:~ $ sudo systemctl status k3s
● k3s.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s.service; enabled;
vendor preset: enabled)
   Active: active (running) since Fri 2020-09-18 16:41:24 BST;
20h ago
     Docs: https://k3s.io
  Process: 359 ExecStartPre=/sbin/modprobe br_netfilter
(code=exited, status=0/SUCCESS)
  Process: 439 ExecStartPre=/sbin/modprobe overlay
(code=exited, status=0/SUCCESS)
 Main PID: 443 (k3s-server)
    Tasks: 116
   Memory: 107.0M
   CGroup: /system.slice/k3s.service
…
```

Figura D-1: ejecución del comando para ver el estado de k3s, desde la Raspberry máster

Ver los logs de k3s

Alguna vez será necesario acudir a los logs de k3s para intentar entender algún problema que pueda haber en este momento, como, por ejemplo, una situación en la que el clúster no arranque. Para poder ver los logs debemos ejecutar el comando de la Figura D-2 desde la Raspberry máster.

```
pi@rpi-master:~ $ journalctl -u k3s
-- Logs begin at Sat 2020-09-19 10:10:37 BST, end at Sat 2020-09-19 13:33:41 BST. --
Sep 19 10:10:37 rpi-master k3s[443]: time="2020-09-19T10:10:37.152579185+01:00" level=debug msg="LIST /registry/certiSep 19 10:10:37 rpi-master k3s[443]: time="2020-09-19T10:10:37.152813091+01:00" level=debug msg="COUNT /registry/certSep 19 10:10:39 rpi-master k3s[443]: time="2020-09-19T10:10:39.330626825+01:00" level=debug msg="UPDATE /registry/lea
…
```

Figura D-2: ejecución del comando para ver los logs de k3s, desde la Raspberry máster

Reiniciar el clúster

A veces puede ser necesario reiniciar el clúster, bien sea para que una configuración de k3s tome efecto, o bien para refrescar el estado del clúster. En estos casos no es suficiente si reiniciamos la Raspberry máster, y, de hecho, eso no sirve para reiniciar el clúster. Lo que debemos hacer es ejecutar el comando de la Figura D-3 desde la Raspberry máster.

```
sudo systemctl restart k3s
```

Figura D-3: comando para reiniciar el clúster k3s, ejecutado desde la Raspberry máster

Tras lanzar su ejecución tendremos que esperar unos segundos hasta que el reinicio termine.

Error *net/http: request canceled while waiting for connection*

La primera implementación del clúster se hizo con una exposición únicamente dentro de la red local, usando un dominio creado localmente y un certificado firmado por una autoridad de confianza creada localmente.

Debido al uso de este dominio local, recibíamos el error de la Figura D-4 cuando intentábamos configurar la aplicación OIDC.

```
may 07 11:17:56 rpi-master k3s[396]: E0507 11:17:56.729180
396 oidc.go:232] oidc authenticator: initializing plugin: Get
https://miclusterlocal.net:3000/.well-known/openid-
configuration: net/http: request canceled while waiting for
connection (Client.Timeout exceeded while awaiting headers)
```

Figura D-4: error de conexión con un dominio creado para una red local

Puesto que se trataba de un dominio disponible dentro de la red local, los servidores DNS configurados en la Raspberry, que de forma indirecta eran los que se usaban a través del clúster, no eran capaces de resolver el dominio, y de ahí el error al intentar realizar la conexión.

Para solucionar este problema basta con modificar el fichero `/etc/hosts` de la Raspberry máster para añadir una entrada manual que resuelva este dominio local, tal como se muestra en la Figura D-5.

```
pi@rpi-master:~ $ cat /etc/hosts
127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters

127.0.1.1       rpi-master
127.0.0.1       miclusterlocal.net
```

Figura D-5: definición de hosts que serán resueltos sin usar servidores DNS, en la Raspberry máster

Error oidc authenticator: initializing plugin:… x509: certificate signed by unknown authority

Con el enfoque inicial del clúster que se ha comentado en el punto anterior, las peticiones a cada aplicación llegaban directamente a ellas mismas y, por tanto, debían gestionar su propia conexión segura, lo que incluye su certificado y clave privada. Esto no era problema para la aplicación OIDC, pero de cara a validar el certificado resultaba en un error ya que el clúster no reconocía la entidad que había emitido el certificado ofrecido por la aplicación:

```
may 07 17:19:52 rpi-master k3s[398]: E0507 17:19:52.832190
398 oidc.go:232] oidc authenticator: initializing plugin: Get
https://micluster.net:8044/.well-known/openid-configuration:
x509: certificate signed by unknown authority
```

Figura D-6: error de validación de certificado al intentar conectar a la aplicación OIDC, al ser un certificado firmado por una entidad no reconocida

Para solucionarlo, es necesario incluir la cadena de certificados de confianza con los certificados de la entidad intermedia y la entidad emisora, y añadir otro parámetro al arranque de k3s indicando la ruta al fichero que contiene esa cadena de confianza:

```
--kube-apiserver-arg "oidc-ca-
file=/var/lib/rancher/k3s/server/tls/oidc-ca-chain.cert.pem"
```

Figura D-7: error de validación de certificado al intentar conectar a la aplicación OIDC, al ser un certificado firmado por una entidad no reconocida

Error nodo NotReady

A veces ocurre que un nodo no está disponible y aunque se reinicie la Raspberry o se deje un tiempo, no vuelve a estar operativo. Si listamos los nodos del clúster (Figura D-8) podemos ver que su estado es *NotReady*. Para solucionarlo debemos reiniciar el clúster (k3s, 2019).

```
λ kubectl get nodes
NAME          STATUS     ROLES    AGE   VERSION
rpi-worker2   NotReady   worker   75d   v1.17.4+k3s1
…
```

Figura D-8: nodo NotReady en el clúster, al consultar el listado de nodos del clúster

Pérdida de conectividad de los pods hacia otros servicios internos del clúster y/o URLs externas

Durante las últimas semanas de realización de este trabajo el he estado experimentando un problema de pérdida de conectividad des los pods hacia algunos servicios internos del clúster y/o externos. El error era claro: no se puede conectar con alguno de los servicios internos y/o URL externa desde ningún pod del clúster. Sin embargo, se trataban de servicios internos que estaban levantados o de URLs externas que eran accesibles, y que con anterioridad funcionaban, y sin haber cambiado nada desde entonces.

No he podido encontrar un patrón de cuál era el problema, ya que la pérdida de conectividad se producía desde/hacia pods en cualquiera de los nodos, por lo que el problema no estaba relacionado claramente con un nodo en concreto. Tampoco había nada en el log del clúster que ayudase a entender el problema. Y la depuración del problema tampoco ayudaba demasiado porque parecía que el destino al que se intentaba conectar simplemente no se podía resolver, como si fuese un dominio inválido o inexistente. El pod relacionado con la asignación de DNS (*coredns*) parecía estar correcto y sin errores. Y aunque se reiniciasen los pods, el problema persistía.

En el momento de escribir esta memoria el problema el autor sigue sin haber encontrado la causa al problema, aunque sí hay sospechas de que quizás pueda ser debido a que la Raspberry máster, el dispositivo en sí, pueda estar sufriendo fallos, ya que coincide que en las últimas semanas se ha quedado colgado un par de veces, y otras veces ha sido necesario su reinicio al, aun sin estar colgado, ser tremendamente lenta al procesar las órdenes.

En cualquier caso, aunque no se puede dar una solución puesto que no se conoce el problema realmente, sí hay un conjunto de operaciones que, de alguna manera, han solucionado el problema en esta situación: reiniciar la red, reiniciar la Raspberry y reiniciar el clúster. En concreto, la combinación que parece más efectiva es la siguiente:

Para la Raspberry máster:

```
sudo systemctl restart networking
sudo systemctl restart k3s
sudo reboot
sudo systemctl restart k3s
```

Figura D-9: pasos a seguir en la Raspberry máster para intentar solucionar el problema de conectividad

Y para cada una de las Raspberry *worker*:

```
sudo systemctl restart networking
sudo reboot
```

Figura D-10: pasos a seguir en cada Raspberry worker para intentar solucionar el problema de conectividad

Alguna vez se ha solucionado haciendo primero las operaciones en la máster y luego en las *worker*, y otras veces al revés. Como decíamos, no se ha encontrado ningún patrón.

Consultar la configuración de kubernetes

En caso de que fuese necesario consultar la configuración de kubernetes, se puede hacer bien yendo al fichero de configuración de kubernetes que corresponda, o bien con el comando de la Figura D-11, que nos mostrará toda la configuración definida, aunque no necesariamente será únicamente con relación a nuestro clúster.

```
λ kubectl config view
```

Figura D-11: comando para mostrar la configuración de kubernetes

Si lo que se quiere ver es cuál es el contexto sobre el que estamos trabajando, se puede ejecutar el comando de la Figura D-12, donde recuperaremos todos los contextos que haya

configurados en la configuración de kubernetes e indicando cuál es el contexto actual sobre el que estamos trabajando.

```
λ kubectl config get-contexts
CURRENT   NAME                   CLUSTER          AUTHINFO
NAMESPACE
          docker-desktop         docker-desktop   docker-desktop
          docker-for-desktop     docker-desktop   docker-desktop
*         rpi                    raspis           atellez9
```

Figura D-12: comando para mostrar el listado de contextos disponibles, e indicando cuál es el actual, desde la configuración de kubernetes

O si se quiere recuperar el id_token que haya configurado para un usuario particular, se puede ejecutar el comando de la Figura D-13.

```
λ kubectl config view -o
jsonpath="{.users[?(@.name=='baldomero')].user.auth-
provider.config.id-token}"
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6InIxTGtiQm8zOTI…
```

Figura D-13: comando para mostrar el id_token para un usuario concreto, recuperándolo desde la configuración de kubernetes

E. Listado de ficheros YAML usados en la creación del clúster

Los ficheros YAML usados a lo largo de este trabajo están, en su mayoría, detallados y explicados en el contexto al que pertenecen, por lo que este anexo no pretende repetir lo mismo de nuevo. Así pues, lo que pretendemos es generar una especie de índice sobre los ficheros, listando todos los ficheros usados para este trabajo y resumir brevemente su utilidad, así como dónde se encuentra el contexto donde explica o hablar de él.

Puesto que los ficheros pertenecen a grupos distintos, vamos a listar los ficheros en función del grupo al que pertenecen:

Instalación del clúster

Este grupo se explica en el anexo A.

- **ssh**. Fichero para activar el acceso vía SSH a una Raspberry.
- **wpa_supplicant.conf**. Configuración de la red inalámbrica en una Raspberry. Ver Figura A-6.

Instalación de la aplicación OIDC

Este grupo se explica en el apartado 4.2.1.

- **00-namespace.yaml**. Creación del *namespace* seguridad.
- **10-redis-oidc.yaml**. Instalación y exposición interna de la base de datos Redis.
- **21-configmap-seguridad.yaml**. Configuración de la aplicación OIDC. Ver Figura 4-13.
- **22-configmap-map-grupos.yaml**. Definición de grupos que se asociarán a los usuarios que se autentiquen con la aplicación OIDC según el id de usuario y dominio de email. Ver Figura 4-14.
- **23-configmap-map-usuarios-locales.yaml**. Definición de los usuarios locales que estarán disponibles aun sin existir como credenciales de la universidad. Ver Figura 4-17.

- **30-uned-oidc.yaml**. Instalación y exposición tanto interna como externa al clúster de la aplicación OIDC. Ver Figura 4-6, Figura 4-7 y Figura 4-8.
- **31-uned-oidc-hpa.yaml**. Configuración de escalado horizontal de la aplicación OIDC. Ver Figura 4-18.
- **40-cluster_admin.yaml**. Asociación de permisos de administrador al grupo designado como tal en la configuración de la aplicaciónOIDC. Ver Figura 4-5.

Instalación de las herramientas de monitorización

Este grupo se explica en el apartado 4.3.

- **00-namespace.yaml**. Creación del *namespace* monitoring.
- **10-prom-clusterrole.yaml**. Asignación de permisos de lectura a ciertos objetos del clúster al usuario usado por la aplicación Prometheus. Ver Figura 4-24.
- **11-prom-config.yaml**. Configuración de Prometheus, tanto de las alertas de las que estará pendiente como de los *jobs* que irá lanzando para extraer información de los diferentes destinos. Ver Figura 4-25.
- **12-prometheus.yaml**. Despliegue y exposición interna de la aplicación Prometheus. Ver Figura 4-26.
- **13-prom-exponer.yaml**. Exposición externa al clúster de la aplicación Prometheus. Ver Figura 4-27.
- **20-kubestatemetrics-clusterrole.yaml**. Asignación de permisos al usuario que usa una de las fuentes de métricas en el clúster.
- **21-kubestatemetrics.yaml**. Instalación de la fuente de métricas mencionada en el punto anterior. Esta fuente estará pendiente de las operaciones de la API de kubernetes.
- **22-kubestatemetrics-serviceaccount.yaml**. Creación del usuario usado por la aplicación anterior.
- **23-node-exporter.yaml**. Instalación y exposición interna de otra fuente de métricas. Esta fuente extrae información de los nodos disponibles en el clúster.
- **30-alertmgr-configmap.yaml**. Configuración de la aplicación Alert manager. Ver Figura 4-38.

- **31-alertmgr-templates.yaml**. Plantillas por defecto para las notificaciones enviadas por alert manager. Ver Figura 4-41.
- **32-alertmgr.yaml**. Despliegue y exposición de la aplicación alert manager. Ver Figura 4-42 y Figura 4-43.
- **33-msteams-cfg.yaml**. Configuración del destino en MS Teams donde enviar las notificaciones de alert manager, así como el formato del mensaje que se envía.
- **34-msteams.yaml**. Despliegue y exposición interna de la aplicación para conectar con MS Teams.
- **40-grafana-datasource-configmap.yaml**. Configuración de las fuentes de datos para Grafana. Ver Figura 4-44.
- **41-grafana-dashboard-configmap.yaml**. Configuración para indicar a la aplicación Grafana donde encontrará los *dashboards* que tenga disponible. Ver Figura 4-45.
- **42-grafana-dashboard.yaml**. *Dashboard* preparado para la aplicación Grafana. Ver Figura 4-46.
- **43-grafana.yaml**. Despliegue y exposición interna de la aplicación Grafana. Ver Figura 4-48.
- **44-grafana-exponer.yaml**. Exposición externa de la aplicación Grafana. Ver Figura 4-49.

Instalación del *dashboard* de kubernetes

Este grupo se explica en el apartado 4.4.

- **dashboard-user.yaml**. Creación del usuario para el *dashboard* de kubernetes y asignación de este al rol de administrador en el clúster. Ver Figura 4-60.
- **dashboard.yaml**. Namespace, despliegue, exposición interna, secretos, así como otras configuraciones necesarias para el *dashboard* de kubernetes. Ver Figura 4-59.
- **exponer-dashboard.yaml**. Exposición externa del dashboard de kubernetes. Ver Figura 4-61.

F. Generación de imagen Docker para Raspberry Pi

Para la aplicación OIDC ha sido necesario desarrollar el código fuente tal como se ha mencionado en el punto 4.2.1.9, pero, además, una vez el código estaba listo había que generar una imagen Docker para poder ser desplegada dentro del clúster.

Una imagen Docker se define con un fichero, `Dockerfile`, que se compone de un conjunto de instrucciones que se deben llevar a cabo para poder compilar la imagen y arrancar la aplicación resultante. Para la aplicación OIDC, el fichero correspondiente se puede ver en la Figura F-1. En nuestro caso se trata de una imagen sencilla que parte de la imagen oficial de NodeJS para su versión 12, y sobre definimos y exponemos el puerto sobre el que la aplicación estará corriendo, así como copiar el código en la carpeta destino deseada y la instalación de dependencias, configurando por último cuál será el comando que se debe ejecutar cuando se desee iniciar la imagen.

```
FROM node:12-alpine

LABEL maintainer="Álvaro Téllez <atellez9@alumno.uned.es>" service="unedoidc"

ENV OIDC_PORT="${OIDC_PORT:-4400}"
ENV MY_HOME="${MY_HOME:-/atellez}"
ENV APP_CODE="${MY_HOME}/${APP_CODE_FOLDER:-tfm/code/unedoidc}"

COPY package*.json ${APP_CODE}/
COPY src ${APP_CODE}/src

RUN cd ${APP_CODE} && npm install --production

EXPOSE ${OIDC_PORT}

WORKDIR ${APP_CODE}

CMD ["npm", "run", "start"]
```

Figura F-1: contenido del fichero Dockerfile para la aplicación OIDC

Las imágenes Docker se deben compilar, y este proceso depende de la arquitectura donde se está trabajando. Así, aunque es sencillo generar una nueva imagen Docker desde el propio ordenador donde se estaba desarrollando, esto no nos valía en nuestro caso debido a que la imagen se generaba para la misma arquitectura del ordenador que generaba la imagen y sin embargo Raspberry tiene otra arquitectura (ARM). Para poder solventar este problema hemos generado la imagen para multi arquitectura (Docker, 2019) (Docker, 2020), usando

para ello la funcionalidad *buildx* ofrecida por Docker como plugin, y que aún no es parte de la instalación por defecto Docker, aunque seguramente lo sea en el futuro.

Una vez que tenemos dicho plugin instalado es suficiente con ejecutar el siguiente comando:

```
docker buildx build --platform linux/arm64,linux/arm/v7,linux/arm/v6 -t atellezr/uned-oidc --push .
```

Figura F-2: comando usado para la generación de la imagen Docker para otras arquitecturas

Este comando generará la imagen para las 3 arquitecturas indicadas: *linux/arm64*, *linux/arm/v7* y *linux/arm/v6*. Si bien es suficiente con *linux/arm/v7*, al ser la arquitectura de las Raspberry que usamos, usamos varias arquitecturas para tenerlas disponibles en caso de que fuese necesario y como muestra de que es posible hacerlo para varias arquitecturas al mismo tiempo. Hay que mencionar que es posible generar las imágenes para más arquitecturas y, de hecho, si se desarrolla alguna otra imagen pensando en que pueda ser usada en cualquier plataforma, se deberían tener en cuenta y generarlas para todas las plataformas disponibles en la herramienta. Lo único que supone el elegir un mayor número de arquitecturas es el tiempo que llevará la compilación para cada una de ellas.

Cuando se inicia el proceso de generación de imágenes multiplataformas, hay algunos pasos comunes que se ejecutan al principio y al final del proceso, y otros pasos que se ejecutan para cada imagen en particular. En las siguientes imágenes podemos ver un par de imágenes sobre el proceso de generación de la Figura F-3 y Figura F-4:

```
atellez-mac:uned-oidc atellez$ npm run docker:buildpush

> uned-oidc@0.0.1 docker:buildpush /Users/atellez/GoogleDrive/tfm/uned-oidc
> git pull && docker buildx build --platform linux/arm64,linux/arm/v7,linux/arm/v6 -t atellezr/uned-oidc --push .

remote: Counting objects: 10, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 8), reused 0 (delta 0)
Unpacking objects: 100% (10/10), done.
From https://bitbucket.org/atellezr/uned-oidc
   46ab22c..23e9980  master     -> origin/master
Updating 46ab22c..23e9980
Fast-forward
 src/lib/actions/interaction.js | 1 +
 src/lib/views/layout.js        | 7 +++++++
 src/views/_layout.ejs          | 7 +++++++
 3 files changed, 15 insertions(+)
[+] Building 12.4s (16/22)
 => [internal] booting buildkit
 => => starting container buildx_buildkit_mybuilder0
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [linux/arm64 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v6 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v7 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v6 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => [internal] load build context
 => => transferring context: 26.15kB
 => [linux/arm64 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => => resolve docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => [linux/arm/v7 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => => resolve docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => CACHED [linux/arm/v7 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => CACHED [linux/arm64 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => CACHED [linux/arm/v6 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => [linux/arm/v6 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm64 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm/v7 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm/v6 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => => # Unknown QEMU_IFLA_INFO_KIND ipip
 => => # Unknown QEMU_IFLA_INFO_KIND ip6tnl
 => [linux/arm64 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => => # Unknown QEMU_IFLA_INFO_KIND ipip
 => => # Unknown QEMU_IFLA_INFO_KIND ip6tnl
 => [linux/arm/v7 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => => # Unknown QEMU_IFLA_INFO_KIND ipip
 => => # Unknown QEMU_IFLA_INFO_KIND ip6tnl
```

Figura F-3: imagen extraída del proceso de generación de la imagen Docker multiplataforma donde se puede ver que se están instalando las dependencias de la aplicación NodeJS para las tres arquitecturas destino

```
> uned-oidc@0.0.1 docker:buildpush /Users/atellez/GoogleDrive/tfm/uned-oidc
> git pull && docker buildx build --platform linux/arm64,linux/arm/v7,linux/arm/v6 -t atellezr/uned-oidc --push .

remote: Counting objects: 10, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 10 (delta 8), reused 0 (delta 0)
Unpacking objects: 100% (10/10), done.
From https://bitbucket.org/atellezr/uned-oidc
   46ab22c..23e9980  master     -> origin/master
Updating 46ab22c..23e9980
Fast-forward
 src/lib/actions/interaction.js | 1 +
 src/lib/views/layout.js        | 7 +++++++
 src/views/_layout.ejs          | 7 +++++++
 3 files changed, 15 insertions(+)
[+] Building 234.9s (22/23)
 => [internal] booting buildkit
 => => starting container buildx_buildkit_mybuilder0
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 32B
 => [linux/arm64 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v6 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v7 internal] load metadata for docker.io/library/node:12-alpine
 => [linux/arm/v6 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => [internal] load build context
 => => transferring context: 26.15kB
 => [linux/arm64 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => => resolve docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => [linux/arm/v7 1/5] FROM docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => => resolve docker.io/library/node:12-alpine@sha256:9623cd396644f9b2e595d833dc0188a880333674488d939338ab5fde10ef7c43
 => CACHED [linux/arm/v7 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => CACHED [linux/arm64 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => CACHED [linux/arm/v6 2/5] COPY package*.json /atellez/tfm/code/unedoidc/
 => [linux/arm/v6 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm64 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm/v7 3/5] COPY src /atellez/tfm/code/unedoidc/src
 => [linux/arm/v6 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => [linux/arm64 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => [linux/arm/v7 4/5] RUN cd /atellez/tfm/code/unedoidc && npm install --production
 => [linux/arm/v7 5/5] WORKDIR /atellez/tfm/code/unedoidc
 => [linux/arm/v6 5/5] WORKDIR /atellez/tfm/code/unedoidc
 => [linux/arm64 5/5] WORKDIR /atellez/tfm/code/unedoidc
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:590fdc8a691a1b84a7f0d247c8cd66b4f834226908fafafbb759eabee99d7838
 => => exporting config sha256:48014f9cc4994cb91d0a8e73007776feef7727bf6a66117f2ee62f2781dbca2e
 => => exporting manifest sha256:2749dc00567601673c89cc31e8f5062c5290dbd4dfaf84210589863fff60e510
 => => exporting config sha256:a8683751b9c917e90d8ce56896e3fe4fa7bfe082de8bd8641f68cb30456df364
 => => exporting manifest sha256:92c2b45bc6b83ca6c3850df6b877bc8798e40ba1e5c02c9ba9a2e9163d1d42d6
 => => exporting config sha256:e4e14a07191383ff20e36c7c9bb8059d6f098e67e18bba871c8eb9d70066b363
 => => exporting manifest list sha256:48a5a877110592fb73e409ec3aba04caa3747061cee3599eba889a6c4dc6de31
 => => pushing layers
 => => pushing manifest for docker.io/atellezr/uned-oidc:latest
```

Figura F-4: imagen extraída del proceso de generación de la imagen Docker multiplataforma donde se puede ver que se ha compilado para las tres arquitecturas y la imagen resultante se está enviando al hub de Docker

Como resultado del proceso, una vez termina satisfactoriamente, tendremos nuestra imagen generada disponible en el *hub* de Docker (Figura F-5 y Figura F-6).

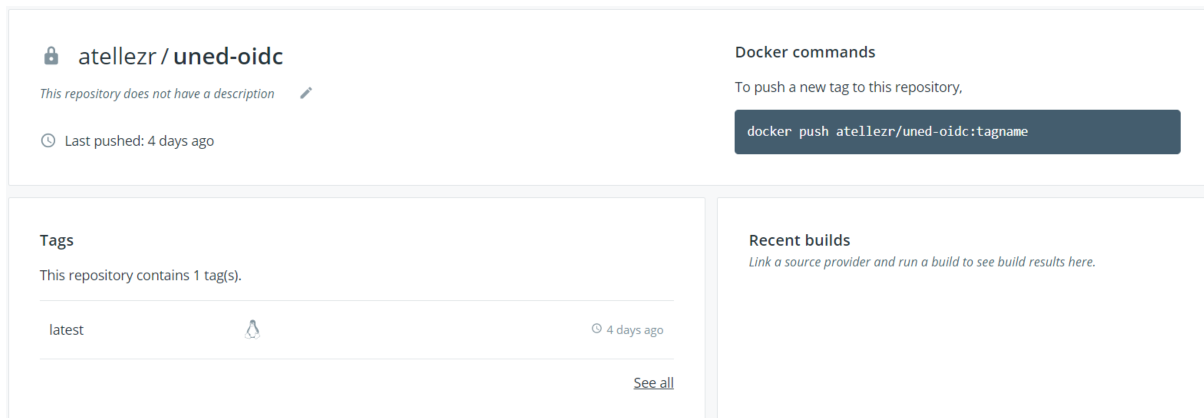


Figura F-5: imagen del proyecto para la aplicación OIDC dentro del hub de Docker

IMAGE
latest
Last updated **4 days ago** by atellezr

docker pull atellezr/uned-oidc:latest

DIGEST	OS/ARCH	COMPRESSED SIZE
92a59be74a33	linux/arm/v6	35.02 MB
d9afeeacfd55	linux/arm/v7	34.41 MB
15d82d1a5a4d	linux/arm64	35.82 MB

Figura F-6: imagen del hub de Docker donde vemos la última imagen de la aplicación OIDC que hemos publicado, y como ésta está disponible para las tres arquitecturas para la que la habíamos compilado

Como se puede ver (Figura F-6), es fácil comprobar dentro del *hub* de Docker para qué arquitecturas se ha generado la imagen.

Si bien este proceso indicado se ha seguido para la aplicación OIDC, es válido para la generación de cualquier imagen Docker que se haya desarrollado en un ordenador local y se quiera hacer disponible para Raspberry o cualquier otra arquitectura soportada.

Por último, hay que indicar que el proyecto utilizado es un proyecto privado dentro de Docker y por tanto sólo los usuarios autorizados podrán descargar e instalar la imagen Docker. Por tanto, para poder instalarla en el clúster será necesario configurar el usuario y contraseña de acceso al *hub* de Docker en forma de secreto, para lo cual hay que ejecutar un comando como el de la Figura F-7.

```
kubectl create secret docker-registry docker.io --docker-server=docker.io --docker-username=USUARIO --docker-password=CONTRASEÑA --docker-email=DIRECCION_DE_EMAIL -n NAMESPACE
```

Figura F-7: comando para la creación de un secreto en el clúster que contiene las credenciales para poder acceder a un repositorio privado del hub de Docker

El nombre que hemos dado al secreto creado en la Figura F-7, *docker.io*, se corresponde en este caso con el nombre del secreto que se carga cuando se instala la aplicación OIDC (ver Figura 4-6). De esta forma, cuando la aplicación aplicación OIDC vaya a iniciar una nueva réplica, recuperará este secreto antes de empezar, disponiendo así de las credenciales oportunas para poder descargar (*pull*) la imagen Docker.

G. Cómo se valida un token JWT

A modo de intentar mejorar el entendimiento sobre el funcionamiento de los tokens JWT, incluyendo el *id_token* usado para la autenticación en el clúster, vamos a mencionar los mecanismos usados para la correcta validación de un token JWT. Decir que el orden de las validaciones puede no corresponderse con el orden aplicado en kubernetes, y que aquí se explican para entender cómo funcionan.

Como se ha visto en el punto 4.2.1.6, un token JWT se compone de 3 secciones separadas por un punto. Cada sección está codificada en base64 para su seguridad en el envío de la información al evitar caracteres *raros* o que puedan cambiar según la codificación.

La primera sección contiene información sobre qué tipo de token estamos tratando y cómo se ha generado su firma. En el ejemplo mostrado, tenemos esta información:

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "r1LkbBo3925Rb2ZFFrKyU3MVex9T2817Kx0vbi6i_Kc"
}
```

Figura G-1: información sobre el tipo de token y cómo se ha generado su firma

Esto nos indica que estamos ante un token JWT cuya firma se ha generado con el algoritmo *RS256* (algoritmo asimétrico usando RSA y SHA-256) usando la clave cuyo identificador es el mencionado en *kid* (*key ID*).

Con esta primera sección ya somos capaces de validar que el token está tal cual se generó y que nada/nadie lo ha modificado. Para ello, nos vamos a la URL relativa a los jwks (ver punto 4.2.1.3 para más información), donde podemos encontrar la clave asociada a ese identificador (Figura G-2).

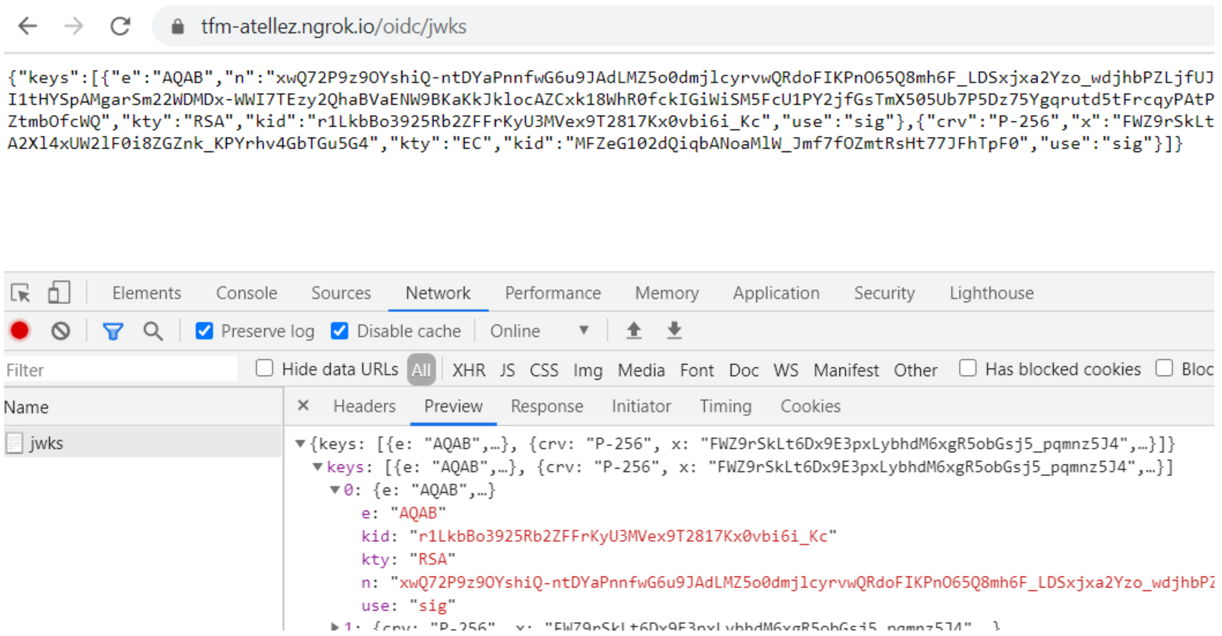


Figura G-2: parte de la información pública asociada a la clave usada para generar el token

La información que aquí se muestra es información pública, y así pues se puede acceder a la misma sin necesidad de aplicar ningún mecanismo de seguridad. En concreto, debemos verificar que exista una clave con el identificador mostrado en el campo *kid*, cuyo tipo sea RSA y su uso sea apto para firmas (ver campos *kty* y *use* de la imagen Figura G-2). Si no se encontrase una clave que encaje en estos requisitos, la firma no podría validarse y por tanto se daría por inválido el token. En este caso, encontramos la clave pública (campo *n*).

Para validar la firma únicamente debemos verificar, usando la clave pública asociada, que la cadena de texto relativa a las 2 primeras secciones (incluyendo los puntos) generan la firma dispuesta en la tercera sección. Puesto que únicamente el emisor, la aplicación OIDC en este caso, dispone de la clave privada, únicamente es él quien puede generar una firma acorde al contenido que estamos validando, por lo que si la firma es válida podemos dar por hecho que quien ha emitido el token ha sido dicho emisor. Y por este motivo es crucial que las claves privadas se gestionen de manera segura. Si una clave privada se viese comprometida se debería eliminar/regenerar cuanto antes en la aplicación OIDC, para que así cambie la clave pública asociada y los tokens previamente generados pasen automáticamente a ser inválidos.

Las siguientes validaciones por realizar tienen que ver con campos definidos en la segunda sección del token (ver Figura 4-16), y en concreto:

- El token aún está activo, lo que se valida con el campo *exp*, que representa el tiempo (en segundos) en el que el token caducará. El emisor calcula este valor como su tiempo actual en el momento de la generación más el tiempo de validez que decida aplicar. La aplicación que verifica el token, kubernetes en este caso, debe comprobar que este valor sea menor que su tiempo actual. Y se asume que ambas partes usan un reloj correctamente sincronizado (por ejemplo, usando servidores NTP).
- La audiencia, o para quién se ha generado el token, es la correcta. Esto quiere decir que el token se haya generado para la aplicación que lo está verificando. Si bien no es nuestro caso, supongamos que la aplicación OIDC hubiese generado el token para otra aplicación, kubernetes vería que la audiencia no se corresponde con la esperada y, aunque el resto de las validaciones fuesen correctas, rechazaría el token.

Se pueden implementar más validaciones de seguridad, pero para los tokens JWT usados como *id_token* estas medidas son suficientes.