Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua

Julio Aladrén García
Grado en Telecomunicaciones Telemática

Manuel Jesús Mendoza Flores
Consultor

Data Entrega

FICHA DEL TRABAJO FINAL

Título del trabajo:	Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua
Nombre del autor:	Julio Aladrén García
Nombre del consultor:	Manuel Jesús Mendoza Flores
Fecha de entrega (mm/aaaa):	01/2020
Área del Trabajo Final:	Administración de redes y sistemas operativos.
Titulación:	Grado en Telecomunicaciones Telemática

Resumen del Trabajo (máximo 250 palabras):

Desde 2013, Docker y posteriormente Kubernetes comenzaron a ganar popularidad, empezando así, a postularse como tecnologías prometedoras que con el paso del tiempo han ido teniendo apoyo de compañías como Microsoft, Red Hat y Google, entre otras. Esto es debido a que sirve de base al paradigma de la arquitectura basada, en microservicios, la cual ofrece grandes ventajas con respecto a anteriores arquitecturas. Entre dichas ventajas podemos destacar la facilidad de escalado, la disgregación del modelo monolítico en diferentes piezas funcionales mantenidas independientemente y el funcionamiento eminentemente stateless entre otras.

Ahondando en Docker y Kubernetes y los hipervisores asociados, los cuales se convierten en referentes tecnológicos base de todo lo descrito anteriormente, aparecen nuevos retos que deben de ser abordados desde nuevas perspectivas. Es por tanto que el objeto de este trabajo es ahondar en los mecanismos de securización de Docker y Kubernetes desde el punto de vista de los hipervisores asociados, con la intención de definir un *baseline* basada en Anchore que permita la securización de entornos en los que se haga uso de ellos.

Abstract (in English, 250 words or less):

Since 2013, Docker and later Kubernetes began to gain popularity and thus began to postulate as promising technologies that over time have been supported by companies such as Microsoft, Red Hat and Google, among others. All of the above serves as a basis of paradigm of architecture based on microservices, which offers significant advantages over previous architectures. Some of these advantages are easy scalation, the disgregation of the monolithic model into different functional pieces and in the eminently stateless operation of the services, among others.

Deeper into Docker and Kubernetes and the associated hypervisors, which become technological references based on everything described above. It is, for this reason, lead to new challenges appear that must be approached from new perspectives. The object of this work is to delve into Docker and Kubernetes security mechanisms from the associated hypervisor's perspective, defining a baseline that allows the security of environments in which they are in use.

Palabras clave (entre 4 y 8):

Docker.
Kubernetes.
Anchore.
Rancher.
Jenkins.
Continuous Integration.
DevSecOps.

Índice

1 Lista de figuras

2 Lista de tablas

3 Introducción

3.1 Propósito

El propósito de este trabajo de final de grado, es el de continuar ahondando en el conocimiento a través de la autoformación, con la idea de obtener como resultado, una base para la definición de recomendaciones de *hardening* que permitan a los sistemas aquí tratados, operar de una forma más segura.

3.2 Descripción de la estructura del Documento

A continuación, se muestra la estructura básica del Documento:

- **Sección 1:** Contiene el propósito de este documento, el resumen general de las secciones del documento, las definiciones, las siglas, y las convenciones utilizadas en el documento.

- **Sección 2:** Contiene el contexto y la justificación del trabajo, los objetivos del trabajo, el enfoque y método seguidos, la planificación del trabajo y un análisis de Riesgos.

- **Sección 3:** Contiene la descripción de las distintas componentes tecnológicas de las que se harán uso en el proyecto.

- **Sección 4:** Contiene una descripción a alto nivel del análisis de vulnerabilidades con Anchore en clústeres de Kubernetes, así como dentro de entornos de desarrollo con integración continua basados en Jenkins.

- **Sección 5:** Contiene los distintos anexos e información complementaria de soporte al proyecto.

- **Sección 6:** Contiene la bibliografía separada en documentos de aplicación y de referencia.

4 Definiciones y abreviaciones

4.1.1 Acrónimos

La siguiente tabla alberga los acrónimos mencionados en el documento.

Acrónimo	Descripción
AD	*Aplication Document*
API	*Application Programming Interface*
CI	*Continuous Integration*
CIS	*Center of Internet Security*
CLI	*Command Line Interface*
CVE	*Common Vulnerabilities and Exposures*
CVSS	*Common Vulnerability Scoring System*
DB	*Database*
Def.	Definición
DNS	*Domain Name System*
EPEL	*Extra Package for Enterprise Linux*
GB	*Gigabyte*
IDS	*Intrusion Detection System*
ISBN	*International Standard Book Number*
IT	Tecnología Informática
JSON	*Java Script Object Notation*
NAT	*Network Address Translation*
OS	*Operating system*
RAM	*Random Access memory*
RD	*Reference Document*
Ref.	Referencia
REST	*Representational state transfer*
RKE	*Rancher Kubernetes Edition*
SHA256	*Secure Hash Algorithm 256 bits*
URL	*Uniform Resource Locator*
VM	*Virtual Machine*
YUM	*Yellowdog Updater, Modified*

Tabla 1: Acrónimos

4.1.2 Definiciones

La siguiente tabla muestra definiciones de aplicación que pueden ser de ayuda en el contexto de este documento.

Termino	Descripción
Baseline	Definición de un punto de partida, a partir del cual se podrá establecer a construir un sistema a partir de las bases o definiciones expuestas. [RD-01]
Clúster	Conjunto de servidores que interconectados entre si se comportan como uno u ofrecen un único servicio. [RD-02]
Commit	Acción de aplicar o guardar un cambio.
Contenedor	Empaquetado *software* que incluye todo lo necesario para que la aplicación sea capaz de correr en un contexto heterogéneo de alto nivel. Los Contenedores además de incluir las librerías y configuraciones básicas ejecutan micro *kernels* de Linux en su mayoría. [RD-03]
COTS	*Commercial off-the-shelf* termino formal utilizado para listar los distintos ítems comerciales usados.
CPU	Elemento *hardware* que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema. [RD-04]
Debug	Proceso de buscar y corregir problemas dentro del ámbito IT
Gate	Cada una de las distintas categorías sobre las que el aplicativo Anchore puede realizar sus análisis.
Hardening	Proceso a través del cual se asegura un sistema reduciendo sus vulnerabilidades a nivel de seguridad. [RD-05]
Hipervisor	Aplicado al contexto de Kubernetes, aplicativo que permite la administración heterogénea tanto de los nodos que forman el clúster, así como la administración de distintos clústeres independientemente de la tecnología que utilicen o de donde se encuentre alojados. [RD-06]
Integración Continua (CI)	Modelo de desarrollo a través del cual se hacen integraciones manuales o automáticas de un código o proyecto, con una alta periodicidad. Facilitando el evidenciado de fallos que podrán solventados cuanto antes.[RD-07]
Pod	Se denomina *pod* a todos aquellos contenedores interrelacionados y desplegados juntos dentro de un mismo host, compartiendo *namespace*, volúmenes y *secrets*. [RD-08]
Policy	Conjunto de evaluaciones y *triggers* que en conjunto durante la fase de análisis permiten obtener una evaluación del cumplimiento de un con tenedor con respecto a nuestra política.
Push	Acción de subir o guardar.
Rollback	Vuelta a un punto de configuración anterior, que ha sido salvaguardado previamente. [RD-09]
Root	Usuario administrador común en sistemas Linux y Unix.
Snapshoot	Copia de seguridad un sistema determinado en un punto de tiempo particular. [RD-10]
Trigger	Mecanismo evaluador sobre una configuración específica, que permite detectar y desencadenar otras acciones en función del resultado de la evaluación sobre la configuración analizada.
Workstation	Computador de sobremesa con prestaciones y características avanzadas.

Tabla 2: Términos

4.2 Notaciones y convenciones

Con el fin de aclarar qué es un requisito y qué es una recomendación, en el presente documento se utilizan las siguientes formas verbales:

- Los términos "DEBERÁN" y "NO DEBERÁN" indican requisitos obligatorios.
- Los términos "DEBERÍA" y "NO DEBERÍA" indican recomendaciones (preferiblemente pero no necesariamente requeridas).
- Los términos "PUEDE" y "NO PUEDE" indican una posibilidad o capacidad.

5 Contexto y justificación del Trabajo

En la actualidad las arquitecturas basadas en microservicios se han convertido en una realidad, con gran apoyo por parte de las grandes empresas tecnológicas de IT, esto unido a las diferentes bases tecnológicas que propician su implementación, como son Docker Kubernetes y sus hipervisores, ha dado lugar a que multitud de empresas estén migrando a nuevas arquitecturas basadas en microservicios.

Dada la gran envergadura del cambio a arquitecturas basadas en microservicios, auspiciada principalmente por Docker y Kubernetes. Hará que ambas tecnologías se conviertan irremediablemente en un gran foco de interés para a nivel de ataques ciberneticos. Esto es debido principalmente a dos motivos:

El primero de ellos, está circunscrito al contexto de una única compañía y se basa en que un atacante con la habilidad de comprometer dichas tecnologías obtendrá acceso, no solo a la aplicación comprometida, si no que tendrá acceso al resto de microservicios que estén corriendo en los contendedores adyacentes al microservicio afectado. Por otro lado, será capaz de comprometer las comunicaciones que discurran entre los microservicios albergados en el clúster.

El segundo motivo dentro de un ámbito más amplio, hace referencia a que un atacante con la habilidad de comprometer una tecnología ampliamente difundida, será capaz de comprometer un gran número de empresas, haciendo uso del mismo mecanismo de ataque.

A mención de las diferentes afecciones que pueden ser resultado de un ataque satisfactorio. Podemos destacar:

- Inhabilitación total o parcial de los procesos de negocio.
- Perdidas monetarias, extorsión, espionaje industrial.
- Publicación de datos privados.
- Así como afectar a la imagen corporativa de la compañía.

Acaecido todo lo anterior, este trabajo pretende establecer un *baseline* de seguridad para el análisis estático de imágenes de Docker sobre un entorno basado en Kubernetes.

5.1 Objetivos del Trabajo

Dentro del alcance del proyecto y de los objetivos conseguidos podemos destacar:

- Analizar la arquitectura y funcionamiento de Docker y Kubernetes, así como las diferentes técnicas de *hardening* aplicables.

- Análisis de imágenes dentro del contexto de la seguridad basándose en la *suite* de seguridad Anchore. Incluyéndolo a su vez como gestor del control de admision en clústeres de Kubernetes.

- Definición de un entorno de integración continua basado en *DevSecOps* haciendo uso de la integración de Jenkins con Anchore.

5.2 Enfoque y método seguido

El presente trabajo estará dividido en dos partes:

La primera de ellas, basándose en el analizador de imágenes Anchore, intentará introducirlo y configurarlo en un clúster de Kubernetes. Permitiendo así, que este se encargue de analizar los contenedores previos a su ejecución, evitando de este modo, que puedan ser ejecutadas imágenes. Que contienen vulnerabilidades o no cumplen con la política, haciendo uso de un control de admisión en el propio clúster de Kubernetes.

La segunda parte, abordará las posibilidades que ofrece Anchore para el análisis de vulnerabilidades en Entornos basados en *DevSecOps*, con integración continua basada en Jenkins integrado con Anchore.

5.3 Planificación del Trabajo

Este Apartado, tiene como objetivo realizar un estudio de planificación, en el que se irán detallando la distribución temporal de cada uno de los hitos del proyecto, junto a una breve descripción. Al final del apartado, mediante un diagrama de Gantt, se mostrará la distribución temporal y la interdependencia de las tareas.

Proyecto: Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua	**Fecha Inicio:** 18/09/2019
Hito 1: Estudio de Docker, Kubernetes, Rancher y Jenkins	**Fecha Fin:** 5/10/2019
Descripción: Análisis de las distintas tecnologías y de la viabilidad de estas, a la hora de incluir dentro del proceso de *hardening* de contenedores en sistemas basados en CI.	**Task 1.1:** Análisis de las distintas tecnologías.

Tabla 3: Hito 1 - Estudio de Docker, Kubernetes, Rancher y Jenkins

Proyecto: Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua	**Fecha Inicio:** 6/10/2019
Hito 2: Creación entorno de pruebas	**Fecha Fin:** 19/10/2019
Descripción: En este hito, se llevarán a cabo todas las tareas relacionadas con la creación del entorno de laboratorio, necesario para el desarrollo del proyecto.	**Task 2.1:** Instalación Docker, Clúster de Kubernetes e Hipervisor Rancher. **Task 2.2:** Instalación Anchor.

Tabla 4: Hito 2 - Creación Entorno de pruebas

Proyecto: Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua	**Fecha Inicio:** 20/10/2019
Hito 3: Análisis del funcionamiento de Anchor dentro de un CI y Control de admisión en Kubernetes	**Fecha Fin:** 28/12/2019
Descripción: En este Hito, se profundizará en el funcionamiento de Anchore, se realizarán análisis de contenedores, modificación de políticas, integración con Jenkins y control de admisión en clústeres de Kubernetes.	**Task 3.1:** Análisis de imágenes del registro. **Task 3.2:** Definición de políticas. **Task 3.3:** Integración con Jenkins, así como Integración dentro de un clúster de Kubernetes para evitar la ejecución de un contenedor vulnerable, integración con Jenkins

Tabla 5: Hito 3 - Análisis del funcionamiento de Anchor dentro de un CI

Proyecto: Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua	**Fecha Inicio:** 29/12/2019
Hito 4: Evaluación de los resultados	**Fecha Fin:** 5/01/2020
Descripción: Durante este hito, se analizarán los resultados obtenidos, se hará retrospectiva de todo lo aprendido y se describirá los posibles casos de aplicación	**Task 4.1:** Retrospectiva y evaluación de resultados. **Task 3.2:** Lecciones aprendidas.

Tabla 6: Hito 4 - Evaluación de los resultados

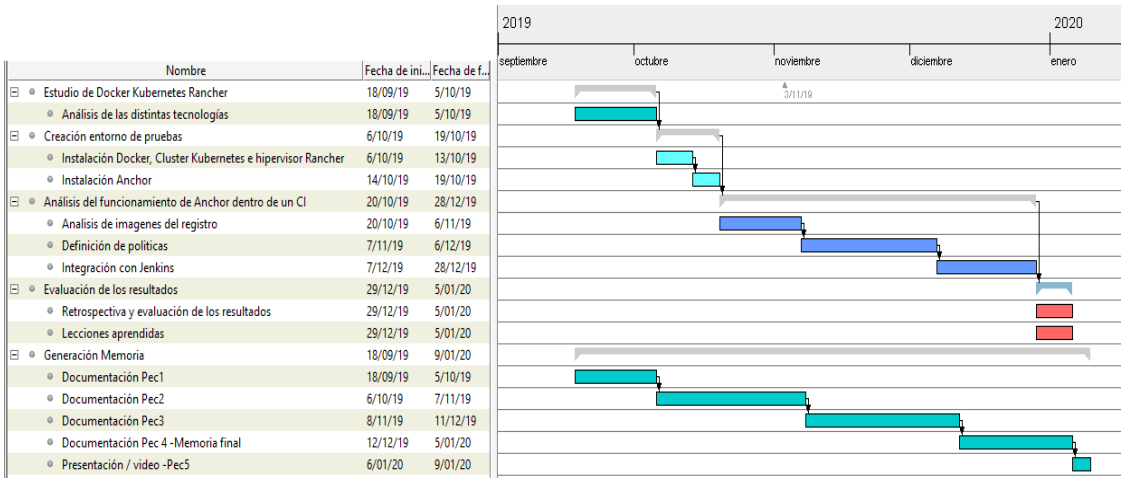


Ilustración 1: Diagrama de Gantt planificación proyecto

.

5.4 Breve sumario de productos obtenidos

Los hitos que se esperan alcanzar a la finalización del proyecto son:

- Conocimientos del funcionamiento base de Docker y Kubernetes.
- Conocimientos del funcionamiento del hipervisor Rancher.
- Establecimiento de una línea guía de seguridad:
    - *Assesment* de seguridad en contenedores ya desplegados.
    - Permitir que previo a cualquier ejecución de un contenedor Anchore evite su ejecución si es vulnerable.
    - Estudio de las posibilidades que ofrece Anchore dentro de entornos de integración continua, que hagan uso de la filosofía *DevSecOps*, basados en Jenkins.
- Conocimientos sobre las distintas tecnologías que permiten el análisis de vulnerabilidades en los contenedores.
- Conocimientos de *hardening* base de Dockers y Kubernetes.

5.5 Análisis de Riesgos

En las siguientes tablas, se referencian los diferentes riesgos detectados durante el análisis de riesgos, así como, las acciones mitigadoras que se llevaran a cabo en caso de que, durante el desarrollo de algunos de los hitos, se encuentren problemas que puedan tener un impacto relevante en la consecución de este.

Código	Descripción	Causa	Probabilidad	Impacto
R01	Fallo del clúster de Kubernetes	Fallo de configuración	Media	Alto
R02	Fallo del servidor/*host*	Fallo *hardware*	Baja	Alto
R03	Fallos derivados de las configuraciones de *hardening* de seguridad en Kubernetes	Fallo de Configuración	Media	Medio
R04	Fallos derivados de las configuraciones de *hardening* realizadas con Anchore	Fallo de Configuración	Media	Medio
R05	Fallo de Red derivado de VMware *Network Driver*, interconexión entre máquinas virtuales del clúster	Fallo *Software*	Baja	Alto
R06	Dificultades en la consecución de ciertos hitos por inexperiencia sobre las tecnologías trabajadas	Falta de experiencia	Media	Alto

Tabla 7: Riesgos

Código	Acción	Tipo	Riesgo post Mitigación
A1R01	*Snapshoot* de las máquinas virtuales	Mitigadora	Bajo
A2R02	Servidor alternativo	Mitigadora	Bajo
A3R03	Rollback a las configuraciones de *hardening* aplicadas en Kubernetes	Correctora	Bajo
A4R04	Rollback a las configuraciones de *hardening* aplicadas en Anchore	Correctora	Bajo
A5R05	Reinstalación Driver VMware Network Driver	Mitigadora	Bajo
A6R06	Se usarán métodos de autoformación y se consultara con la comunidad de la tecnología en referencia	Correctora	Medio

Tabla 8: Plan de contingencia

5.6 Breve descripción de los otros capítulos de la memoria

Con la pretensión de introducir al lector en la consistencia y desarrollo de la memoria de proyecto a continuación se presenta una descomposición en capítulos del proyecto:

Descripción de las componentes tecnológicas:
Este capítulo se encargará de introducir al lector en las distintas componentes tecnológicas de las cuales se hará uso durante el desarrollo del proyecto, siempre desde un enfoque introductorio que permita la compresión de los capítulos posteriores con mayor facilidad.

Diseño y arquitectura propuesta:
A lo largo del capítulo se hará referencia a dos propuestas de arquitectura. La primera de ellas como propuesta funcional a nivel arquitectónico para la implementación de clústeres de Kubernetes en entornos corporativos teniendo como foco la seguridad. La segunda propuesta arquitectónica es una adaptación de la primera, para facilitar su implementación en un entorno reducido de laboratorio.

Configuración de Anchore:
Este capítulo se encargará de albergar todas aquellas definiciones y desarrollos a nivel de configuración necesarias para que permitan al lector continuar con el desarrollo del texto. El desarrollo de distintas configuraciones a nivel de montaje y de forma extendida se detalla en la documentación anexa al final del proyecto.

Configuración de una imagen testigo vulnerable:
Este capítulo aborda la creación de imágenes de Docker que contengan vulnerabilidades, permitiendo así, posteriormente comprobar las distintas funcionalidades de Anchore.

DevSecOps, uso de Anchore dentro de entornos de integración continua basados en contenedores:
Este capítulo aborda como Anchore puede ser integrado junto a distintos softwares de automatización de despliegues en entornos de integración continua para facilitar la orientación hacia entornos orientados a *DevSecOps*

Control de Admisión en Kubernetes basado en Anchore:
Este capítulo aborda la configuración de un control de admisión en clústeres de Kubernetes con respecto a una política de seguridad definida que permita la admisión o descarte en la ejecución de contenedores vulnerables.

Conclusiones:
A lo largo del desarrollo de este capítulo se tratará el grado de madurez y aprendizaje obtenido tras la ejecución del mismo, así como, posibles puntos de mejora para ser tratados en posteriores revisiones o propuestas de proyectos.

Anexos:
Este capítulo contiene toda la información de referencia necesaria para la implantación de proyecto. Así como, distintas configuraciones necesarias para su ejecución, que por su extensión y para evitar la pérdida del foco sobre el objeto del proyecto han sido postergadas al anexo.

Bibliografía:

Este capítulo está dividido en dos secciones albergando así todas aquellas referencias bibliográficas necesarias para el desarrollo del documento. Contiene documentos de aplicación [AD-XX] que son necesarios y permiten ahondar en detalles técnicos y de instalación de cada una de las componentes tecnológicas. Por otro lado los documentos de referencia [RD-XX] hacen mención de todos aquellos documentos que o bien han servido de base documental para el desarrollo del proyecto o son de especial interés para ampliar conocimientos.

6 Descripción de las componentes tecnológicas

A lo largo de este capítulo se abordarán y definirán las diferentes tecnologías que servirán de base para la ejecución del proyecto, proporcionando así, una descripción pormenorizada de cada una de ellas desde un punto de vista individual. La pretensión de este capítulo introductorio es servir de base de conocimiento prescriptiva, para poder abordad de una forma más sencilla los siguientes capítulos.

En capítulos posteriores se abordará como a través de la arquitectura definida, las diferentes componentes tecnológicas descritas individualmente en este capítulo, como interactúan y se relacionan entre sí, para proporcionar un servicio de análisis de vulnerabilidades en contenedores, previniendo la ejecución de estos en caso de que no cumplan con ciertas políticas de seguridad definidas.

6.1 Docker

Docker es un proyecto de código abierto, que a nivel tecnológico ha brindado a la comunidad una nueva aproximación al concepto de virtualización. Permitiendo desplegar y ejecutar aplicaciones dentro de contenedores de software, haciendo uso de un único sistema operativo anfitrión. Para ello, Docker hace uso de las características de aislamiento de recursos aportadas por el kernel de Linux, tales como los *namespaces* y los cgroups.

Namespaces: Permiten aislar la visibilidad que tiene una aplicación de su entorno operativo. Esto incluye aislamiento del árbol de procesos, ID de usuario, sistema de ficheros y puntos de montaje, así como, la propia red. Esto proporciona un aislamiento entre la propia aplicación y el sistema operativo en el que está corriendo.

Cgroups: Permiten un asilamiento de la aplicación a nivel de recursos máquina, Asignando restricciones y cuotas a nivel de *CPU*, memoria *RAM*. Así como, escritura y lectura a nivel de disco y de red que pueden ser consumidos por la aplicación.

Para contextualizar este nuevo paradigma definido por Docker en el que las aplicaciones corren dentro de contenedores completamente independientes, debemos de compararlo con otros paradigmas de virtualización actualmente líderes en el mercado, como son las máquinas virtuales.

6.1.1 Comparativa arquitectónica entre Docker y Máquina virtual

Las máquinas virtuales permiten la abstracción del hardware físico, por su parte el hipervisor permite que varias máquinas virtuales se ejecuten en un solo equipo. Cada *VM* incluye una copia completa de un sistema operativo, la aplicación, los binarios y las bibliotecas necesarias, que ocupan decenas de *GB*. Las máquinas virtuales también pueden ser lentas al arrancar debido a que deben arrancar un sistema operativo por completo.

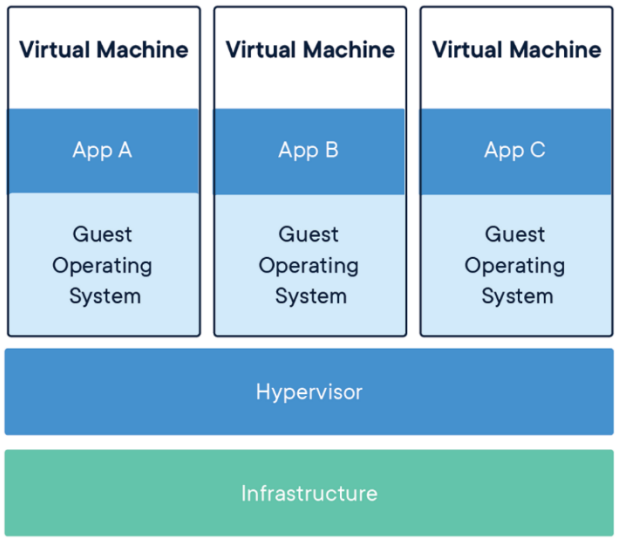


Ilustración 2: Arquitectura basada en máquinas virtuales

Los contenedores son una abstracción varios niveles por encima de la abstracción *hardware* de las máquinas virtuales. En los contenedores se abstrae la capa de aplicación empaquetándose código y dependencias juntos en un contenedor. Varios contenedores pueden ejecutarse en la misma máquina y compartir el núcleo del sistema operativo con otros contenedores. Cada uno de los cuales se ejecuta como procesos aislados en el espacio de usuario. Entre las ventajas de los contenedores podemos destacar que ocupan menos espacio que las máquinas virtuales y requieren menos recursos para poder ser ejecutados, ya que por lo general no requieren de los recursos necesarios que harían falta para correr una máquina virtual y su sistema operativo completos.

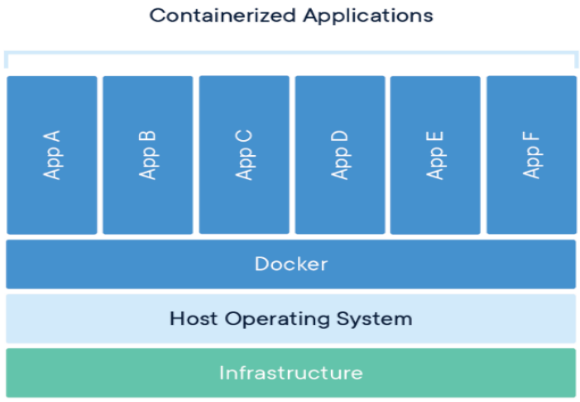


Ilustración 3: Arquitectura basada contenedores

En el anexo Instalación de Docker sección 13.1 puede encontrar información detallada sobre la instalación de Docker. La siguiente referencia bibliográfica tiene como objetivo permitir al lector ampliar conocimientos sobre Docker [RD-11].

6.2 Kubernetes

Kubernetes es una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones. Kubernetes agrupa los contenedores que conforman una aplicación en unidades lógicas para una fácil administración.

6.2.1 Descripción de los componentes de Kubernetes

Componentes de un clúster de Kubernetes:

El siguiente diagrama contiene los distintos componentes básicos de un clúster de Kubernetes, los cuales serán explicados posteriormente.

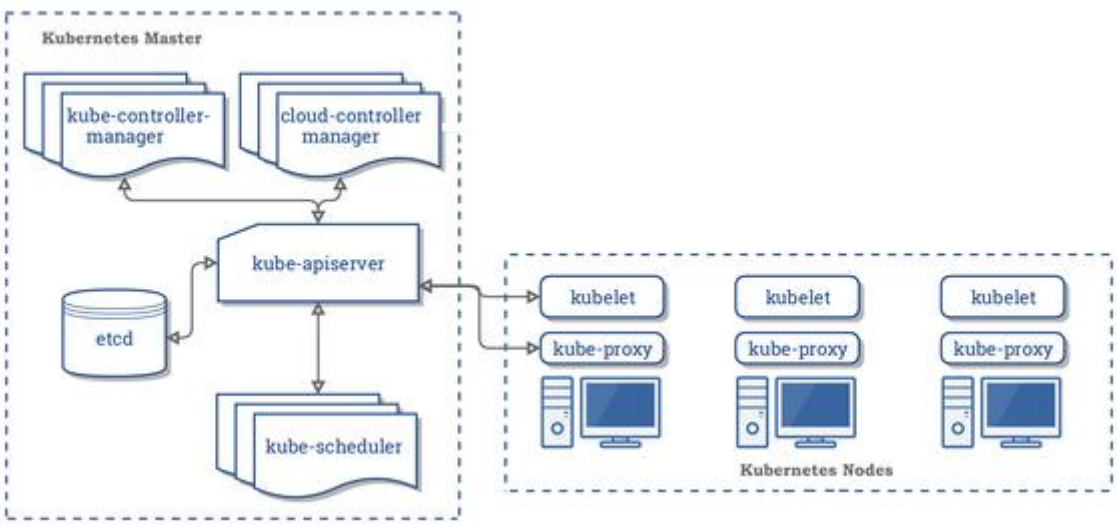


Ilustración 4: diagrama de componentes de un clúster de Kubernetes

Procesos destacables dentro de los nodos máster:

A continuación se hace referencia a toda la serie de procesos que se ejecutan en el nodo *máster* y de los cuales es conveniente tener una descripción para comprender el funcionamiento de un clúster de Kubernetes.

kube-apiserver:

El servidor de la *API* de Kubernetes valida y configura los datos de los objetos *API* que incluyen *pods*, servicios, controladores y todo aquello que esté relacionado con el propio clúster. El servidor *API* presta servicio a las operaciones *REST* y proporciona un *front-end* al clúster, a través del cual interactúan todos los demás componentes.

Kube-controller-mánager:

Demonio que incorpora los bucles de control del núcleo. Podemos definir un bucle de control como aquel bucle que nunca termina y que regula el estado del sistema. Kube-Controller-Manager dentro de Kubernetes es un bucle de control que observa el estado compartido del clúster a través del *API* y realiza cambios intentando pasar desde estado actual del clúster hacia el estado deseado.

Kube-scheduler:

El módulo de planificación de Kubernetes se encarga de la repartición de la carga de trabajo, controla también mantiene la afinidad entre *pods* para mejorar el desempeño del clúster. Así como, la localización de los datos, de esta forma es capaz de planificar el trabajo dentro del clúster, dentro de unos valores adecuados, que permiten al clúster funcionar de forma correcta.

Etcd:

Servicio de almacenamiento de datos del clúster, encargado de mantener toda la información de estado del clúster y su configuración, en clústeres de gran tamaño *etcd* puede estar distribuido entre varios nodos que no necesariamente deben ser nodos master del propio clúster.

Procesos destacables dentro de los nodos worker:

A continuación se hace referencia a toda la serie de procesos que se ejecutan en el nodo *worker* y de los cuales es conveniente tener una descripción para comprender el funcionamiento de un clúster de Kubernetes.

Kubelet:

Se considera como el proceso primario que corre dentro de cada nodo *worker*, es el encargado de gestionar la adhesión del nodo al clúster, además de ser el encargado de mantener al clúster informado sobre los diferentes pods y *workloads* que está corriendo dentro del propio nodo.

Kube-Proxy:

El módulo de proxy se encarga de hacer de proxy de red para la gestión y balanceo de los distintos flujos de red. Parte de estos flujos irán destinados a *pods* que corren dentro del propio nodo y otros serán redireccionados al clúster para que sean gestionados por otros nodos *worker* los cuales contenga en ejecución los pods destino del tráfico de red.

Container engine runtime:

Hace referencia al propio entorno de ejecución de contenedores. Para el desarrollo de este proyecto se ha utilizado Docker el cual es explicado en el capítulo 6.1.

Kubernetes API:

Mediante la *API* de Kubernetes o mediante Kubectl, se pueden mandar instrucciones para interactuar con el clúster y especificar o modificar su estado (arrancar *pods*, arrancar replicas, parar nodos, etc). Una vez que se específica el estado deseado, el plano de control de Kubernetes realizará las acciones deseadas para ejecutar las acciones requeridas y que el clúster funcione según las órdenes dadas.

La siguiente referencia bibliográfica contiene información detallada sobre kubernetes [RD-12].

6.2.2 RKE

Rancher Kubernetes Engine (*RKE*) es una distribución de Kubernetes certificada por la Cloud Native Computing Fundation, su principal ventaja es la de funcionar enteramente dentro de contenedores Docker. Al funcionar como contenedores el servidor donde será instalado, únicamente requiere tener instalado Docker, solucionando así el problema que supone desplegar y configurar Kubernetes. Esto supone una mejora en la automatización del despliegue de nodos y la propia instalación del clúster de Kubernetes.

Para facilitar la puesta en marcha de entornos de laboratorio se ha utilizado RKE como solución de Kubernetes. En el anexo Instalación clúster de Kubernetes basado en RKE sección 13.2 puede encontrarse detalles sobre la instalación de Kubernetes en [RD-13].

6.3 Rancher

Rancher es un proyecto de software libre que permite la orquestación y administración sencilla de clústeres de Kubernetes. Una de sus mayores ventajas es poder administrar clústeres de Kubernetes independientemente de si son *On premise* o si se encuentran desplegados en *Cloud*.

Actualmente se ha convertido en una de las soluciones de hipervisor lideres para la orquestación de clústeres de Kubernetes.

6.3.1 Descripción de los componentes de Rancher

El siguiente esquema muestra de manera simplificada los distintos componentes de Rancher. Un servidor de Rancher se puede dividir comúnmente en 5 bloques funcionales como son, Clúster Controller, Clúster Agent, Rancher *API* server, Auth Proxy y etcd. Los cuales detallaremos posteriormente.

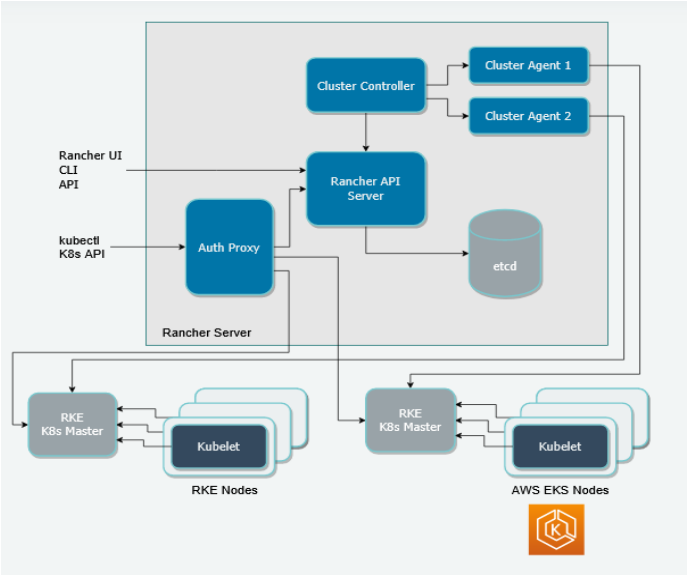


Ilustración 5: Esquema componentes de Rancher

Rancher API server:

se constituye como una capa por encima del Kubernetes *API server* así como de Kubernetes *etcd* proporcionando las siguientes funcionalidades:

- Gestión de usuarios: El servidor Rancher *API* gestiona las identidades de usuario que corresponden a proveedores de autenticación externos como Active Directory o GitHub.
- Control de acceso: El servidor Rancher *API* gestiona el control de acceso y las políticas de seguridad.
- Proyectos: Un proyecto es un grupo de múltiples espacios de nombres y políticas de control de acceso dentro de un clúster.
- Nodos: El servidor Rancher *API* rastrea las identidades de todos los nodos en todos los clústeres.

Clúster Controller y Clúster Agents:

Se encargan de implementar toda la lógica que permite orquestar y administrar los distintos clústeres de Kubernetes que este gestionando Rancher:

- El controlador de clúster: pone en marcha la lógica necesaria para la instalación global de Rancher. Realiza las siguientes acciones:

- Configuración de políticas de control de acceso a clústeres y proyectos.

- Aprovisionamiento de clústeres mediante llamada, ya sean del tipo que sean AWS, Google Kubernetes Edition, Rancher Kubernetes Edition o directamente gestionando las instancias de Docker.

- Los agentes: pone en funcionamiento la lógica requerida para el clúster correspondiente. Realiza las siguientes actividades:

    - Gestión de la carga de trabajo, como la creación e implementación de pods dentro de cada clúster.

    - Aplicación de los roles y vínculos definidos en las políticas globales de cada clúster.

    - Comunicación entre clústeres y servidor de Rancher: eventos, estadísticas, información de nodos y salud.

Proxy de autenticación:

El proxy de autenticación reenvía todas las llamadas a la *API* de Kubernetes. Se integra con servicios de autenticación como, autenticación local, Active Directory y GitHub. En cada llamada a la *API* de Kubernetes, el proxy de autenticación, valida a la persona que llama y establece las cabeceras de suplantación de Kubernetes adecuadas, antes de reenviar la llamada a los servidores maestros de Kubernetes. Rancher se comunica con los clústeres de Kubernetes usando una cuenta de servicio.

La siguiente referencia bibliográfica contiene información extendida sobre el entorno Rancher [RD-14].

6.4 Anchore

Anchore Engine es un proyecto de código abierto que proporciona un servicio centralizado de inspección profunda, análisis y certificación de imágenes de contenedores. Se proporciona como una imagen de contenedor Docker que puede ejecutarse de forma independiente o en un clúster de Kubernetes.

- Permite análisis de vulnerabilidades.
- Análisis de *secrets* and passwords para verificar que no hay credenciales almacenadas dentro de la imagen.
- Análisis de los paquetes del sistema operativo.
- Análisis de librerías.
- Análisis de Docker File.
- Análisis pormenorizado de ficheros (permisos, pertenencia a usuarios y grupos)

6.4.1 Anchore como control de admisión de contenedores

Anchore puede integrarse con Kubernetes mediante un controlador de admisión que se comunica con el Anchore API Engine. Antes de ejecutar un contenedor para garantizar que la imagen del contenedor cumpla con las políticas de la organización y no contiene vulnerabilidades que puedan verse afectadas por una explotación de las mismas malintencionadas.

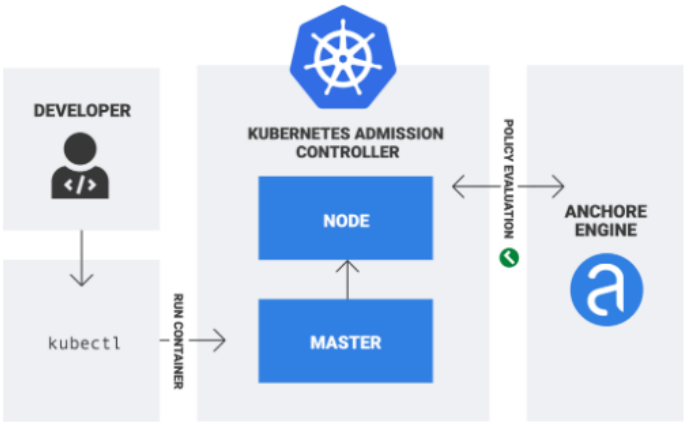


Ilustración 6: Anchore como controlador de admisión de contenedores

6.4.2 Descripción de los componentes de Anchore

Anchore Engine CLI:

Es la principal interfaz de línea de comandos que proporciona la suite de *Anchore* para poder gobernar la solución. Se encarga principalmente de interpretar y enviar los comandos pasados a Anchore Engine API.

Anchore Engine API:

La *API* de Anchore Engine es la *API* principal para todo el sistema. Este servicio permite orquestar toda la solución, utilizada también para analizar imágenes y obtener evaluaciones de políticas y gobernar la solución por completo.

Anchore Policy Engine:

El motor de políticas es responsable de cargar el resultado de un análisis de imágenes, normalizar, estructurar los datos de manera que sean rápidamente encontrados, escanear en busca de vulnerabilidades en los artefactos encontrados de la imagen y proporcionar una evaluación rápida de las políticas sobre esos datos.

Anchore Engine Analyzer:

El analizador Anchore Engine es el componente que realiza toda la descarga de imágenes y el análisis de heavy-lifting. Recibe el trabajo del servicio *Simplequeue* sondeando colas específicas y ejecuta el análisis de imágenes, cargando los resultados al catálogo y al motor de políticas una vez completado.

Anchore Engine Database:

Anchore se construye alrededor de una sola base de datos Postgresql. Contiene tablas para todos los servicios necesarios. Los servicios no se comunican a través del *DB*, sólo a través de llamadas explícitas a la API, pero las tablas de la base de datos están consolidadas para facilitar las operaciones de gestión.

6.4.3 Flujo de trabajo de Anchore

El siguiente diagrama muestra a modo introductorio el flujo simplificado de ejecución de Anchore, así como, una explicación del comportamiento esperado en cada una de las fases del flujo:

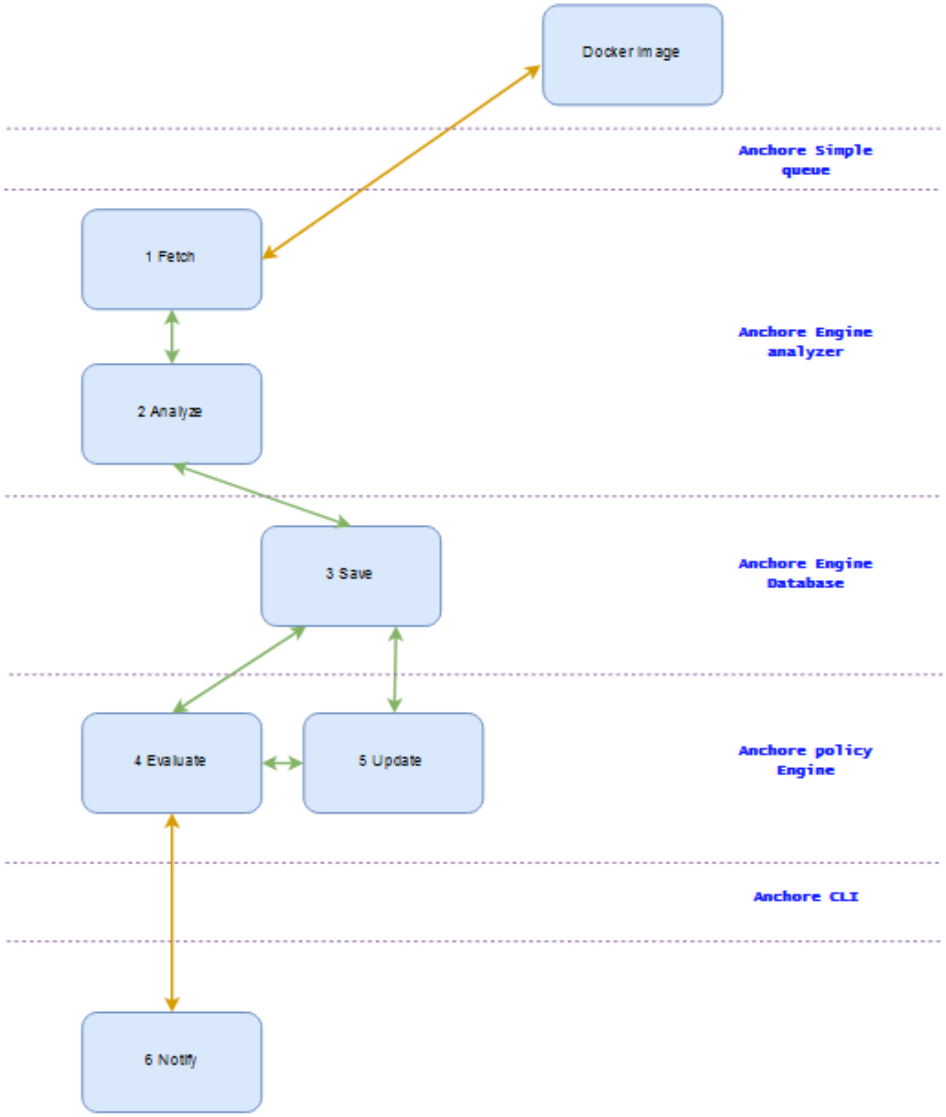


Ilustración 7: Flujo interacción módulos Anchore

- Fetch: Permite obtener el contenido de la imagen y extraerla, pero nunca ejecutarla.

- Analyze: Se encarga de analizar la imagen, ejecutando un conjunto de analizadores de Anchore sobre el contenido de la imagen para extraer y clasificar tantos metadatos como sea posible.

- Save: Permite guardar el análisis resultante en la base de datos para su uso y auditoría en el futuro.

- Evaluate: Módulo encargado de evaluar políticas contra el resultado del análisis, incluyendo coincidencias de vulnerabilidades en los artefactos descubiertos en la imagen.

- Update: Actualiza los últimos datos con las últimas vulnerabilidades dadas de alta utilizados para la evaluación de políticas y coincidencias de vulnerabilidades. Por otro lado, también es el encargado de actualizar automáticamente los resultados del análisis de imágenes.

- Notify: Este flujo se encarga de notificar a los usuarios de los cambios en las evaluaciones de políticas y coincidencias de vulnerabilidades en el análisis de imágenes. [RD-15]

7 Diseño y Arquitectura propuesta

A lo largo de este capítulo se expondrá dos diseños arquitectónicos a nivel de infraestructura basados en Kubernetes.

El primero de ellos pretende conceptualizar una propuesta de arquitectura de Kubernetes para un entorno empresarial en el que los distintos entornos de *test*, producción y *management* están separados entre sí para mejorar una respuesta ante posibles incidentes de seguridad.

El segundo de ellos consiste en una arquitectura de laboratorio, tomando como base la arquitectura de entornos corporativos y propiciando que se permita probar las distintas tecnologías objeto de estudio de este proyecto. Teniendo presente restricciones tecnológicas que conlleva la ejecución del mismo en una única *workstation* y dos máquinas virtuales.

7.1 Arquitectura entornos corporativos

Dicha arquitectura está basada en el entorno de ejecución de contenedores de Docker, orquestada con 3 clústeres de Kubernetes independientes para mantener la separación de entornos y orientada a una mejor respuesta ante incidentes de seguridad.

A continuación, se detallan cada uno de los tres entornos seguidos por un diagrama complementario.

Test_environment:

El primer clúster será el encargado de la ejecución de contenedores de *test* que estén en fase de desarrollo. Este clúster y como parte de un flujo de *DevSecOps*, permitirá que un desarrollador que quiera desplegar una nueva imagen de Docker a través de Jenkins, pueda ejecutar un análisis con Anchore. Esto permitirá comprobar si su contenedor contiene vulnerabilidades críticas y que por tanto deben de ser solucionadas. Ya que no serán ejecutadas en el clúster posteriormente debido al control de admisión de contenedores de que dispone el clúster.

Sec_environment:

El segundo clúster será el encargado de la orquestación de todos los contenedores dedicados a la ejecución de las herramientas de seguridad que se encargaran del análisis del resto de los entornos de contenedores. Entre las ventajas que podemos encontrar al tener un clúster dedicado a herramientas de seguridad se encuentran las siguientes:

- No ejecutará contenedores de producción que hayan podido ser desarrollados por terceras empresas y que no estén alineadas con el desarrollo seguro.

- Sus recursos estarán plenamente dedicados al análisis de vulnerabilidades en imágenes que están corriendo en otros clústeres.

- Los cambios y modificaciones necesarios para la publicación de servicios que suelen afectar a clúster en los que se encuentran contenedores de servicios (producción/desarrollo), no afectan a este clúster, que únicamente publicará los pocos accesos y *API* que sean necesarios para su funcionamiento.

- Puede ser usado como security Toolbox, que permita contener, no únicamente Anchore, podra contener también soluciones tales como: wazuh, *IDS* o Falco para análisis de logs y comportamiento, hasicorp vault, notary para firmado de imágenes, o snyk para análisis de vulnerabilidades.

Prod_environment:

Como se ha expuesto anteriormente esta propuesta busca la separación de los entornos en distintos clústeres, para mantener aislados entre si el entorno de producción, del entorno de desarrollo, así como, del entorno donde se ejecutan todas las herramientas de seguridad. Esto supone una mejora sustancial en el nivel de seguridad.

Como ejemplo de lo anteriormente indicado, el clúster de *test* por su naturaleza de uso podría ser más propenso a sufrir una vulnerabilidad explotable que pudiese llegar a comprometer el clúster por completo. Dicho clúster al estar separado, el incidente quedaría aislado al entorno de *test*, sin comprometer el clúster de seguridad, ni el de producción.

Como partida final, el tercer clúster se encargará de la ejecución de todos los contenedores que se encuentren en producción. A continuación, se muestra un esquema en el que aparecen representados los tres clústeres con cada una de sus componentes principales.

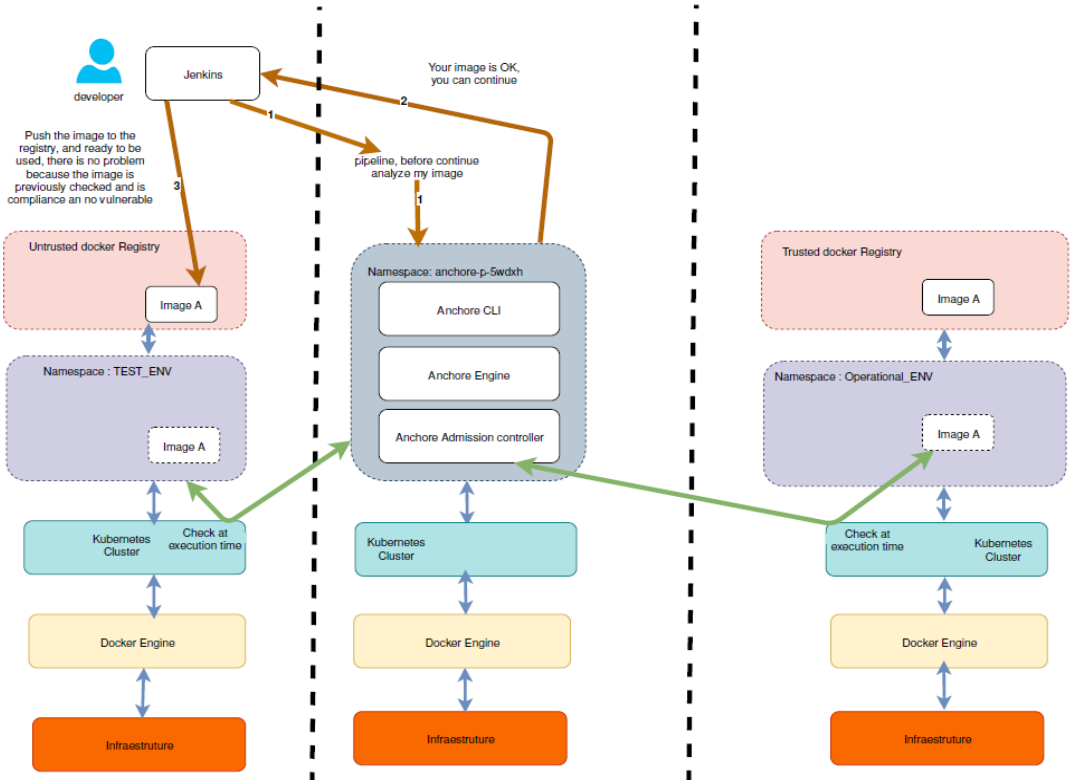


Ilustración 8: Arquitectura entornos corporativos

En el esquema anterior se puede apreciar en el flujo de flechas de color marron como un desarrollador hace un *push* de una imagen de Docker. Previo a ello, la imagen es analizada por la solución Anchore y en caso de ser una imagen sin vulnerabilidades, puede ser subida al clúster de *test* para su ejecución.

Indicado por las flechas verdes se puede apreciar que cada vez que el clúster procede a ejecutar una imagen, previamente pide comprobación de la misma al *pod* de control de admisión de Anchore.

7.2 Arquitectura de laboratorio

Para analizar el funcionamiento de Anchore como analizador de vulnerabilidades en imágenes durante el desarrollo de este proyecto se ha creado un entorno reducido de laboratorio sobre máquinas virtuales Vmware, que permita analizar las funcionalidades ofrecidas por Anchore en cuanto a análisis de imágenes.

El siguiente esquema muestra como mediante una simplificación a *namespaces* de Kubernetes se ha simulado cada uno de los tres clústeres en un único clúster, teniendo siempre como objetivo ser un entorno simulado de la arquitectura corporativa propuesta en la sección anterior.

El entorno de laboratorio será ejecutado en dos máquinas virtuales con CentOS instalado, Kubemaster *server* hará de master dentro de clúster y kubenode será un servidor que tendrá el rol de *worker* .

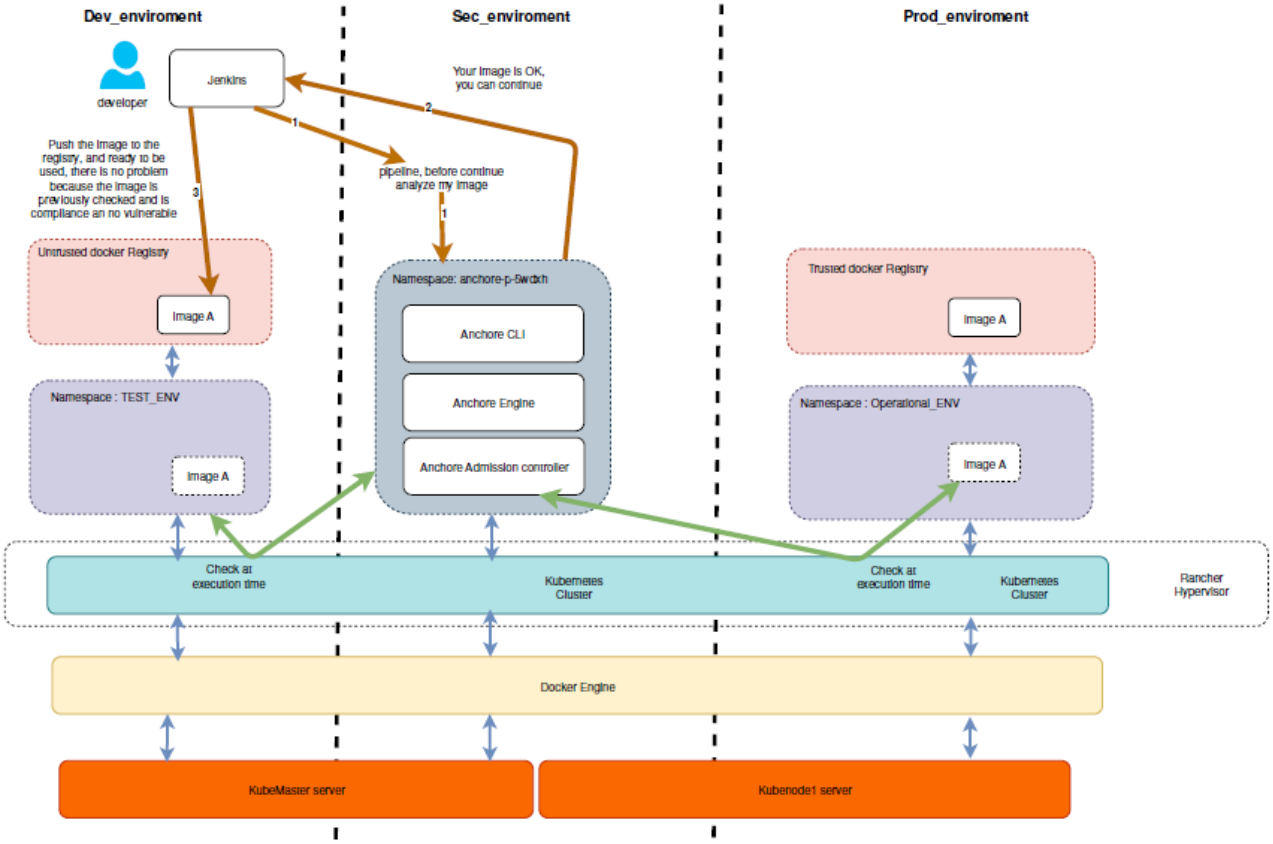


Ilustración 9: Arquitectura Laboratorio

8 Configuración de Anchore.

Con la pretensión de mantener el foco sobre el propio análisis de vulnerabilidades y los flujos descritos en el apartado anterior sobre las arquitecturas descritas, se ha procedido a condensar en esta sección las partes imprescindibles y necesarias de la configuración de Anchore que permitan al lector mantener la idea del funcionamiento de Anchore. Dejando el resto de los apartados fundamentales de la configuración e instalación de Anchore como anexo, aparatado 13.4.1.

8.1 Instalación de Anchore-cli

Con el fin de administrar Anchore, la solución presenta *Anchore* API Engine. Este *pod* es el encargado de toda la gestión de comunicaciones a través de *API* entre los distintos *Pod* de Anchore. Publicando también al exterior dicha *API* para poder gestionar la solución de Anchore.

Anchore *CLI* es un cliente ligero que permite la administración de la solución Anchore haciendo uso de la *API* anteriormente descrita, en el entorno de laboratorio será instalado en el propio servidor *Kubemaster*.

Activación del repositorio *EPEL*:

```
# yum install epel-release
```

Instalación de *phyton package installer*:

```
 # yum install python-pip
```

Instalación Anchore *CLI*:

```
 # pip install anchorecli
```

Una vez instalado mediante la siguiente sintaxis de comando se podrán pasar comandos al contenedor de Anchore encargado de la *API,* Anchore API Engine.

```
# anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image list
```

Anchore-cli --u (usuario) –p (password) –url (url publicada por el contenedor Anchore API Engine) seguida del comando a ejecutar, en la línea de comando descrita anteriormente Anchore mostraría un listado con las imágenes de Docker ingestadas y analizadas.

8.2 Registros de contenedores dados de alta en Anchore

Para que la solución Anchore pueda realizar el análisis de las imágenes de Docker, requiere de acceso directo a los mismos registros de Docker que tenga configurados el propio clúster de Kubernetes. Para ello se procederá a dar de alta con la siguiente secuencia de comandos:

Por defecto Anchore no muestra ningún registro dado de alta:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 registry list
```

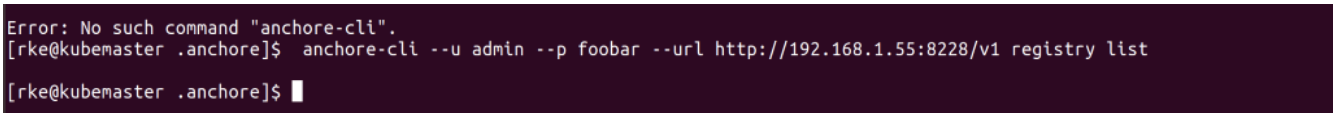


Ilustración 10: Registries dados de alta en Anchore

Para continuar con el propósito de proyecto se dará de alta un registro de Docker Hub, (anexo 13.3), creado exprofeso para el proyecto. Dicho registro será agregado a Anchore para pueda acceder a las imágenes con el fin de analizarlas. En el entorno de laboratorio creado para el proyecto, este registro será una simulación del registro de *test* donde los desarrolladores subirán sus imágenes para testearlas.

Alta de un nuevo registro en Anchore:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1  registry add
docker.io/dockerhubjuliorepository2019/ancho retest dockerhubjuliorepository2019
lamota2143-3
```

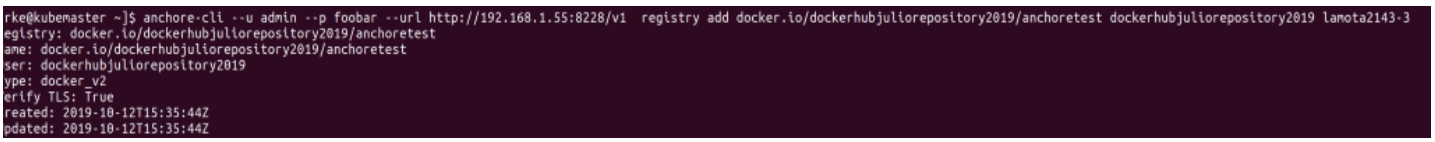


Ilustración 11: Alta de un registro en Anchore

Como comprobación, se ejecutará de nuevo el comando para listar los registros. Chequeando así, que ahora si muestra este último registro dado de alta. Anchore ahora será capaz de analizar las diferentes vulnerabilidades de las imágenes que contenga dicho registro.

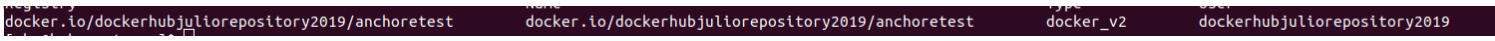


Ilustración 12: Registros dados de alta en Anchore

En caso de querer dar de baja un registro utilizaremos el siguiente comando:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1  registry del <name>
```

El siguiente documento de aplicación permite obtener información detallada sobre la configuración y uso de registros en Anchore [AD-01]

8.3 Políticas de análisis en Anchore

Las políticas pueden ser descritas dentro del ámbito de Anchore, como un conjunto de reglas y checkeos que permiten a Anchore obtener una valoración de una imagen de Docker, con respecto al cumplimiento o no cumplimento de la imagen con respecto a la política.

Cada una de las reglas que contiene una política puede tener como resultado dos estados *STOP* y *WARN*. El primero de ellos indicará que, ante la evaluación y resultado positivo de una regla de STOP, el resultado debería de ser bloqueante. Por el contrario, *WARN* indica que ante la evaluación positiva de una regla de WARN el resultado debe de ser un aviso.

A modo de ejemplo y para aclarar el concepto descrito anteriormente, la siguiente regla indica que en caso de que al analizar una imagen de Docker exista el usuario *Root* considerado según nuestra política como un usuario crítico, la política devolverá un indicador o *flag* de *stop* en dicha evaluación.

Política en formato *JSON*:

```
"action": "STOP",
                    "comment": "section 4.1",
                    "gate": "dockerfile",
                    "id": "c96bf84d-0e76-435c-a94c-0f556bbaf45f",
                    "params": [
                        {
                            "name": "users",
                            "value": "root,docker,vulnerableuser"
                        },
                        {
                            "name": "type",
                            "value": "blacklist"
                        }
                    ],
                    "trigger": "effective_user"
```

Como ejemplo de la evaluación anterior, Jenkins mostrará el siguiente aviso en el reporte:

d0957ffdf8a2ea8c8925903862b65a1b6850dbb019f88d45e927d3d5a3fa0c31	docker.io/dockerhubjuliorepository2019/anchoretest:centos6	68e630cef4a8533b139875aa5fc54da5	dockerfile	effective_user	User root found as effective user, which is explicity not allowed list	STOP	false	cb417967-266b-4453-bfb6-9acf67b0bee5

Ilustración 13: Ejemplo de evaluación de Jenkins

8.3.1 Uso básico de políticas en Anchore

En esta sección, serán descritos los comandos básicos de gestión de políticas en Anchore, con la pretensión de que posteriormente sean utilizados para la modificación de una política.

El siguiente comando permite listar las políticas que Anchore tiene descargadas y cuáles de ellas pueden aplicarse. Cabe destacar que únicamente una de ellas puede estar activa al mismo tiempo a nivel de *engine*.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1  policy list:
```

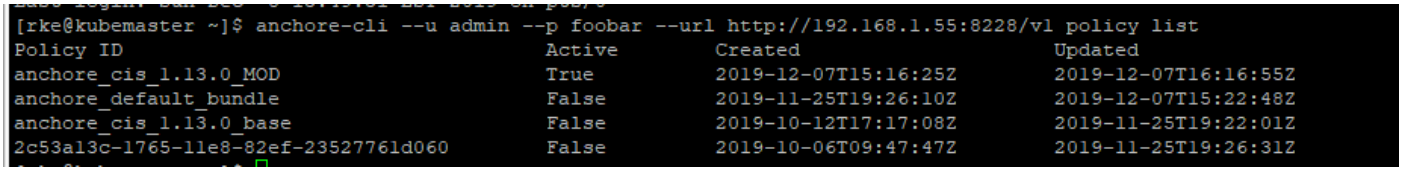


Ilustración 14: Políticas aplicables en Anchore

La política '*2c53a13c-1765-11e8-82ef-23527761d060*' corresponde con la política por defecto que usa Anchore, podemos aplicar otras políticas haciendo uso de las publicadas dentro de Anchore Hub, una de ellas y de especial interés es la basada en las recomendaciones del *CIS.*

En caso de querer obtener más información de una política, el siguiente comando permite obtener información detallada:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 policy get 2c53a13c-1765-11e8-82ef-23527761d060
```

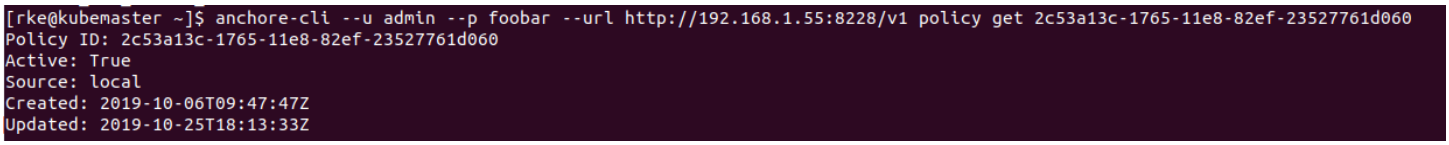


Ilustración 15: Información de una política detalla en Anchore

Con la pretensión de mantener un orden dentro de una política, Anchore separa por categorías o *gates* cada uno de los apartados en los que puede realizar análisis. Con el siguiente comando se pueden mostrar las distintas categorizaciones permitidas dentro de una política.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 policy describe
```

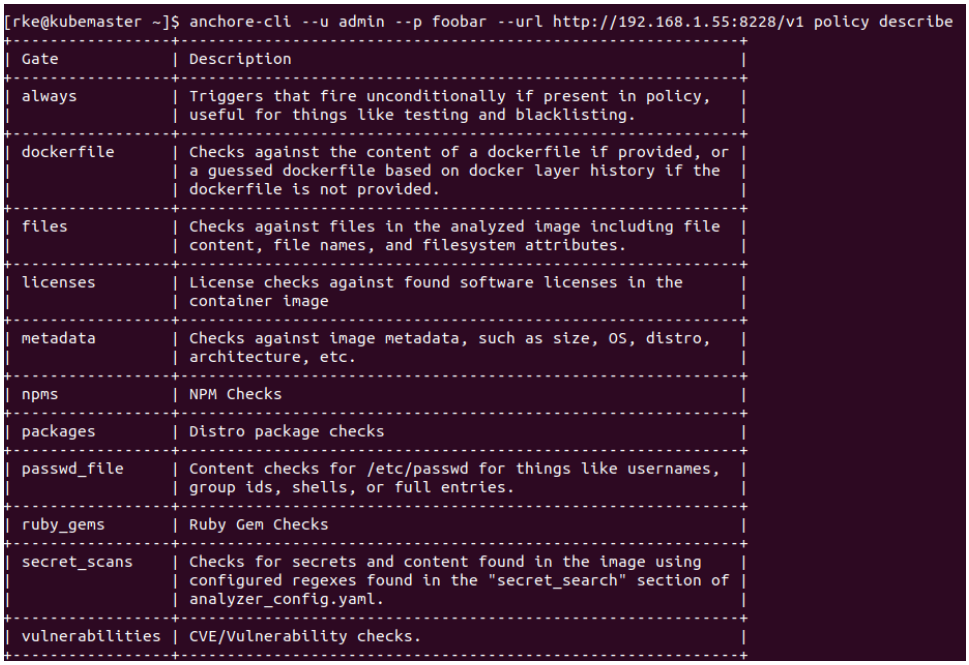


Ilustración 16: Categorías de análisis cubiertos por una política

Dentro de cada *gate* de política existen distintos *triggers,* siendo estos las propias evaluaciones que capturaran el resultado del análisis de una regla. Siguiendo el ejemplo anterior, la regla de chequeo de usuario *Root* pertenece al *gate* “dockerfile” *trigger* “effective_user”:

```
"action": "STOP",
                    "comment": "section 4.1",
                    "gate": "dockerfile",
                    "id": "c96bf84d-0e76-435c-a94c-0f556bbaf45f",
                    "params": [
                        {
                            "name": "users",
                            "value": "root,docker,vulnerableuser"
                        },
                        {
                            "name": "type",
                            "value": "blacklist"
                        }
```

El siguiente comando permite mostrar los *triggers* que existen dentro de cada *gate*. En concreto este comando muestra los *triggers* existentes dentro del *gate* de vulnerabilidades.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 policy describe --gate=vulnerabilities
```

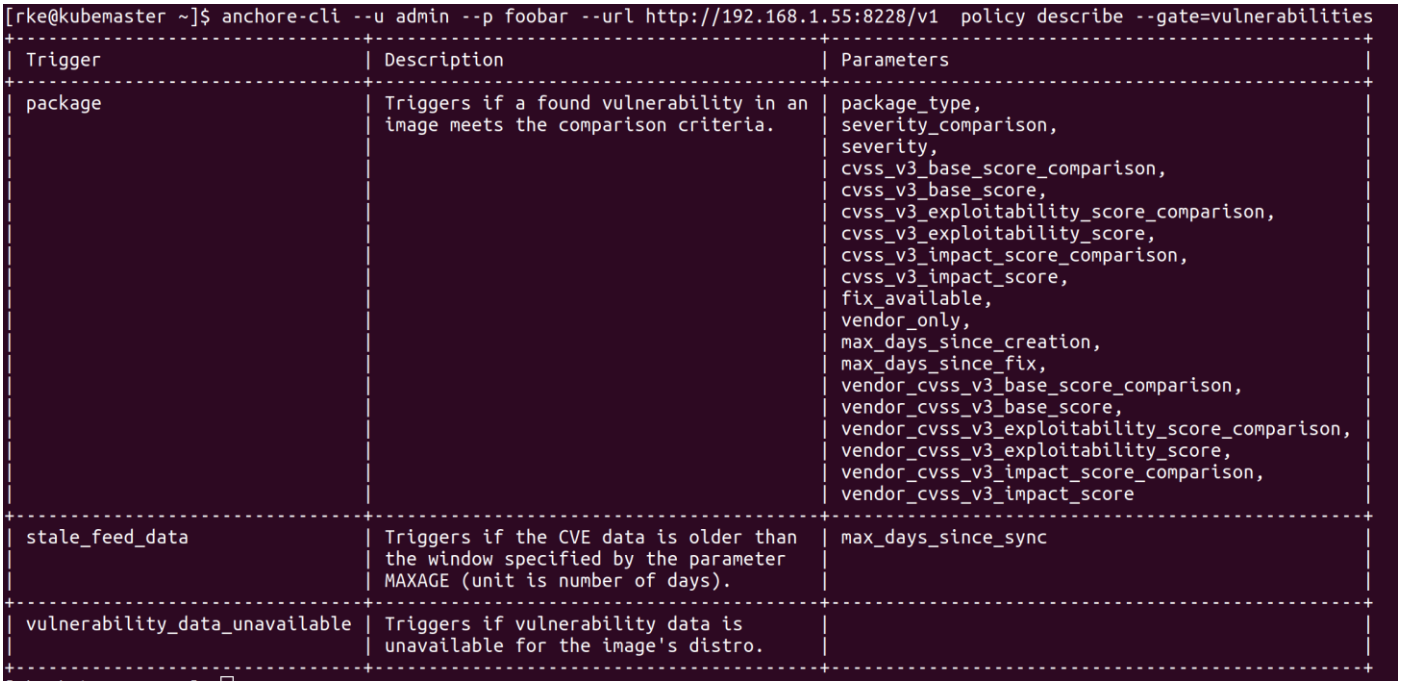


Ilustración 17: Triggers permitidos dentro de la categoría vulnerabilidades

8.3.2 Instalación de políticas en Anchore

Anchore dispone de un archivo o *hub* en el que son publicadas distintas políticas que pueden ser de interés. Un ejemplo de ello, es la basada en las recomendaciones del *CIS*. A lo largo de esta sección se abordará como descargar políticas desde Anchore Hub y como proceder a su modificación.

El siguiente comando lista las políticas publicadas en la comunidad de Anchore Hub. Como se puede comprobar en la captura inferior, actualmente existen cuatro políticas disponibles.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   policy hub list
```

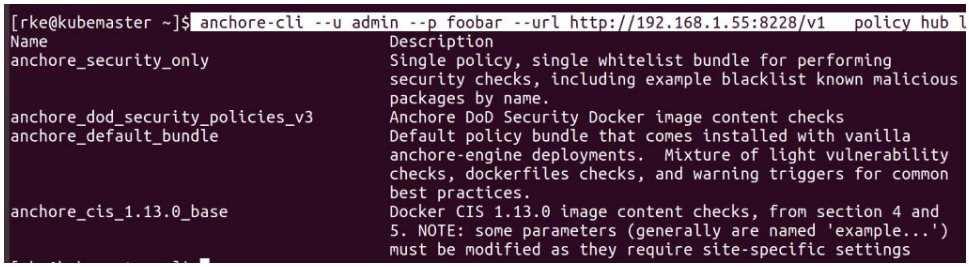


Ilustración 18: Lista políticas disponibles en Anchore

Para descargar una política además de obtener información detallada sobre ella, utilizaremos el siguiente comando:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   policy hub get
anchore_cis_1.13.0_base
```

Ilustración 19: Información detallada de una política de Anchore

Una vez la política ha sido descargada para proceder con su instalación, ejecutaremos el siguiente comando. En este caso se procederá a instalar la política basada en las recomendaciones del *CIS*.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   policy hub install
anchore_cis_1.13.0_base
```

Para comprobar si se descargó correctamente podemos ejecutar el comando *policy list*.

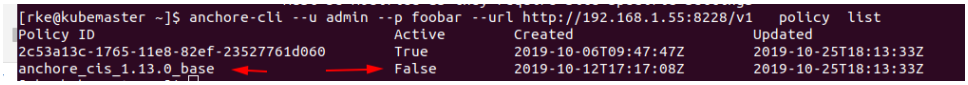


Ilustración 20: Instalación de una política de Anchore

En caso de que fuese requerido la activación de una política en concreto y que pase a ser la política por defecto que aplicaría Anchore, ejecutaremos el siguiente comando.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   policy  activate
anchore_cis_1.13.0_base
```

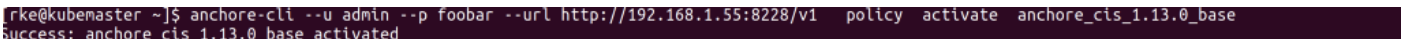


Ilustración 21: Activación de una política de Anchore

Ejecutando de nuevo *policy list* se podrá comprobar que efectivamente ha quedado activada:

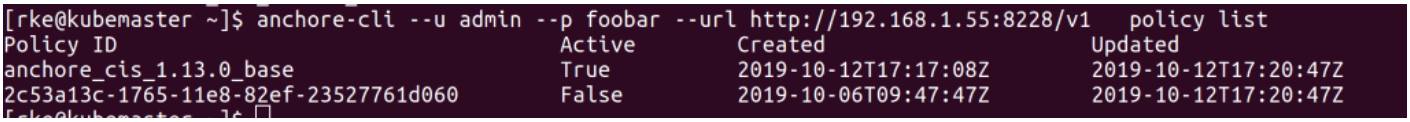


Ilustración 22: Comprobación de activación de una política de Anchore

8.3.3 Modificación de políticas en Anchore

En esta sección será abordado el procedimiento seguido para modificar una política y su posterior instalación. Debido a su interés desde el punto de vista de seguridad se ha procedido a tomar como base la política del *CIS* "Anchore_CIS_1.13.0_base" que tras las modificaciones introducidas, según el interés del proyecto dará lugar a "Anchore_CIS_1.13.0_MOD".

A través del siguiente comando se pueden sacar todas las reglas que implementan las políticas de *CIS* el formato obtenido es formato *JSON*.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1  policy get
anchore_cis_1.13.0_base –detail
```

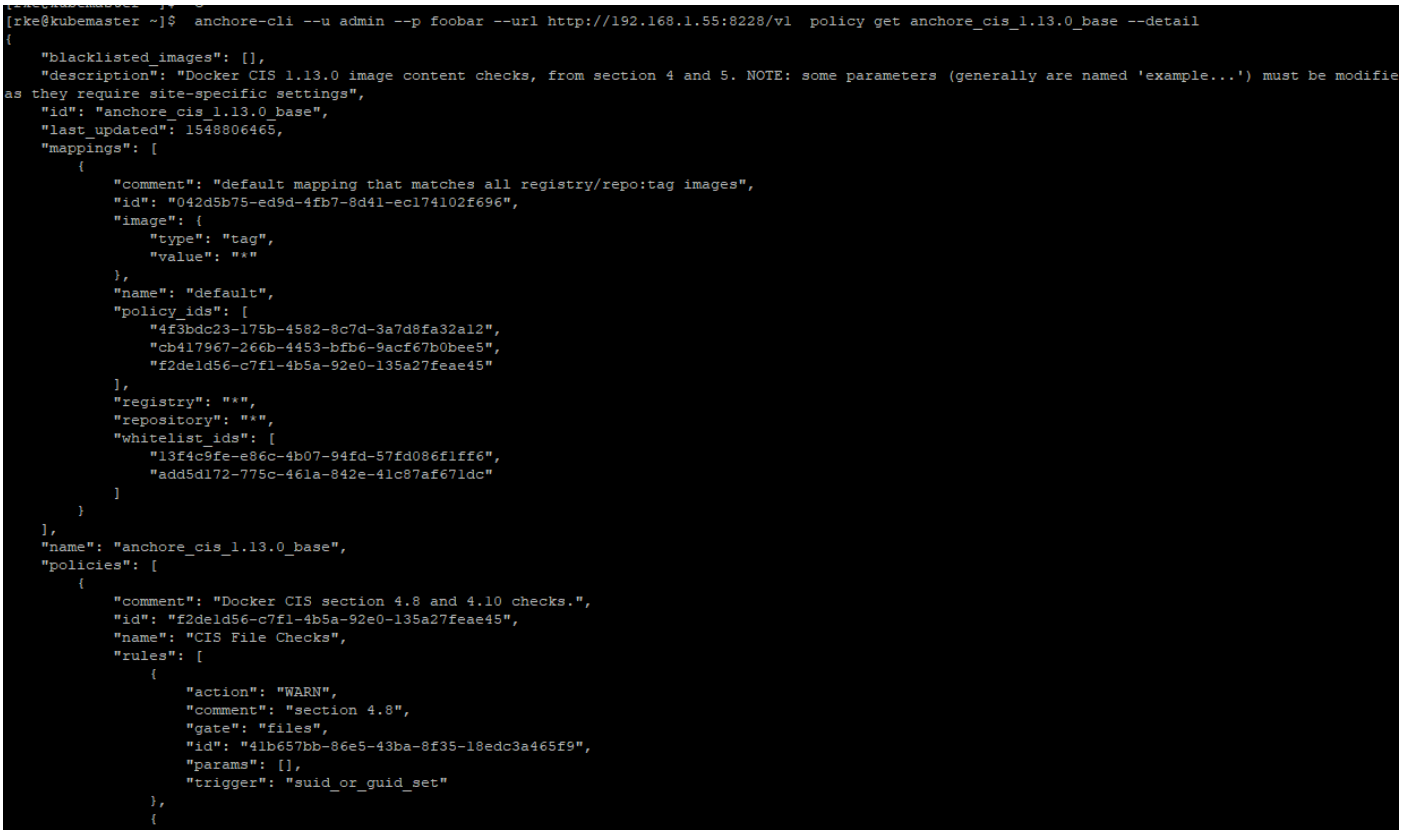


Ilustración 23: Detalle de reglas de una política de Anchore

Para facilitar el paso de la política a un fichero que podamos modificar ejecutaremos el comando anterior, pero con una redirección a fichero.

```
Anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1  policy get
anchore_cis_1.13.0_base --detail >>Anchore_CIS_1.13.0_MOD.json
```

A continuación, se hace referencia a las modificaciones en la política que se han introducido a modo de testeo de la misma. Por su propio origen, la política del *CIS* resulta demasiado restrictiva por lo que se ha procedió a reducir de *STOP* a *WARN*. Cabe destacar que se ha modificado a *WARN* las referentes a vulnerabilidades de paquetes de criticidad media y *low*.

Se ha incluido la inspección de puerto 22 no permitido en los contenedores, así como, el chequeo de existencia del usuario *Root*.

En la sección documentos de aplicación entrada [AD-02] puede encontrar el fichero .*JSON* con la política aplicada en el proyecto.

Una vez finalizado con el proceso de modificación de la política en función de las necesidades y con la ejecución del siguiente comando se procederá a la instalación de la política.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 policy add
Anchore_CIS_1.13.0_MOD.json
```

el siguiente comando permitirá comprobar si la política fue correctamente instalada en Anchore:

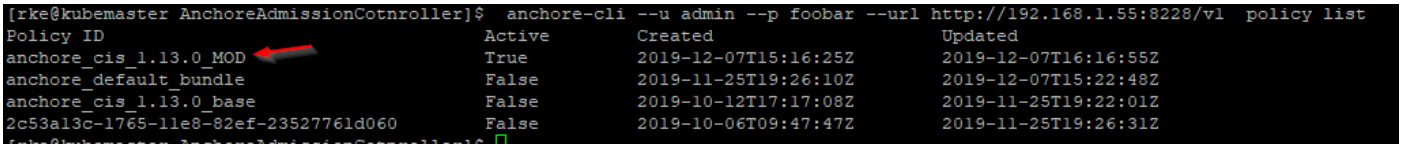


Ilustración 24: Aplicación de una política de Anchore

El último paso será la activación de dicha política haciendo uso del comando *policy activate*:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 Policy Activate
anchore_cis_1.13.0_MOD
```

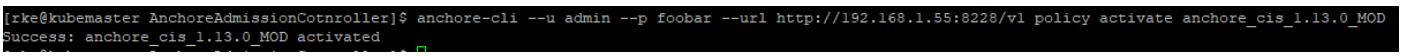


Ilustración 25: Activación de una política en Anchore

El siguiente documento de aplicación amplia la información contenida en este capítulo con respecto al uso de políticas dentro de Anchore [AD-03].

8.4 Anchore Admission Controller

Dentro de la composición de los distintos *pods* de la solución base de Anchore no se incluye el *pod* de control de admisión para clústeres de Kubernetes, para poder hacer uso de esta funcionalidad, se deberá desplegar dicho *pod* de forma manual con el siguiente procedimiento.

Se procederá a crear el fichero de secretos con las credenciales de la *API*:

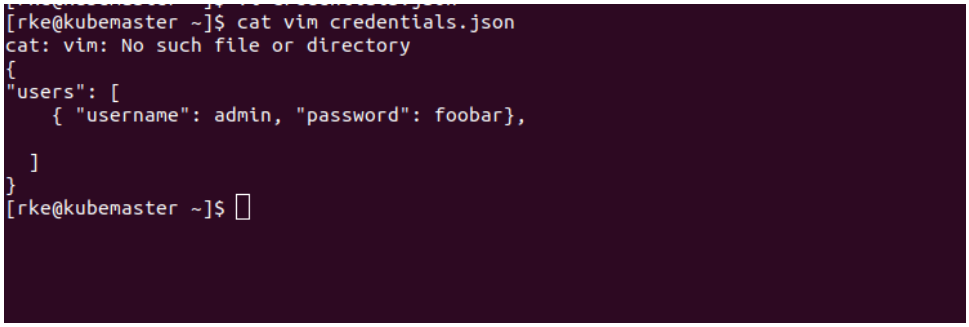


Ilustración 26: Secrets Anchore Admission Controller

Se deberá importar el fichero de secretos en el *namespace* en el que se pretenda desplegar el *pod* de Anchore Admission Controller.

Listado los *namespaces* activos en el clúster:

```
[rke@kubemaster ~]$ kubectl get namespaces --show-labels
NAME              STATUS   AGE     LABELS
anchore-p-5wdxh   Active   15d     cattle.io/creator=norman,field.cattle.io/projectId=p-5wdxh,mcapp=anchore
cattle-system     Active   22d     field.cattle.io/projectId=p-9zlmp
centos6           Active   4d23h   cattle.io/creator=norman,field.cattle.io/projectId=p-5wdxh
default           Active   22d     field.cattle.io/projectId=p-5wdxh
ingress-nginx     Active   22d     field.cattle.io/projectId=p-9zlmp
jenkins           Active   15d     cattle.io/creator=norman,field.cattle.io/projectId=p-5wdxh
kube-node-lease   Active   22d     field.cattle.io/projectId=p-9zlmp
kube-public       Active   22d     field.cattle.io/projectId=p-9zlmp
kube-system       Active   22d     field.cattle.io/projectId=p-9zlmp
```

Ilustración 27: Namespaces activos clúster Kubernetes

Importación de las variables de secreto en el *namespaces* donde tenemos desplegados el resto de los contenedores de Anchore:

```
kubectl create secret generic anchore-credentials --from-file=credentials.json --namespace=anchore-p-5wdxh
```

```
rke@kubemaster AnchoreAdmissionCotnroller]$ kubectl create secret generic anchore-credentials --from-file=credentials.json --namespace=anchore-p-5wdxh
ecret/anchore-credentials created
```

Ilustración 28: Importación de Secrets al clúster de Kubernetes

Comprobación de que el secreto fue generado correctamente:

```
[rke@kubemaster AnchoreAdmissionCotnroller]$ kubectl get secrets --namespace=anchore-p-5wdxh
NAME                           TYPE                                  DATA   AGE
anchore-credentials            Opaque                                1      5m29s
anchore-p-5wdxh-anchore-engine Opaque                                2      15d
anchore-p-5wdxh-postgresql     Opaque                                1      15d
default-token-kjsws            kubernetes.io/service-account-token   3      15d
dockerhubanchorprivateregistry kubernetes.io/dockerconfigjson        1      15d
```

Ilustración 29: Comprobación de alta de Secrets en un namespace

Tras los pasos anteriores y mediante el *script* indicado por los desarrolladores de Anchore Engine *Controller* se generará un fichero. yaml que desplegará el *pod* de Anchore Admission Controller. Previo despliegue del *pod* y a modo de reseña entre los parámetros más importantes a tener en cuenta en este fichero de configuración están:

- Ip del servicio de Anchore API Engine.
- Política de Anchore que Admission Controller aplicará sobre el clúster.
- Modo de actuación de Admission Controller. Entre las diferentes opciones permitidas están:

  - Modo *policy*: Admission Controller requerirá que el *pod* pase con resultado favorable el análisis de Anchore.

  - Modo análisis: Admission Controller requerirá que el *pod* forme parte de la base de datos de Anchores y este analizado.

  - Modo Breakglass: Admission Controller actuará en modo trasparente y no tomará acciones sobre los *pod*s

```
[rke@kubemaster AnchoreAdmissionCotnroller]$ cat values.yaml
credentialsSecret: anchore-credentials
anchoreEndpoint:        "http://192.168.1.55:8228"
#enabled: true
#requestAnalysis: true

policySelectors:
  - Selector:
      ResourceType: "pod"
      SelectorKeyRegex: ".*"
      SelectorValueRegex: ".*"
    PolicyReference:
      Username: "admin"
      # This is the default bundle id in anchore engine
     # PolicyBundleId: "2c53a13c-1765-11e8-82ef-23527761d060"
   #PolicyBundleId: anchore_default_bundle
    PolicyBundleId: "anchore_cis_1.13.0_MOD"
    #PolicyBundleId: "anchore_cis_1.13.0_base"
    # Mode is one of: "policy", "analysis", or "breakglass". policy=>require policy pass, analysis=>require image analyzed, breakglass=>do nothing
    Mode: policy
```

Ilustración 30: Fichero configuración de despliegue del pod Admission Controller

Una vez generado el fichero values.yaml, se procederá a desplegar el contenedor en el clúster con el siguiente comando:

```
helm install --name anchoreacon --namespace anchore-p-5wdxh anchore/anchore-admission-controller -f values.yaml
```

Como resultado dentro de Rancher se comprobará que el *pod* Anchore Admission Controller está corriendo dentro del *namespace* donde se encuentran el resto de contenedores de la solución:

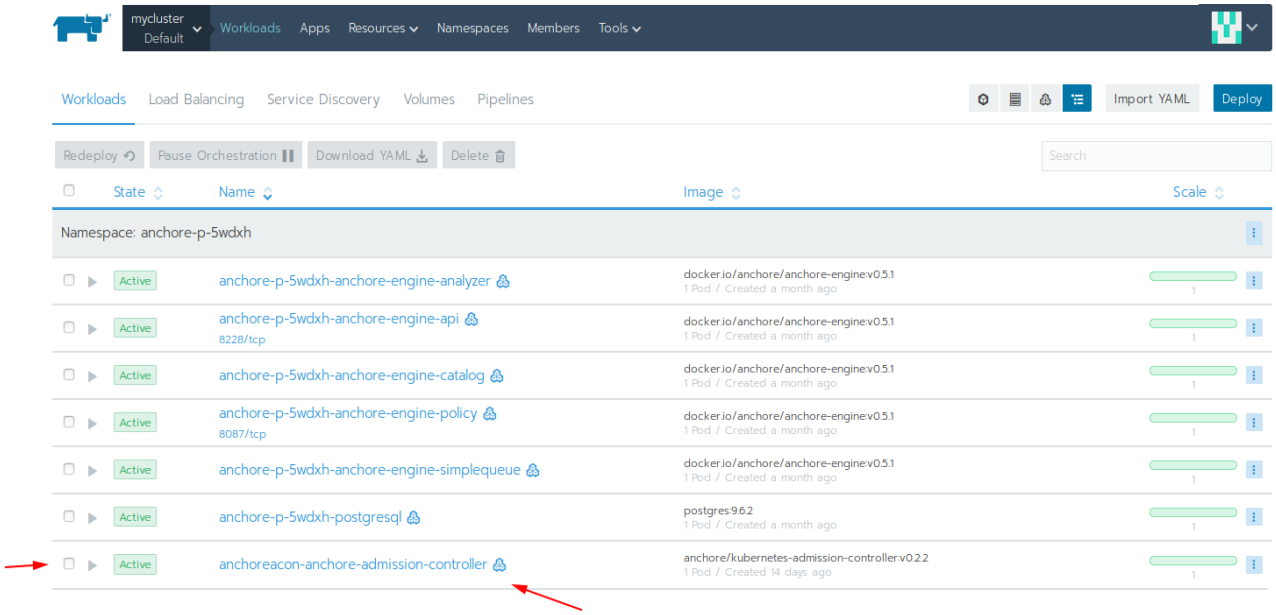


Ilustración 31: Comprobación de Anchore Admission Controller desplegado

8.4.1 Anchore Admission Controller webhook

En la sección anterior fue descrita la instalación del *pod* Anchore Admission Controller encargado de la gestión del control de admisión de los *pod*s, que pueden ser ejecutados en el clúster. El último punto de configuración necesario, es facilitar que el clúster de Kubernetes sea consciente de Anchore Admission Controller y se comunique con la previa ejecución de cualquier *pod*, Esto puede ser conseguido mediante el uso de webhooks de Kubernetes, en concreto, mediante el uso de Validating Admission Webhooks.

El equipo de desarrollo de Anchore Admission Controller proporciona un script que permite la generación del webhook necesario en formato. yaml para ser configurado sobre el propio clúster.

Como requisito previo a la ejecución del *script* es necesario poseer el kubeconfig que contiene los certificados que permiten la comunicación con la *API* de Kubernetes.

Tras su ejecución se obtiene un fichero ".yaml" con el webhook resultante, cabe destacar que es recomendable quitar el carácter # que hace de comentario sobre las líneas indicadas. Esto facilitara al administrador del clúster que los *pod* contenidos en ciertos *namespaces* sean escaneados por Anchore siempre que hagan uso de la etiqueta *exclude.admission.anchore.io.*

Es recomendable el uso de esta etiqueta en el namspace que contiene los *pods* de la propia solución de Anchore. Así como, *pod* que por su especial interés no requieran de ser evaluados, especialmente si el modo de comportamiento de Anchore Admission Controller es configurado en modo *policy*.

```
apiVersion: admissionregistration.k8s.io/v1beta1
kind: ValidatingWebhookConfiguration
metadata:
  name: anchoreacon-anchore-admission-controller.admission.anchore.io
webhooks:
- name: anchoreacon-anchore-admission-controller.admission.anchore.io
  clientConfig:
    service:
      namespace: default
      name: kubernetes
      path: /apis/admission.anchore.io/v1beta1/imagechecks
    caBundle: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN3akNDQWFxZ0F3SUJBZ01CQURBTkJna3Foa2lHOXcwQkFRc0ZBREFTTVJBd0RnWURWUVFERXdkcmRXSmwKTFdOaE1lCNFhEVEU1TURreU9URXdOVFF6T0ZvWERUSTVNRGt5TmpFd05UUXpPRm93RWpFUU1BNEdBMVVFQXhNSAphM1ZpWlMxallUQ0NBU0l3RFFZSktvWklodmNOQVFFQkJRQURnZ0VQQURDQ0FRb0NnZ0VCQU1SbS9TTTVZWXpwCjhnbm9na0ZlbGRRQkdIV2tlMDl2YzBCcGE0Wk9PZjhkWGU5ZExhTnhQaWltaXpvRHJpYXBIZ2hteXlFQTljOEsKV1BpYXBsbVlxalpZSHF2VG9Ma3ptNjN5NFRPM1J1ZEtvTUdWL0NFRTdKeWlzclZqYTd5bVQ2aDNKREF6Y0FQRgozQ2tjRzVJQnZISjJWVTVrZFJuVWQ2d2d6RGZHVXlBRkk5cHhkeFVoczRvUVNVU3hscXRLUnhDTlQzbk5iUTRwCnpZTjdPcUVIZlVXSHhGNWRJRGZzWW14SUtpV3lZTnFMR2E1L3p4L0
50UT1FUE01S0F5aGdlc0FZYmklUkFoT2YKdnVIZHBjTVRPM3liWHNEMHlQRmJzUFhwT1BUUURGN1ZWTkVvMlBCU0xpV1FwQ2RsanpVTGtvSzl3UjdNWDJFNQpZUTc0ZTQvc0pJc0NBd0VBQWFNakl1DRXdEZl
lEVlIwUEFRSC9CQVFEQWdLa01BOEdBMVVkRXdFQi93UUZNQU1CCkFmOHdEUV1KS29aSWh2Y05BUUVMQ1FBRGdnRUJBSTZKT2dod3ZnbXNPVHZPd01rWGFNT2xGRmVOQTNtSXIlVWoKVUtVRGZYeXRrYmxsMy
9ram5WLytRN0RCR2YrMVZGanBhaGhLTnBCNXhtZWZMZlREYzBGWEpWY2xlcng0L05Wdwp3Q3NFQktESXgvNnRra2pPcGdCSloxKytiTG9hSEN0bj10TEt3ODE3QjR4UEgySCtsazlHd3ZjMW9tU2NGdklHCk
5MbkJGUHFMZVozaFhpMVA2SHJRRFJSTFR0Sm9XUEJqbGJHVkJ6eEJ3OERqTlpod0xaWTR0R0FCTE9ERGNhdXoKYUhMQVFsS0ZmQU8rK0pCcDRUaVBZZldSSFZHR1UvODJwQjduNDQlaTUvQ3I2ek5HUGhDWF
hpNld2enc2QWZiMgpINVM5eWU0Uk9UcmEreUJsTzljRlpJR3JVaFVjazVoT0VFT0Z5blVRRzNEN1RoSzdrT3M9Ci0tLS0tRU5EIENFU1RJRklDQVRFLS0tLS0K
  rules:
  - operations:
    - CREATE
    apiGroups:
    - ""
    apiVersions:
    - "*"
    resources:
    - pods
  failurePolicy: Fail
# Uncomment this and customize to exclude specific namespaces from the validation requirement
  namespaceSelector:
    matchExpressions:
      - key: exclude.admission.anchore.io
        operator: NotIn
        values: ["true"]
[rke@kubemaster AnchoreAdmissionCotnroller]$
```

Ilustración 32: Creación .yaml configuración webhook

A través del siguiente comando, se procederá a configurar el webhook de admisión.

```
kubectl apply -f validating-webhook.yaml
```

Los siguientes documentos de aplicación permiten obtener información detalla del funcionamiento interno de Anchore Admission Controller así como una descripción pormenorizada de todas las configuraciones posibles [AD-04], [AD-05].

8.5 Comprobación de las imágenes que Anchore tiene ingestadas

A través de la ejecución del siguiente comando, se pueden comprobar todas las imágenes que Anchore tiene ingestadas y el estado en el que se encuentran. En este caso todas las existentes están ya analizadas.

```
# anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image list
```

```
[rke@kubemaster ~]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image list
Full Tag                                                    Image Digest                                                            Analysis Status
docker.io/centos:7                                          sha256:a36b9e68613d07eec4ef553da84d0012a5ca5ae4a830cf825bb68b929475c869 analyzed
docker.io/dockerhubjuliorepository2019/anchoretest:centos6  sha256:9aae95c8043f4e401178d68006756dc68982ae6d0693b71a714754227ce0abc6 analyzed
docker.io/dockerhubjuliorepository2019/anchoretest:ubuntumysql sha256:85e63cd729c4eedf2b52db72f624cc82cf9bad9932cb4ac55d24d67125f813b8 analyzed
docker.io/hello-world:latest                                sha256:92c7f9c92844bbbb5d0a101b22f7c2a7949e40f8ea90c8b3bc396879d95e899a analyzed
docker.io/httpd:2                                           sha256:ca908f415a15fdba408f82537d295350772afa985112ee62db6709fea994a682 analyzed
docker.io/httpd:2.4                                         sha256:64d607e1a0d145d90e4bf811491c3b51c04a55d393e307c0ab0fb8e26c8b098c analyzed
docker.io/mysql:latest                                      sha256:0491ecfc011cdebbd6c9bc2f5cd8fd0bd43f6e9b96caae96fb404eb00381068d analyzed
docker.io/nginx:latest                                      sha256:f56b43e9913cef097f246d65119df4eda1d61670f7f2ab720831a01f66f6ff9c analyzed
docker.io/tomcat:7.0                                        sha256:88dab61a89a836ce47ee937c0f02fcf5ba70f30dde9dd6944c9234167b7132e5 analyzed
docker.io/tomcat:latest                                     sha256:8aee1001456a722358557b9b1f6ee8eecad675b36e4be10f9238ccd8293bc856 analyzed
```

Ilustración 33: Imágenes ingestadas por Anchore

8.6 Agregar una nueva imagen de forma manual para su análisis

Con el siguiente comando se puede indicar a Anchore que descargue dicha imagen y proceda a su análisis.

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image add
docker.io/mysql:latest
```

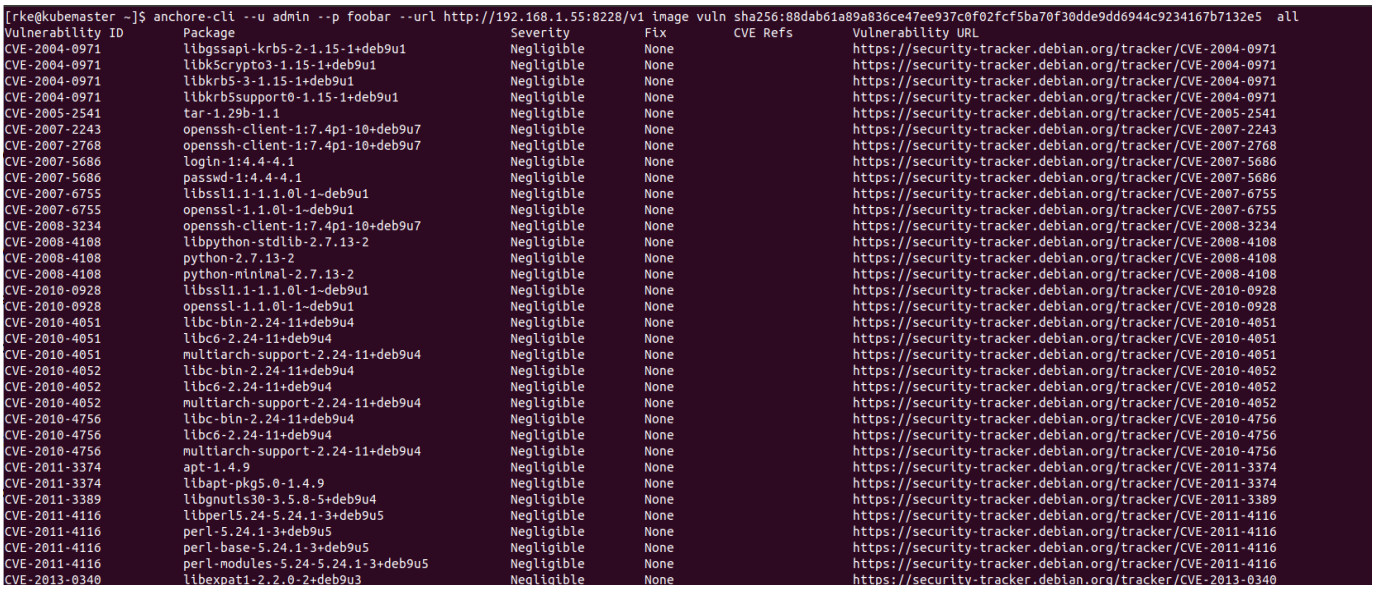
```
Image Digest: sha256:0491ecfc011cdebbd6c9bc2f5cd8fd0bd43f6e9b96caae96fb404eb00381068d
Parent Digest: sha256:7345ce4ce6f0c1771d01fa333b8edb2c606ca59d385f69575f8e3e2ec6695eee
Analysis Status: analyzed
Image Type: docker
Analyzed At: 2019-10-18T01:36:55Z
Image ID: c8ee894bd2bd30f1c5758baf1c4cf2bd49ff7161b29ae404a8562b523fc3f642
Dockerfile Mode: Guessed
Distro: debian
Distro Version: 9
Size: 462202880
Architecture: amd64
Layer Count: 12

Full Tag: docker.io/mysql:latest
Tag Detected At: 2019-10-17T15:33:23Z
```

Ilustración 34: Agregado de una imagen manualmente a Anchore

8.7 Obtener el resultado del análisis de una imagen

A través del siguiente ejemplo, se indicará a Anchore que muestre los resultados del análisis de una imagen de Tomcat versión 7. Cabe destacar que para indicar la imagen en este caso se ha utilizado, el SHA256 que Anchore usa como imagen ID. También podría utilizarse el nombre completo de la imagen.

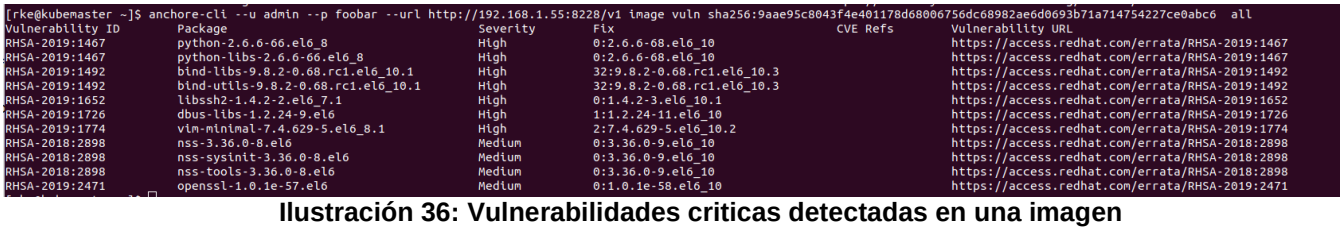
```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln
sha256:78338b6cb2a7414985e898ed518871dec5fd97ee5c8862d8fd88150f1c04d7e3  all
```



```
[rke@kubemaster ~]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln sha256:88dab61a89a836ce47ee937c0f02fcf5ba70f30dde9dd6944c9234167b7132e5  all
Vulnerability ID        Package                                 Severity        Fix         CVE Refs        Vulnerability URL
CVE-2004-0971           libgssapi-krb5-2-1.15-1+deb9u1          Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2004-0971
CVE-2004-0971           libk5crypto3-1.15-1+deb9u1              Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2004-0971
CVE-2004-0971           libkrb5-3-1.15-1+deb9u1                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2004-0971
CVE-2004-0971           libkrb5support0-1.15-1+deb9u1           Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2004-0971
CVE-2005-2541           tar-1.29b-1.1                           Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2005-2541
CVE-2007-2243           openssh-client-1:7.4p1-10+deb9u7        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-2243
CVE-2007-2768           openssh-client-1:7.4p1-10+deb9u7        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-2768
CVE-2007-5686           login-1:4.4-4.1                         Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-5686
CVE-2007-5686           passwd-1:4.4-4.1                        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-5686
CVE-2007-6755           libssl1.1-1.1.0l-1~deb9u1               Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-6755
CVE-2007-6755           openssl-1.1.0l-1~deb9u1                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2007-6755
CVE-2008-3234           openssh-client-1:7.4p1-10+deb9u7        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2008-3234
CVE-2008-4108           libpython-stdlib-2.7.13-2               Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2008-4108
CVE-2008-4108           python-2.7.13-2                         Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2008-4108
CVE-2008-4108           python-minimal-2.7.13-2                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2008-4108
CVE-2010-0928           libssl1.1-1.1.0l-1~deb9u1               Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-0928
CVE-2010-0928           openssl-1.1.0l-1~deb9u1                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-0928
CVE-2010-4051           libc-bin-2.24-11+deb9u4                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4051
CVE-2010-4051           libc6-2.24-11+deb9u4                    Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4051
CVE-2010-4051           multiarch-support-2.24-11+deb9u4        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4051
CVE-2010-4052           libc-bin-2.24-11+deb9u4                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4052
CVE-2010-4052           libc6-2.24-11+deb9u4                    Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4052
CVE-2010-4052           multiarch-support-2.24-11+deb9u4        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4052
CVE-2010-4756           libc-bin-2.24-11+deb9u4                 Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4756
CVE-2010-4756           libc6-2.24-11+deb9u4                    Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4756
CVE-2010-4756           multiarch-support-2.24-11+deb9u4        Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2010-4756
CVE-2011-3374           apt-1.4.9                               Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-3374
CVE-2011-3374           libapt-pkg5.0-1.4.9                     Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-3374
CVE-2011-3389           libgnutls30-3.5.8-5+deb9u4              Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-3389
CVE-2011-4116           libperl5.24-5.24.1-3+deb9u5             Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-4116
CVE-2011-4116           perl-5.24.1-3+deb9u5                    Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-4116
CVE-2011-4116           perl-base-5.24.1-3+deb9u5               Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-4116
CVE-2011-4116           perl-modules-5.24-5.24.1-3+deb9u5       Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2011-4116
CVE-2013-0340           libexpat1-2.2.0-2+deb9u3                Negligible      None                        https://security-tracker.debian.org/tracker/CVE-2013-0340
```

Ilustración 35: Vulnerabilidades detectadas en una imagen

Ejemplo del resultado de análisis de nuestra imagen testigo, que contiene vulnerabilidades. En la captura siguiente se muestran las vulnerabilidades críticas que hacen que Anchore Admission Controller ,como veremos posteriormente, no permita que se ejecute dicha imagen por no cumplir con la política.



```
[rke@kubemaster ~]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln sha256:9aae95c8043f4e401178d68006756dc68982ae6d0693b71a714754227ce0abc6  all
Vulnerability ID        Package                                 Severity        Fix                                 CVE Refs        Vulnerability URL
RHSA-2019:1467          python-2.6.6-66.el6_8                   High            0:2.6.6-68.el6_10                           https://access.redhat.com/errata/RHSA-2019:1467
RHSA-2019:1467          python-libs-2.6.6-66.el6_8              High            0:2.6.6-68.el6_10                           https://access.redhat.com/errata/RHSA-2019:1467
RHSA-2019:1492          bind-libs-9.8.2-0.68.rc1.el6_10.1       High            32:9.8.2-0.68.rc1.el6_10.3                  https://access.redhat.com/errata/RHSA-2019:1492
RHSA-2019:1492          bind-utils-9.8.2-0.68.rc1.el6_10.1      High            32:9.8.2-0.68.rc1.el6_10.3                  https://access.redhat.com/errata/RHSA-2019:1492
RHSA-2019:1652          libssh2-1.4.2-2.el6_7.1                 High            0:1.4.2-3.el6_10.1                          https://access.redhat.com/errata/RHSA-2019:1652
RHSA-2019:1726          dbus-libs-1.2.24-9.el6                  High            1:1.2.24-11.el6_10                          https://access.redhat.com/errata/RHSA-2019:1726
RHSA-2019:1774          vim-minimal-7.4.629-5.el6_8.1           High            2:7.4.629-5.el6_10.2                        https://access.redhat.com/errata/RHSA-2019:1774
RHSA-2018:2898          nss-3.36.0-8.el6                        Medium          0:3.36.0-9.el6_10                           https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2018:2898          nss-sysinit-3.36.0-8.el6                Medium          0:3.36.0-9.el6_10                           https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2018:2898          nss-tools-3.36.0-8.el6                  Medium          0:3.36.0-9.el6_10                           https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2019:2471          openssl-1.0.1e-57.el6                   Medium          0:1.0.1e-58.el6_10                          https://access.redhat.com/errata/RHSA-2019:2471
```

Ilustración 36: Vulnerabilidades criticas detectadas en una imagen

9 Preparación de una imagen testigo vulnerable

Este capítulo, pretende establecer las bases para la creación de una imagen de Docker, que contenga vulnerabilidades y que permita al estudio que estamos realizando servir como imagen testigo para comprobar las diferentes funcionalidades de Anchore. Con la pretensión de conseguir este objetivo, se ha procedido a instalar Docker en una máquina virtual, para poder generar una imagen que posteriormente será subida a un registro privado de Docker Hub.

En el anexo Instalación de Docker sección 13.1 pueden seguirse las instrucciones de instalación de Docker, así como en la sección 13.3 puede ver las instrucciones para crear un registro privado de Docker Hub.

Otra de las configuraciones necesarias será la configuración en este registro de Docker de nuestro registro privado de Docker Hub. Que será el registro que utilizará en nuestro entorno de laboratorio, la parte del clúster y los *namespaces* dedicados al entorno de *test* (test_env)

Basaremos nuestra imagen testigo en una versión de CentOS 6. Para obtenerla ejecutaremos el siguiente comando que nos permitirá ejecutar el contenedor.

```
docker run -t -i  centos:centos6 /bin/bash
```

Tras la descarga de la imagen, nuestro entorno local de Docker procederá a ejecutar el contenedor mostrándonos una Shell, como la siguiente:

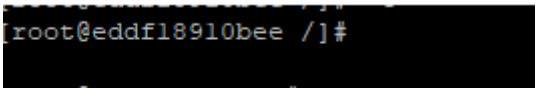


Ilustración 37: *CLI* de un contenedor en ejecución

Para aumentar la vulnerabilidad, hemos elegido instalar una versión de BIND, que contiene una vulnerabilidad con el siguiente comando.

```
yum install bind-9.8.2
```

Una vez finalizadas las modificaciones del contenedor, procederemos a realizar un Docker *commit* para guardar el estado y las modificaciones, que hemos realizado en el contenedor.

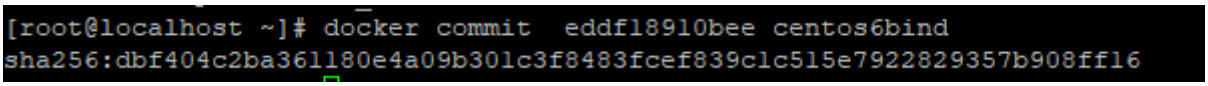


Ilustración 38: Commit de un contenedor de Docker

Una vez guardado el estado, procederemos a hacer un *push* del contenedor a nuestro registro de Docker Hub.

```
docker push dockerhubjuliorepository2019/anchoretest:centos6bind
```

```
[root@localhost ~]# docker push dockerhubjuliorepository2019/anchoretest:centos6bind
The push refers to repository [docker.io/dockerhubjuliorepository2019/anchoretest]
62357e7f8f61: Pushed
af6bf1987c2e: Layer already exists
centos6bind: digest: sha256:c0fedb39f69edec4e0be88294a77e40d88c8dldcdaadla1640d5b9ab9a389437 size: 741
```

Ilustración 39: Upload de un contenedor al repositorio de Docker

Desde este momento, esta imagen estará disponible para descarga y ejecución en nuestro clúster de *test*. Para lo cual, previa ejecución del mismo Anchore, deberá de realizar un análisis de esta imagen, previa ejecución, como se demostrará en la siguiente sección.

10 DevSecOps, uso de Anchore dentro de entornos de integración continua basados en contenedores

A lo largo de este capítulo, se estudiarán las posibilidades que ofrece Anchore dentro de entornos de integración continua, que permitan el análisis de vulnerabilidades en contenedores, que estén en fase de desarrollo.

10.1 Introducción a *DevSecOps*

Durante los últimos años, se está sobrellevando un cambio, en el cual la seguridad se está desplazando hacia la izquierda sin dejar de existir en el resto de procesos que forman la cadena de vida del software. Esta cadena va desde la conceptualización del software, pasando por su desarrollo, paso a operaciones y mantenimiento del mismos.

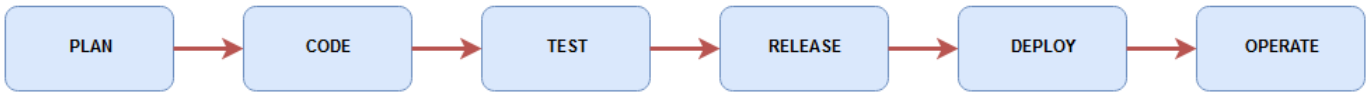


Ilustración 40: Flujo DevOps

En los primeros años, la balanza de la seguridad tenía su peso desplazado hacia la derecha debido a que el mayor esfuerzo de seguridad, tanto en la parte de planificación de la misma, como en la parte de inversión de recursos de seguridad, se hacía sobre la fase de operaciones. Invirtiendo en la protección del conjunto de la infraestructura o buscando tecnologías que ofrezcan protección frente a vulnerabilidades con origen de tipo código.

Sin que se pierda foco a nivel de seguridad en la parte de operaciones anteriormente descrita, la balanza de seguridad se está moviendo hacia la parte izquierda, ganando peso en la parte de desarrollo y *test*.

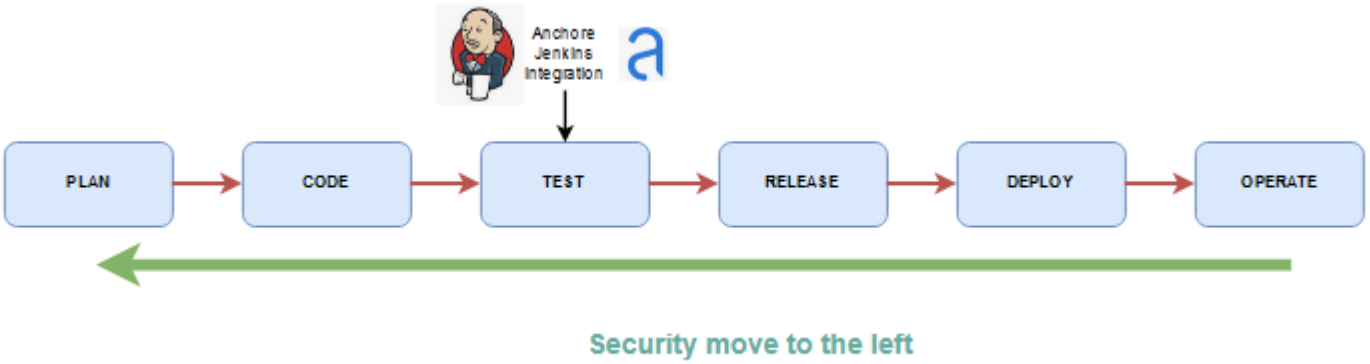


Ilustración 41: Seguridad dentro de *DevSecOps*

A lo largo del capítulo evaluaremos como Anchore puede ayudar en la fase de desarrollo y *test* a resolver vulnerabilidades previo despliegue de contenedores en un clúster de *test*, en entornos DevSecOps.

10.2 Anchore para evaluación de contenedores dentro de un flujo de CI basado en Jenkins

Anchore dispone de distintos plugin, que facilitan la integración de los softwares de automatización de despliegues, como puede ser *Jenkins* con Anchore Engine API. Esto ofrece la posibilidad al desarrollador, de comprobar que vulnerabilidades están presentes en su contenedor, previo a que el mismo sea subido y ejecutado en el clúster de *test*.

Como comprobaremos en capítulos posteriores, tanto la política de admisión a ejecución en el clúster de producción, como la de evaluación de imágenes a través de jenkins será la misma. Haciendo así, que una imagen que no cumpla en la fase de *test* con jenkins no será ejecutada posteriormente en el clúster de producción, por no cumplir con la política de Anchore.

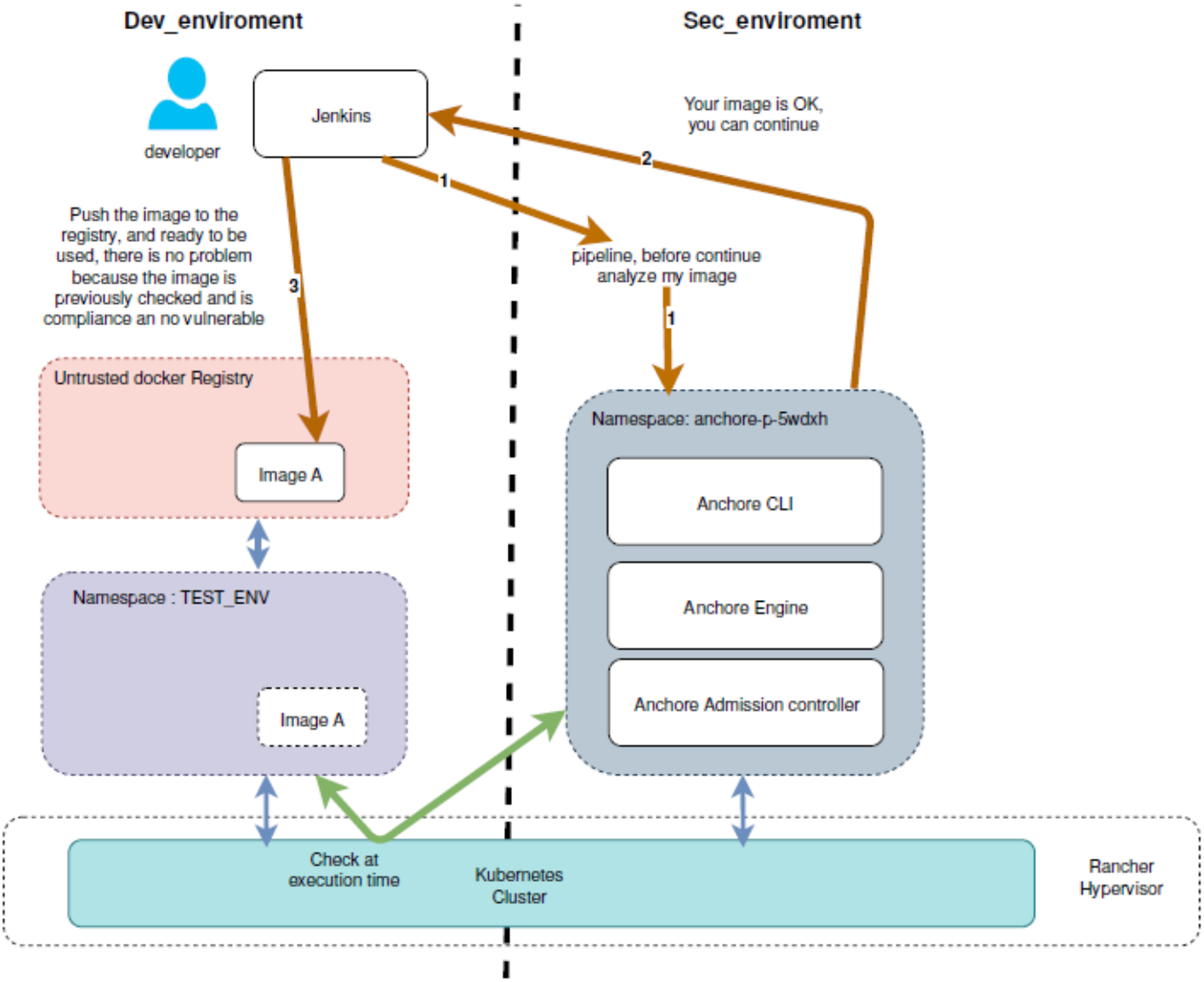


Ilustración 42: Integración de Anchore dentro de entornos CI

1) La solicitud de despliegue realizada por parte del desarrollador, primeramente ha de ser analizada por Anchore, para ello Jenkins lanza la solicitud de análisis a Anchore API Engine haciendo uso de la *API*.

2) Anchore procede a descargar la imagen en caso de que no la tenga y a realizar el análisis de vulnerabilidades sobre la misma. Una vez obtenido un resultado en base a la política de seguridad devuelve el reporte a Jenkins con las vulnerabilidades detectadas.

3) En el caso de que la Imagen cumpla con la política de seguridad y el reporte devuelto por parte de Anchore a Jenkins sea satisfactorio, la imagen del contenedor podrá ser subida al registro de *test*.

10.3 Integración de jenkins con Anchore

La manera en que jenkins se comunicará con la solución de Anchore, será a través del plugin Anchore Container Image Scanner Plugin, haciendo uso de la *API* de Anchore, ofrecida a través del *pod Anchore Engine API*.

Para la instalación y despliegue de Jenkins se ha utilizado el mismo clúster de Kubernetes de laboratorio. En el Anexo *Instalación de Jenkins* sección 13.5.1, se detallará el proceso de instalación de Jenkins sobre Kubernetes.

Continuando con el proceso de integración entre Jenkins y Anchore, procederemos a acceder a la instancia de Jenkins, publicada fuera del clúster a través del puerto 8080, accederemos al apartado administrar Jenkins y administrar *plugins*.

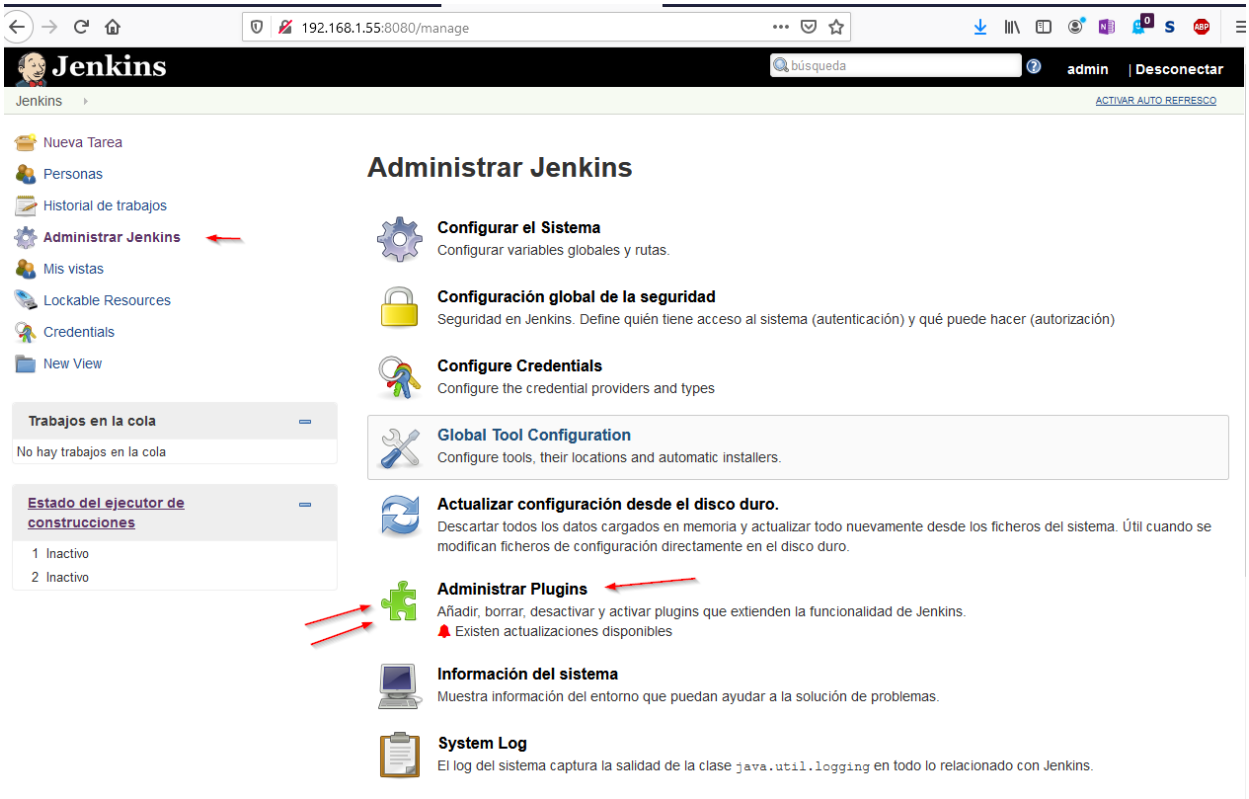


Ilustración 43: Administrador de Plugins de Jenkins

Desde la pestaña denominada “todos los Plugins”, instalaremos el *plugin* de Anchore, que se encargará de conectar con la *API* de la solución de Anchore, y permitirá el análisis de los contenedores durante la fase de *test*. Una vez instalado podremos encontrarlo en la pestaña de Plugins instalados.

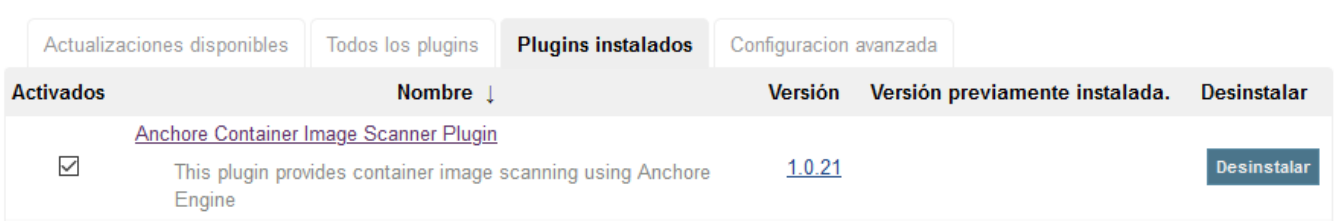


Ilustración 44: Anchore jenkins plugin

Para configurar el plugin, deberemos acceder al apartado Administrar Jenkins, de nuevo y accederemos a configurar el sistema.

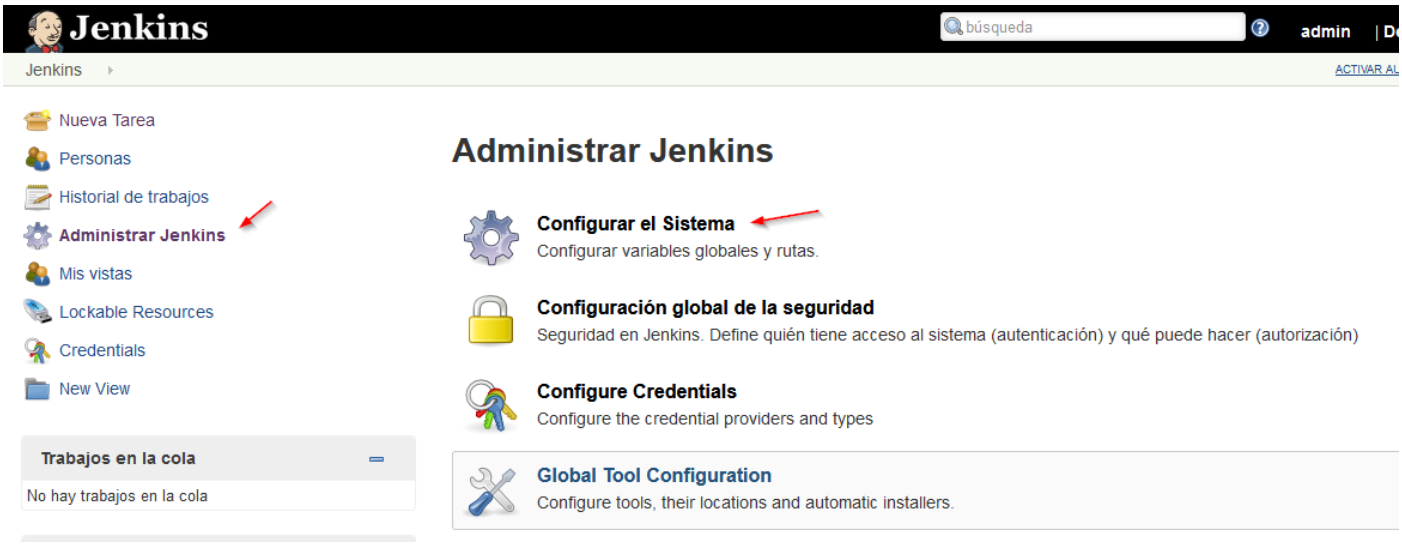


Ilustración 45: Configuración plugin Jenkins II

En el apartado Anchore Container Imagen Scanner, configuraremos la *IP* de la *API* de Anchore publicada por el *pod* Anchore Engine API y las credenciales de acceso a la *API*.

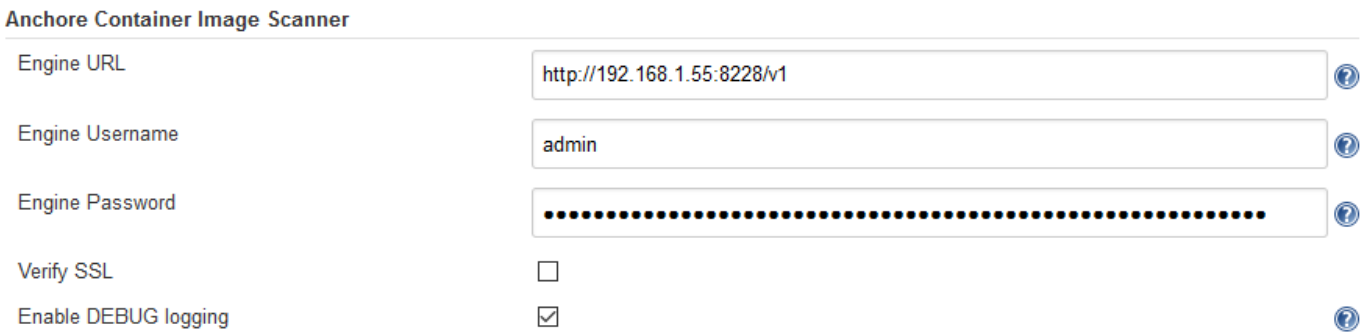


Ilustración 46: Configuración plugin Jenkins II

10.4 Comprobación Jenkins con Anchore

En esta sección se procederá, a comprobar el funcionamiento de Anchore y los resultados obtenidos en conjunto con Jenkins. Para ello se creará un pipeline que generará un contenedor a modo de emulación del flujo normal de despliegue seguido por los desarrolladores en entornos de CI.

Pulsaremos sobre nueva tarea, daremos un nombre a la tarea y elegiremos *pipeline* y pulsaremos ok.

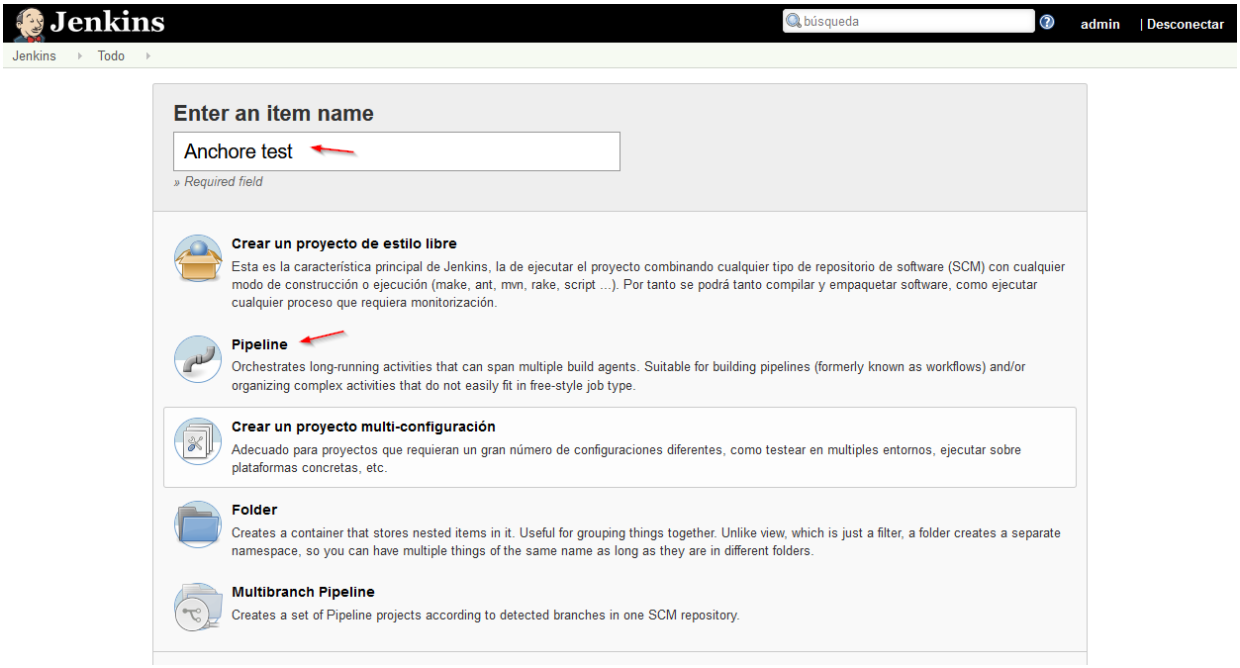


Ilustración 47: Creación de una Pipeline de Jenkins I

Sobre la configuración del pipeline, todas las configuraciones serán dejadas por defecto y en el cuadro de *script* de pipeline, utilizaremos el siguiente código:

```
node {
def imageLine = 'docker.io/dockerhubjuliorepository2019/anchoretest:centos6 '
 //   docker.io/dockerhubjuliorepository2019/anchoretest:ubuntumysql
 //   def imageLine = 'docker.io/dockerhubjuliorepository2019/anchoretest:ubuntumysql'
//def imageLine = 'docker.io/tomcat:latest'
writeFile file: 'anchore_images', text: imageLine
anchore name: 'anchore_images'
}
```

Quedando el código insertado sobre el cuadro de dialogo de *script* de la siguiente forma:

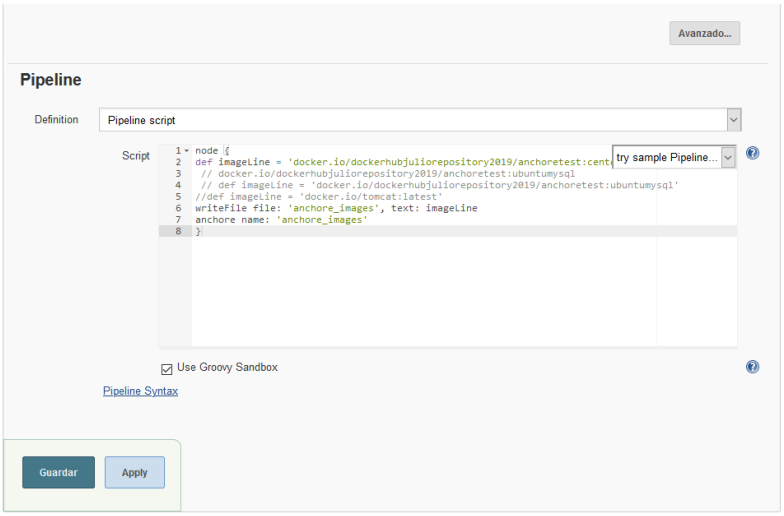


Ilustración 48: Creación de un pipeline de jenkins II

Una vez finalizado, pulsaremos sobre guardar y aplicar. En el siguiente menú contextual pulsaremos sobre construir ahora para ejecutar la pipeline.

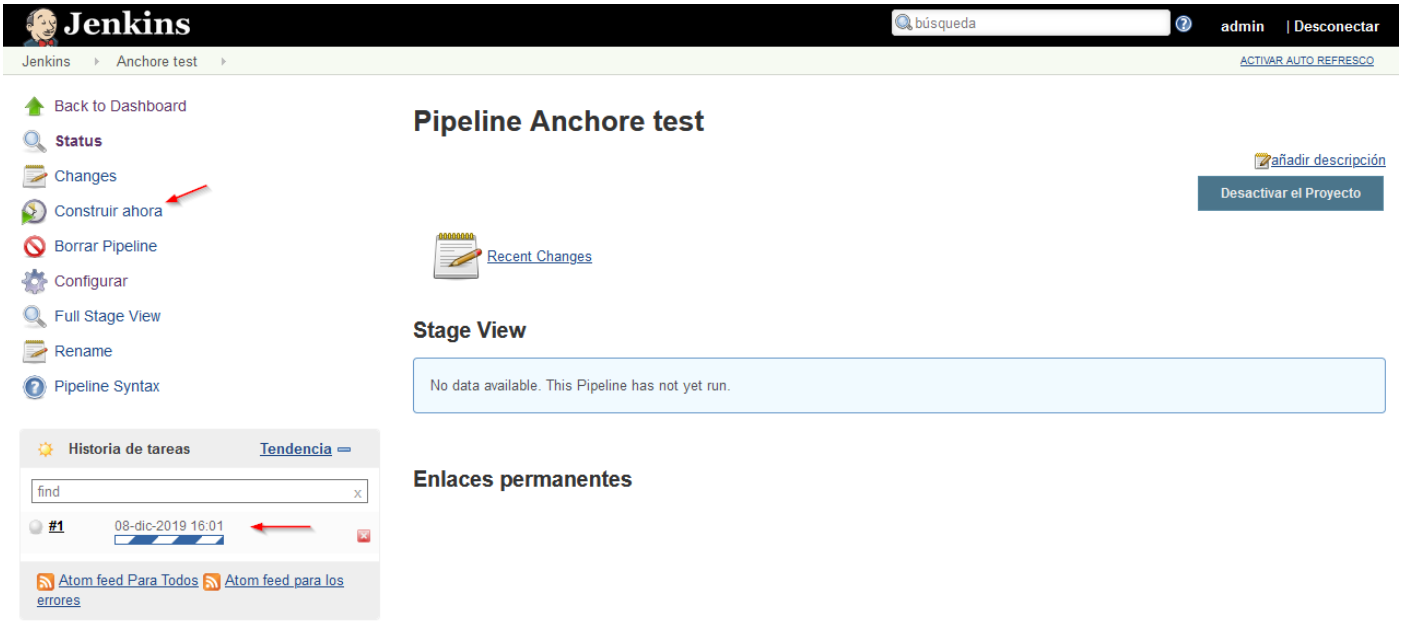


Ilustración 49: Ejecución de un pipeline Jenkins I

Se comprobará que la tarea ha quedado finalizada en rojo y pulsando sobre ella.

Ilustración 50: Ejecución de un pipeline Jenkins II

Pulsando sobre ella podremos acceder al resultado de la tarea y en concreto al resultado Anchore Report, que es el que evito que la tarea se completase.

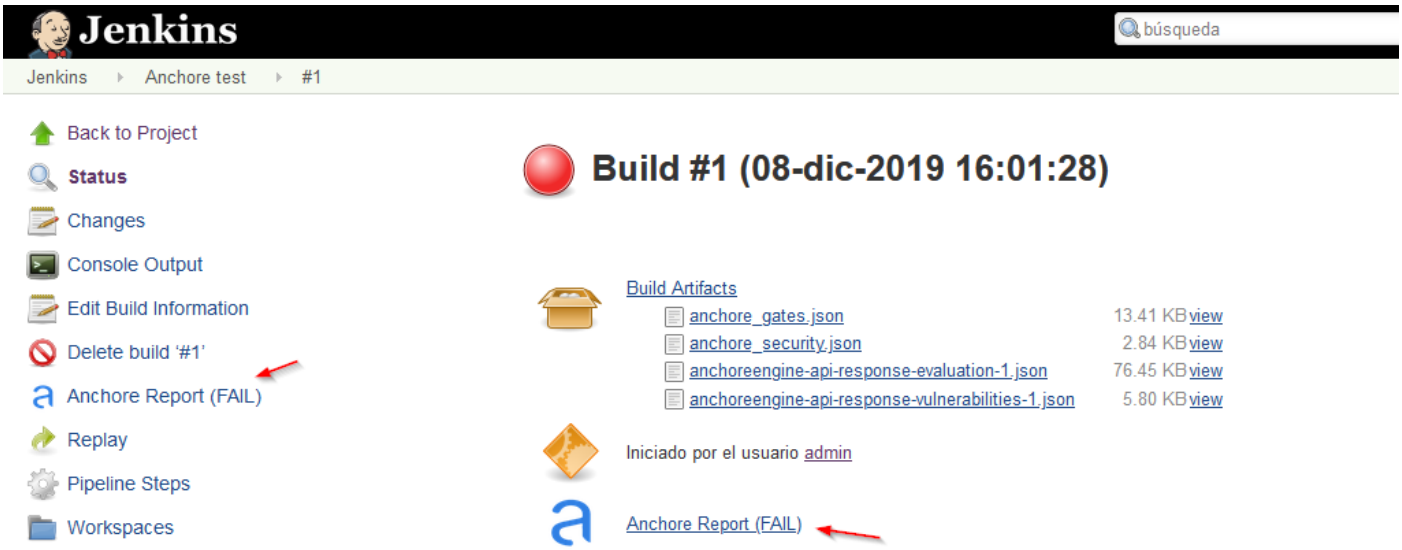


Ilustración 51: Anchore report en Jenkins

Pulsando sobre el reporte de Anchore, comprobaremos que políticas de Anchore no cumple el contenedor en fase de desarrollo, y que vulnerabilidades encontró y su nivel de criticidad. Así como, el uso de determinados usuarios no permitidos en el entorno de ejecución del contenedor.

Como se puede comprobar en la capturas inferiores según nuestra política la evaluacion del cotnenedor ha generado 8 *stop* actions, algunas de ellas basadas en vulnerabilidades sobre BIND the tipo alto y otras sobre la existencia del usuario Root dentro del propio entorno del contenedor.

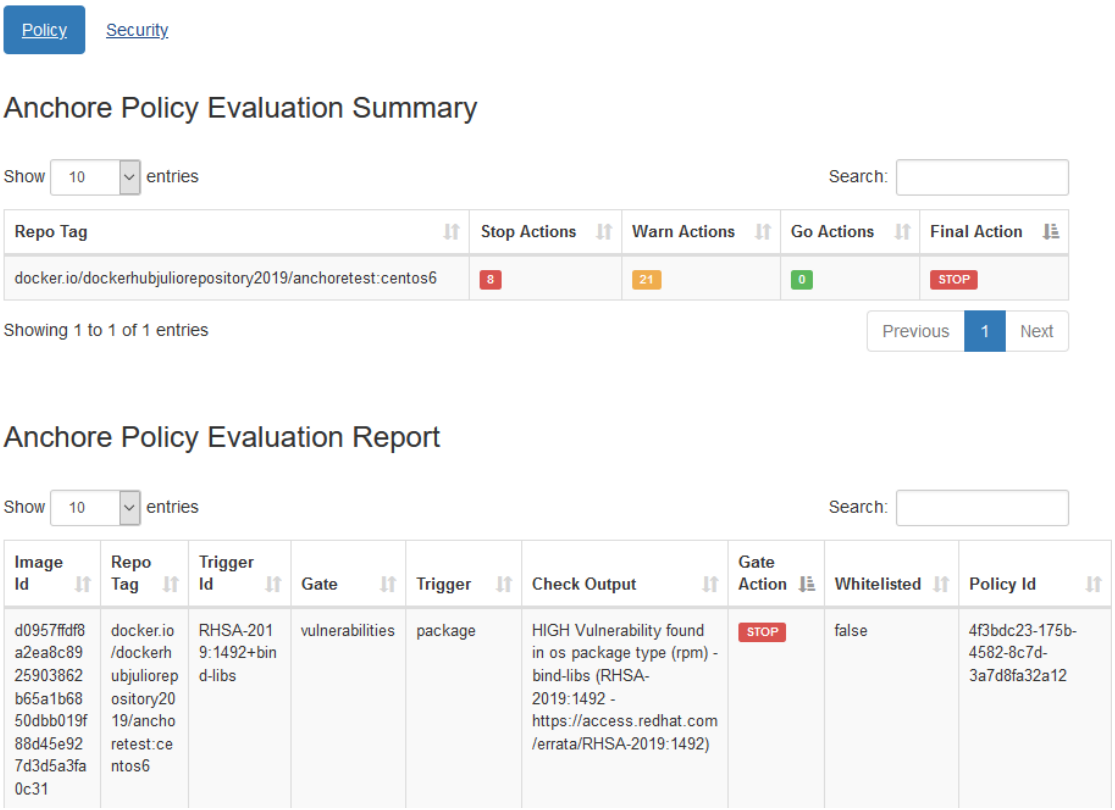


Ilustración 52: Anchore report en Jenkins (*policy* evaluation I)

d0957ffdf8a2ea8c8925903862b65a1b6850dbb019f88d45e927d3d5a3fa0c31	docker.io/dockerhubjuliorepository2019/anchoretest:centos6	68e630cef4a8533b139875aa5fc54da5	dockerfile	effective_user	User root found as effective user, which is explicity not allowed list	STOP	false	cb417967-266b-4453-bfb6-9acf67b0bee5

Ilustración 53: Anchore report en Jenkins (*policy* evaluation II)

En la siguiente captura puede comprobarse como el reporte generado por Anchore, también da información detallada sobre las vulnerabilidades, la versión en la que se solucionó la vulnerabilidad y acceso a *URL* que puedan ser de interés para solución de la vulnerabilidad.

Common Vulnerabilities and Exposures (CVE) List

Show 10 entries Search:

Tag	CVE ID	Severity	Vulnerability Package	Fix Available	URL
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1492	High	bind-libs-9.8.2-0.68.rc1.el6_10.1	32:9.8.2-0.68.rc1.el6_10.3	https://access.redhat.com/errata/RHSA-2019:1492
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1492	High	bind-utils-9.8.2-0.68.rc1.el6_10.1	32:9.8.2-0.68.rc1.el6_10.3	https://access.redhat.com/errata/RHSA-2019:1492
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1726	High	dbus-libs-1.2.24-9.el6	1:1.2.24-11.el6_10	https://access.redhat.com/errata/RHSA-2019:1726
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1652	High	libssh2-1.4.2-2.el6_7.1	0:1.4.2-3.el6_10.1	https://access.redhat.com/errata/RHSA-2019:1652
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1467	High	python-2.6.6-66.el6_8	0:2.6.6-68.el6_10	https://access.redhat.com/errata/RHSA-2019:1467
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1467	High	python-libs-2.6.6-66.el6_8	0:2.6.6-68.el6_10	https://access.redhat.com/errata/RHSA-2019:1467
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2019:1774	High	vim-minimal-7.4.629-5.el6_8.1	2:7.4.629-5.el6_10.2	https://access.redhat.com/errata/RHSA-2019:1774
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2018:2898	Medium	nss-3.36.0-8.el6	0:3.36.0-9.el6_10	https://access.redhat.com/errata/RHSA-2018:2898
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2018:2898	Medium	nss-sysinit-3.36.0-8.el6	0:3.36.0-9.el6_10	https://access.redhat.com/errata/RHSA-2018:2898
docker.io/dockerhubjuliorepository2019/anchoretest:centos6	RHSA-2018:2898	Medium	nss-tools-3.36.0-8.el6	0:3.36.0-9.el6_10	https://access.redhat.com/errata/RHSA-2018:2898

Showing 1 to 10 of 11 entries Previous 1 2 Next

Ilustración 54: Anchore report en Jenkins (CVE list)

Como ha podido comprobarse durante este capítulo, la integración de Anchore con Jenkins permite realizar un análisis de vulnerabilidades y de recomendaciones de seguridad durante la fase de *test* de un proyecto de desarrollo. Esto da lugar a que puedan ser solventadas muchas vulnerabilidades a nivel de paquete y de contenedor, previas a la puesta en producción del contenedor.

11 Control de admisión en Kubernetes basado en Anchore

La función del control de admisión es evitar que se ejecuten en nuestro clúster, imágenes que no cumple con nuestra política por contener vulnerabilidades críticas. A continuación se muestra un grafico que indica cada una de las fases que componen el control de admision

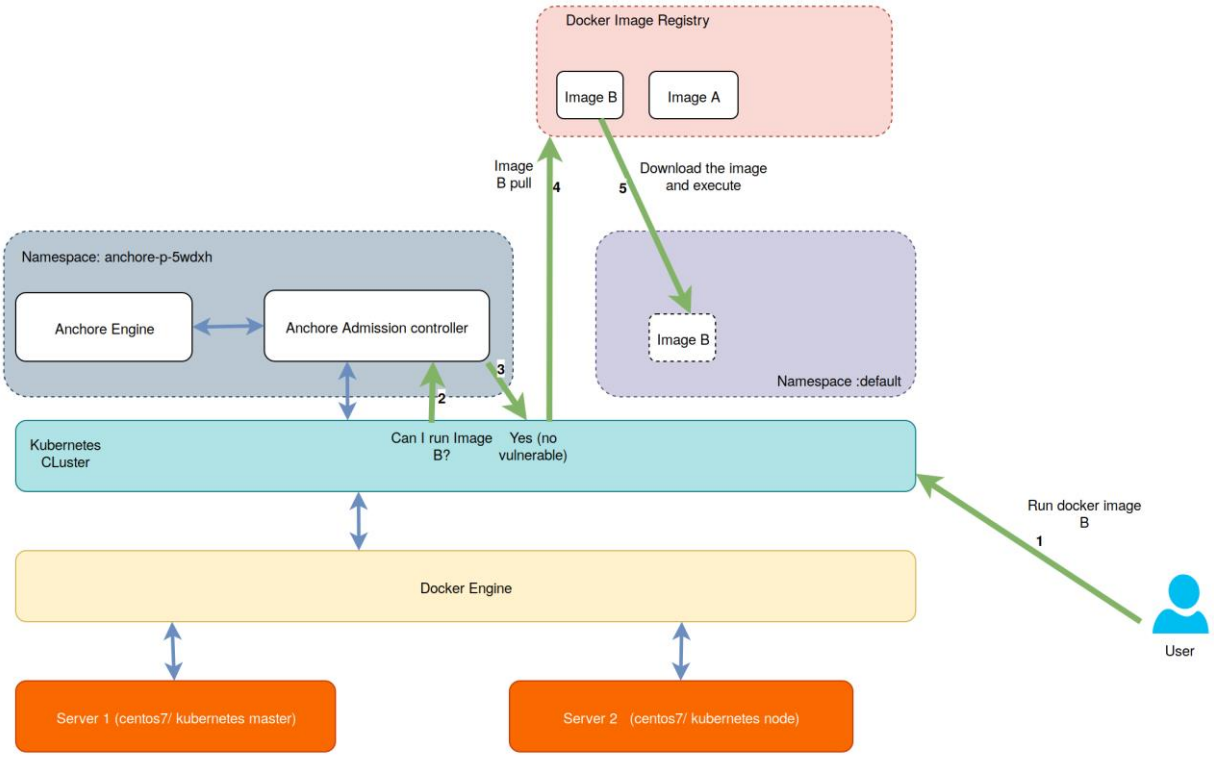


Ilustración 55: Flujo Anchore Admission Controller

1) Solicitud de ejecución de la imagen B.

2) El clúster de Kubernetes solicita la validación a Anchore.

3) Chequea si ya tiene evaluada previamente esta imagen, para darle un consentimiento al clúster o por el contrario denegarle la ejecución. En caso de que no tenga una evaluación previa de la imagen la descargará y la analizará. Como último paso indicara al clúster de Kubernetes que es este caso puede ejecutar dicha imagen ya que no contiene vulnerabilidades.

4) El clúster de Kubernetes procede a ejecutar la imagen lanzando un *pull* sobre la imagen.

5) El clúster la descarga la imagen del registro genera el contenedor y lo ejecuta dentro del *namespace* correspondiente.

11.1 Testeo del control de admisión

Para testear el control de admisión de Anchore, intentaremos ejecutar la imagen vulnerable que preparamos anteriormente y que subimos a nuestro registro untrusted.

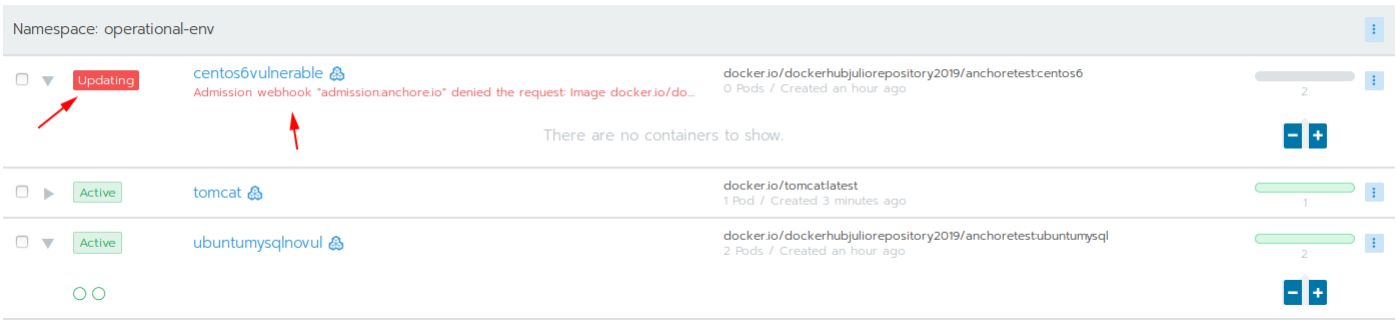


Ilustración 56: Intento de ejecución de una imagen vulnerable

Imagen conocida corriendo dentro de dos nodos distintos.

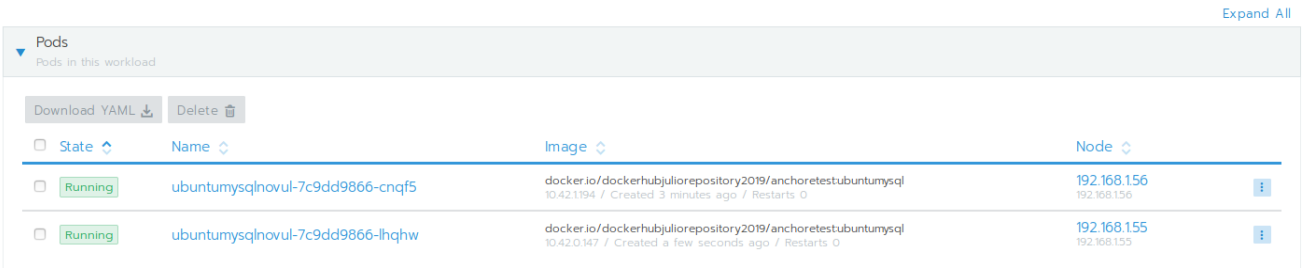


Ilustración 57: Ejecución de una imagen no vulnerable

11.2 Lanzamiento de una imagen no conocida

Durante el proceso normal de ejecución dentro de *pods*, dentro de un clúster de Kubernetes en producción, puede darse el caso en que la imagen que se requiere ejecutar no haya sido validada previamente y que por tanto requiera análisis por parte de Anchore.

En la siguiente captura, vemos como Anchore ha pausado la ejecución del pod durante el proceso de análisis de Anchore. En este caso el propio motor de Anchore procederá a descargar la imagen, evaluarla, obtener un veredicto y darle este veredicto al propio clúster de Kubernetes para que proceda con la ejecución si procede.

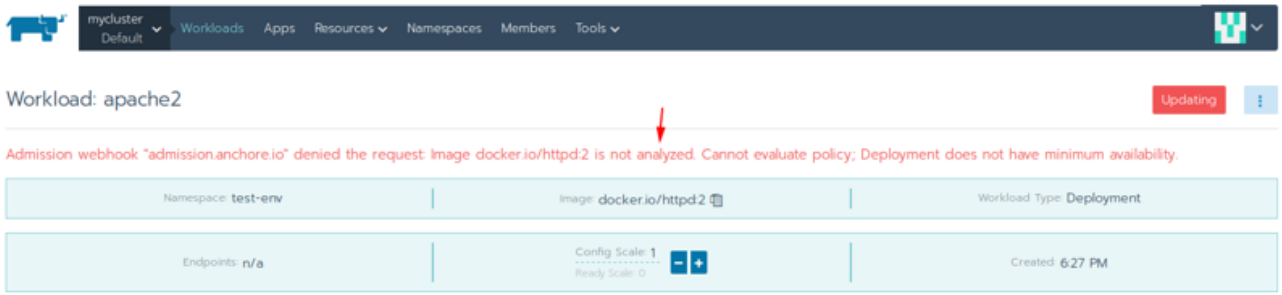


Ilustración 58: Análisis de una imagen no conocida

En la siguiente captura podemos ver en detalle el resultado dado por Anchore tras el análisis de la imagen. Podemos ver que el status es pass por lo que el clúster de Kubernetes podrá ejecutar la imagen.

```
[rke@kubemaster ~]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   evaluate check docker.io/tomcat:latest
Image Digest: sha256:8aee1001456a722358557b9b1f6ee8eecad675b36e4be10f9238ccd8293bc856
Full Tag: docker.io/tomcat:latest
Status: pass
Last Eval: 2019-11-05T17:35:07Z
Policy ID: 2c53a13c-1765-11e8-82ef-23527761d060
```

Ilustración 59: Detalle de una imagen analizada por Anchore

11.3 Detección de vulnerabilidades sobre paquetes instalados en la imagen

En la sección de preparación de la imagen testigo, para verificar el funcionamiento de Anchore se instaló una versión del servidor *DNS Bind*, que contenía vulnerabilidades críticas. Con el fin de comprobar si Anchore además de analizar la paquetería instalada por defecto en la imagen, es capaz de analizar y detectar vulnerabilidades en paquetes instalados a posteriori o durante la construcción del mismo.

El control de admisión instalado en el clúster al intentar desplegar el *pod*, muestra la advertencia que indica que esa imagen no cumple con la política y haciendo una llamada con Anchore *CLI*, se puede comprobar como efectivamente ha capturado también las vulnerabilidades críticas que este paquete de *Bind* contiene, como se muestra en la imagen inferior.

```
[rke@kubemaster ~]$ anchore-cli --debug --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln sha256:9aae95c8043f4e401178d68006756dc68982ae6d0693b71a714754227ce0abc6 all
INFO:anchorecli.clients.apiexternal:As Account = None
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /v1 HTTP/1.1" 200 5
INFO:anchorecli.clients.apiexternal:As Account = None
DEBUG:anchorecli.clients.apiexternal:GET url=http://192.168.1.55:8228/v1/images/sha256%3A9aae95c8043f4e401178d68006756dc68982ae6d0693b71a714754227ce0abc6/vuln/all?vendor_only=True
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /v1/images/sha256%3A9aae95c8043f4e401178d68006756dc68982ae6d0693b71a714754227ce0abc6/vuln/all?vendor_only=True HTTP/1.1" 200 5941
DEBUG:anchorecli.cli.utils:fetched httpcode from response: 200
Vulnerability ID        Package                                 Severity        Fix                             CVE Refs        Vulnerability URL

RHSA-2019:1467          python-2.6.6-66.el6_8                   High            0:2.6.6-68.el6_10                               https://access.redhat.com/errata/RHSA-2019:1467
RHSA-2019:1467          python-libs-2.6.6-66.el6_8              High            0:2.6.6-68.el6_10                               https://access.redhat.com/errata/RHSA-2019:1467
RHSA-2019:1492          bind-libs-9.8.2-0.68.rc1.el6_10.1       High            32:9.8.2-0.68.rc1.el6_10.3                      https://access.redhat.com/errata/RHSA-2019:1492
RHSA-2019:1492          bind-utils-9.8.2-0.68.rc1.el6_10.1      High            32:9.8.2-0.68.rc1.el6_10.3                      https://access.redhat.com/errata/RHSA-2019:1492
RHSA-2019:1652          libssh2-1.4.2-2.el6_7.1                 High            0:1.4.2-3.el6_10.1                              https://access.redhat.com/errata/RHSA-2019:1652
RHSA-2019:1726          dbus-libs-1.2.24-9.el6                  High            1:1.2.24-11.el6_10                              https://access.redhat.com/errata/RHSA-2019:1726
RHSA-2019:1774          vim-minimal-7.4.629-5.el6_8.1           High            2:7.4.629-5.el6_10.2                            https://access.redhat.com/errata/RHSA-2019:1774
RHSA-2018:2898          nss-3.36.0-8.el6                        Medium          0:3.36.0-9.el6_10                               https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2018:2898          nss-sysinit-3.36.0-8.el6                Medium          0:3.36.0-9.el6_10                               https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2018:2898          nss-tools-3.36.0-8.el6                  Medium          0:3.36.0-9.el6_10                               https://access.redhat.com/errata/RHSA-2018:2898
RHSA-2019:2471          openssl-1.0.1e-57.el6                   Medium          0:1.0.1e-58.el6_10                              https://access.redhat.com/errata/RHSA-2019:2471
[rke@kubemaster ~]$
```

Ilustración 60: Vulnerabilidad detectada en Bind *DNS* server

Si accedemos a la web que contiene el reporte de RedHat mostrada por Anchore, referente a estos dos paquetes [AD-06] , comprobamos que esta versión está afectada por la vulnerabilidad CVE-2018-5743 y categorizada con un CVSSv3 de 8.6 [AD-07].

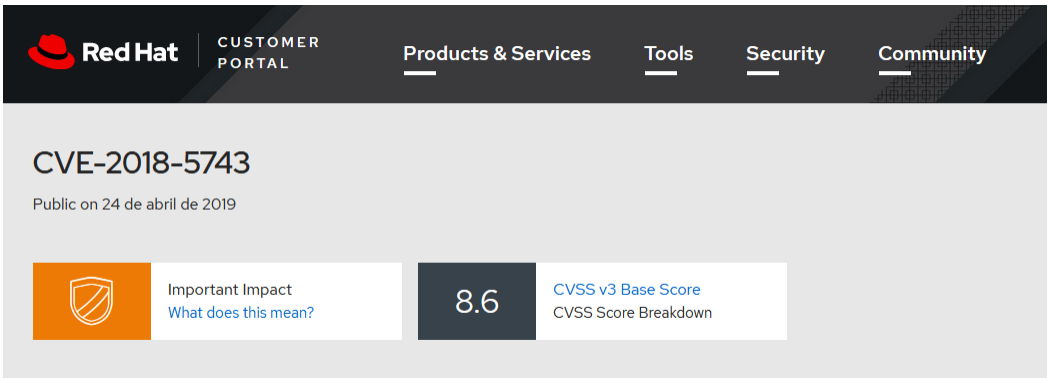


Ilustración 61: Reporte red-hat sobre una vulnerabilidad

12 Conclusiones

A lo largo del desarrollo del proyecto se ha tenido la oportunidad de trabajar con muchas tecnologías que son novedosas y ampliamente utilizadas actualmente, sobre las cuales es más que interesante analizar las diferentes soluciones de seguridad.

A nivel personal el esfuerzo para realizar el proyecto ha venido derivado en gran parte por la curva de aprendizaje sobre distintas tecnologías, así como por su puesta en marcha. Debido a que posteriormente era necesario desarrollar las componentes de seguridad estudiadas las cuales eran el verdadero objetivo del proyecto.

Sin embargo, todo este esfuerzo de puesta en marcha y estudio de las componentes de seguridad objetivo, ha supuesto una gran satisfacción y ha servido para que a nivel personal pudiese aprender y desarrollar mis conocimientos sobre este nuevo paradigma tecnológico que son los contenedores y la gestión de los mismos.

Es conocido que el ámbito de la seguridad es sumamente amplio dentro de los contenedores y su gestión, pudiendo aplicarse en cada una de las diferentes capas que lo componen, como podemos ver a alto nivel en el siguiente diagrama:

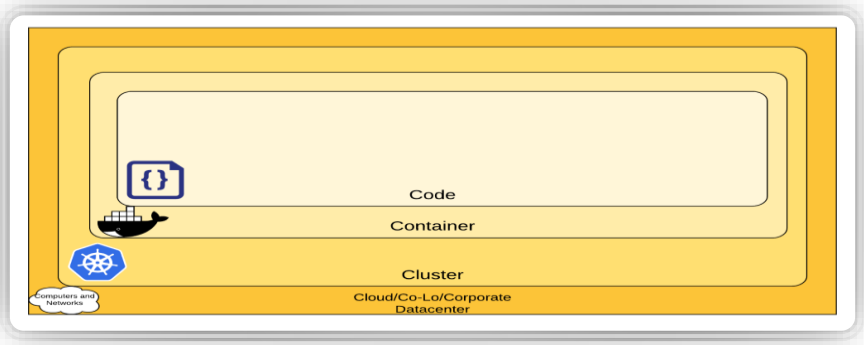


Ilustración 62: Componentes de un entornos basado en contenedores.

Es por ello por lo que este proyecto se ha basado en el estudio del análisis estático de vulnerabilidades previo a ejecución de los contenedores en su ámbito correspondiente con carácter preventivo.

Como posibles vías de ampliación del proyecto se podrían realizar un estudio sobre técnicas de análisis del comportamiento de los contenedores durante su ejecución. Por otro lado sería sumamente interesante realizar un estudio sobre la toma de logs y evidencias en un entorno de producción con contenedores, que sea de utilidad en análisis de incidentes de seguridad en el que se vean implicados contenedores.

13 Anexos.

13.1 Instalación de Docker

La siguiente referencia bibliográfica perteneciente a la documentación de Docker brinda información detallada sobre el proceso de instalación de Docker [AD-08]

A continuación, se describe de forma breve el resumen de los comandos utilizados para la instalación de Docker sobre CentOS.

Instalación previa de la paquetería requerida yum-utils, yum-config-manager, device-mapper-persistent-data lvm2 y devicemapper.

```
$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

Configuración de repositorio de Docker:

```
$ sudo yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
```

Instalación de Docker:

```
$ sudo yum install docker-ce docker-ce-cli containerd.io
```

Inicio del servicio de Docker:

```
$ sudo systemctl start docker
```

Testeo de Docker, ejecución del contenedor hello-world:

```
$ sudo docker run hello-world
```

Captura del resultado de ejecución correcto:

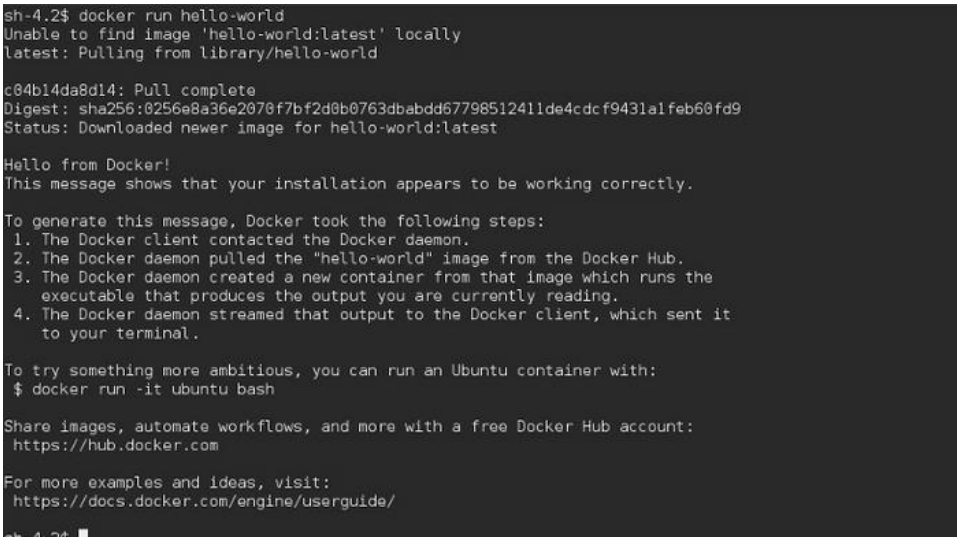


Ilustración 63: Ejecución contenedor Hello-World

13.2 Instalación clúster de Kubernetes basado en *RKE*.

En esta sección se tratará la instalación del clúster de Kubernetes. Con el objetivo de simplificar el despliegue, se ha elegido Rancher Kubernetes Edition. A diferencia de un clúster de Kubernetes convencional, *RKE* únicamente requiere que el servidor tenga instalado Docker, ya que todos los servicios necesarios para montar el clúster en lugar de ser servicios corriendo sobre el propio servidor son a su vez también contenedores.

Como se comentó en el párrafo anterior *RKE* requiere únicamente que cada uno de los servidores que compondrán el clúster tengan instalado Docker y a su vez que sean accesibles desde el servidor *máster* por SSH para ser desplegados mediante *script*.

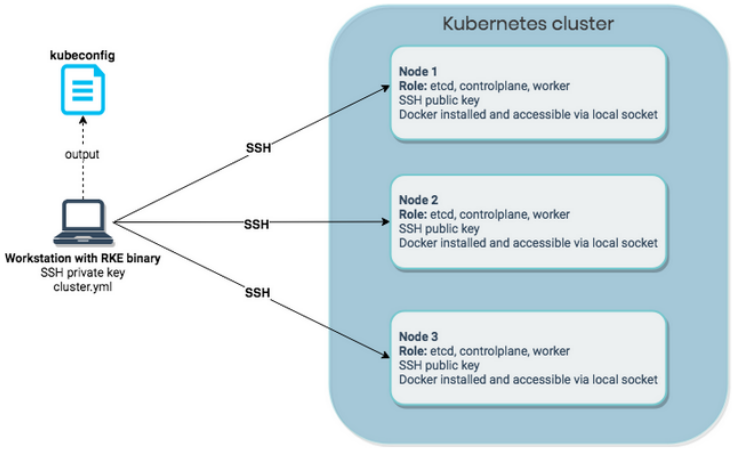


Ilustración 64: Despliegue de *RKE* mediante *script*

Rancher pone a disposición de los usuarios un *script* que se encargara de automatizar el despliegue, disponible en la siguiente *URL* [AD-09].

Una vez descargado cambiaremos el nombre al *script* por *rke*:

```
mv rke_linux-amd64 rke
```

Permisos de ejecución al *script* con el siguiente comando:

```
chmod +x rke
```

Para testear su funcionamiento ejecutaremos:

```
rke --version
```

Una vez comprobado que el *script* es funcional deberemos generar un fichero cluster.yaml que contendrá toda la información referente a los nodos que formaran parte del clúster. Dentro de esta información relevante se incluye la IP y el rol que tendrá cada nodo del clúster. También pueden agregarse otras configuraciones como los registros de Docker que tendrá dados de alta el clúster y certificados que puedan ser de interés, entre otras opciones.

Un ejemplo de fichero de configuración puede ser encontrado en la siguiente *URL* [AD-10].

El último paso será levantar el clúster haciendo uso del siguiente comando:

```
rke up
```

Este comando ejecutará el *script* con el fichero de configuración indicado previamente y se encargará de desplegar el clúster de Kubernetes.

Una vez desplegados todos los nodos del clúster el *script* nos devolverá el fichero kubeconfig que contiene la información necesaria incluyendo las claves públicas para poder conectar con el clúster mediante kubectl.

```
kube_config_cluster.yml
```

Para comprobar si el clúster levantó correctamente, fijaremos el kubeconfig como variable de entorno con el siguiente comando:

```
export KUBECONFIG=$(pwd)/kube_config_cluster.yml
```

Procederemos a ejecutar:

```
kubectl get nodes
```

Si el despliegue ha sido correcto obtendremos la información de los nodos que forman el clúster:

```
[rke@kubemaster ~]$ kubectl get nodes
NAME            STATUS   ROLES                      AGE   VERSION
192.168.1.55    Ready    controlplane,etcd,worker   94d   v1.14.6
192.168.1.56    Ready    controlplane,etcd,worker   94d   v1.14.6
```

Ilustración 65: Información sobre nodos del clúster

13.3 Creación de un registro privado de Docker Hub

Esta sección explicará la creación de un registro privado de Docker Hub, para ello accederemos al portal web de Docker a través de la siguiente *URL* [RD-16]

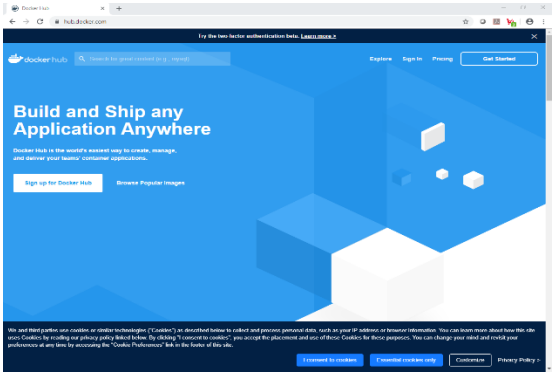


Ilustración 66: Docker Hub web

Una vez dados de alta accederemos a la configuración del registro. Para el entorno de laboratorio se ha procedido a crear un registro privado, donde han quedado albergadas las imágenes de testeo utilizadas durante el proyecto y de las que hace uso el clúster de Kubernetes de laboratorio:

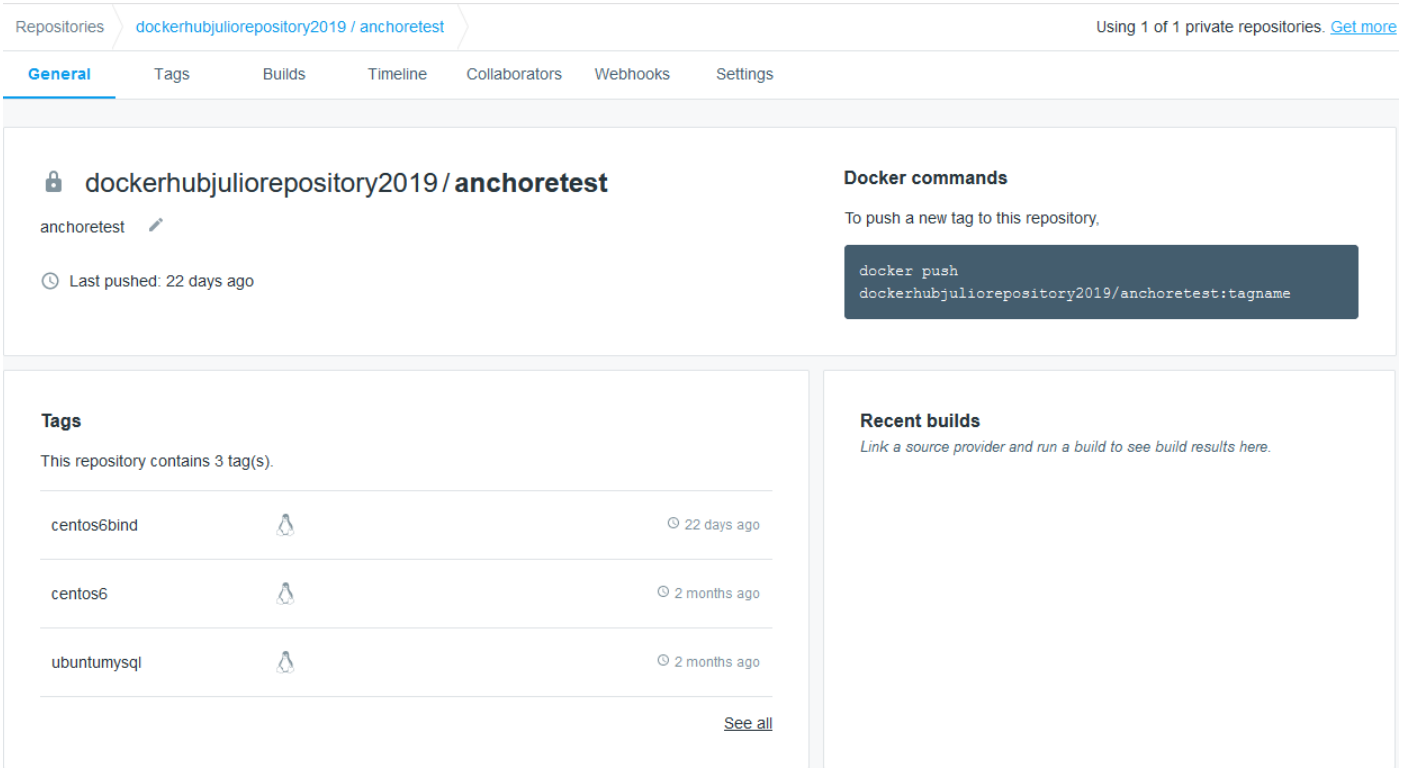


Ilustración 67: Repositorio Docker Hub con las imágenes usadas.

13.4 Instalación y configuración de Anchore

13.4.1 Instalación Anchore

Basándonos en la facilidad de despliegue de aplicaciones de Rancher, se hará uso del catálogo de aplicaciones que brinda dicho hipervisor. Para desplegar la solución de Anchore, en el apartado Apps se procederá a hacer click sobre *Launch*, dándonos así acceso al catálogo de aplicaciones que ofrece Rancher.

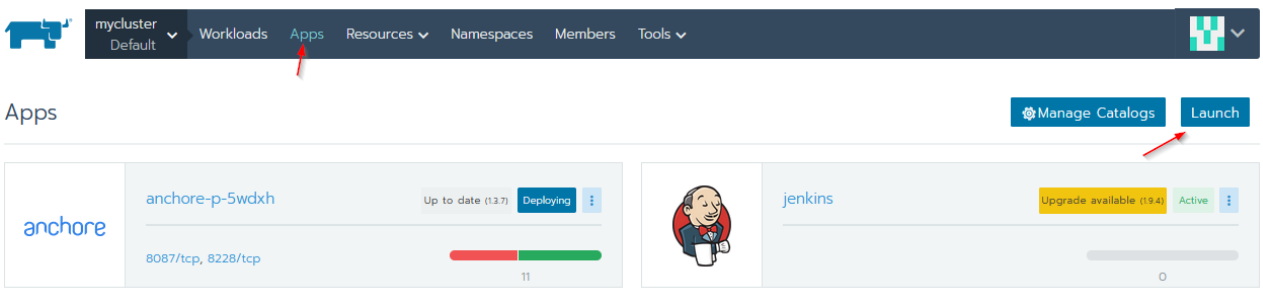


Ilustración 68: Despliegue Anchore en Kubernetes I

El catálogo de aplicaciones disponibles está basado principalmente en Helm y varía en función de los repositorios y Docker Registries dados de alta. Para proceder con la instalación de Anchore haremos *click* sobre *view details*.

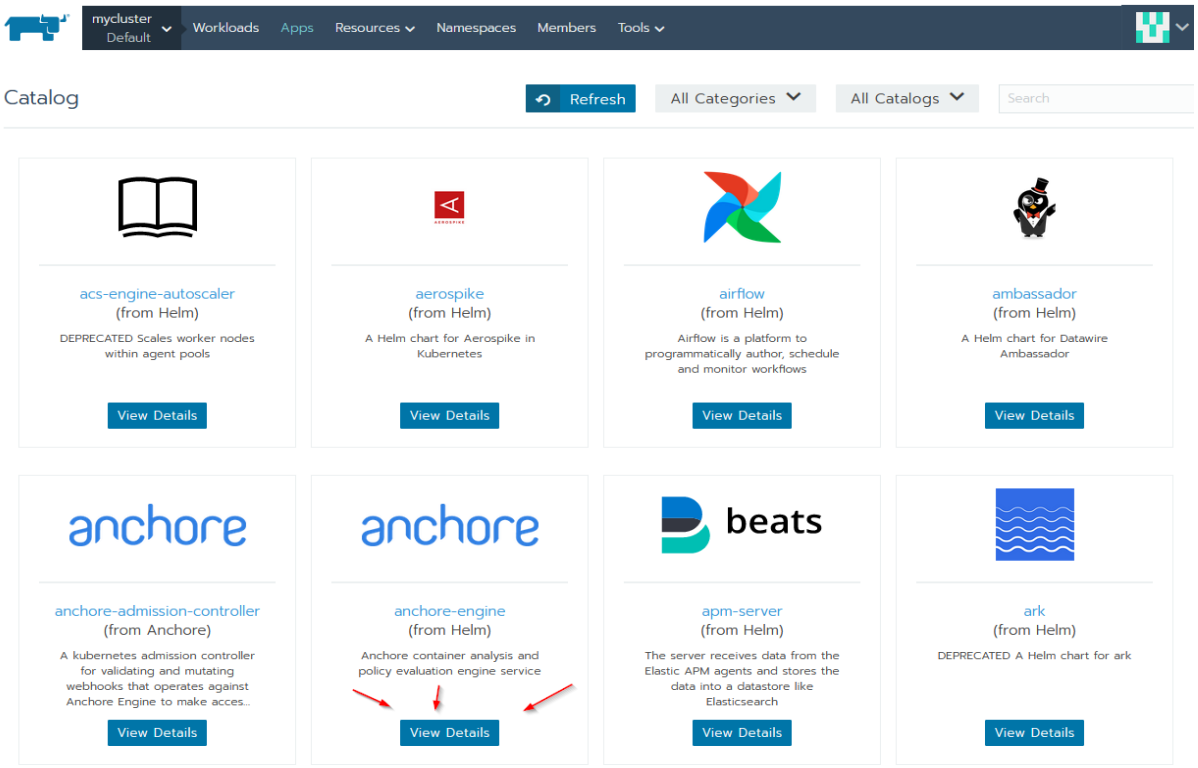


Ilustración 69: Despliegue Anchore en Kubernetes II

Como parte de las configuraciones necesarias para proceder con el despliegue será necesario, elegir la versión de Anchore que se desea desplegar. Así como, el *namespace* donde será desplegada.

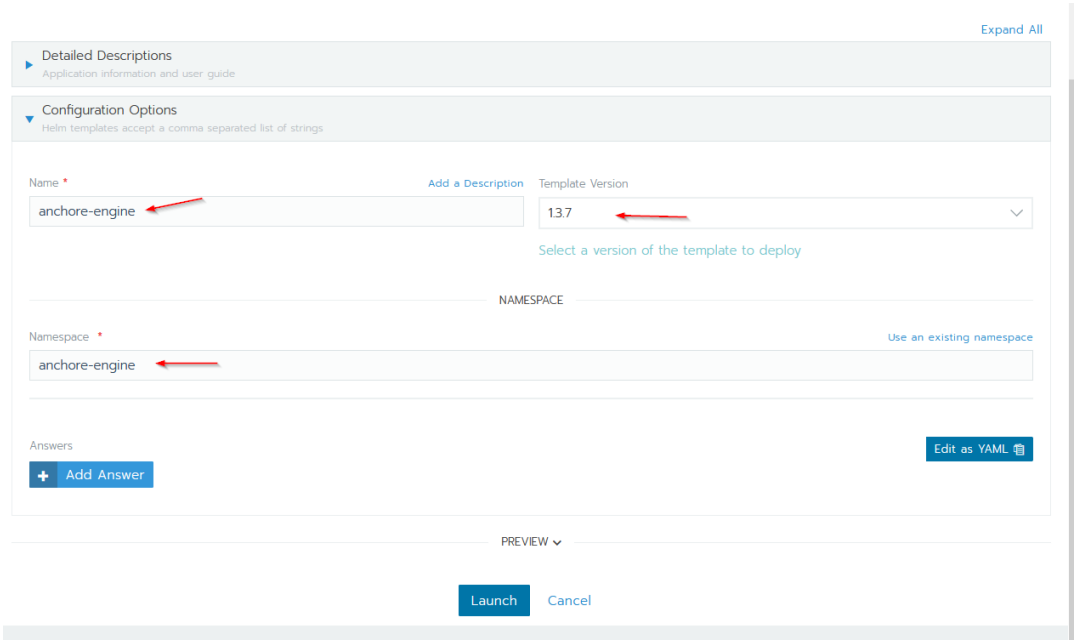


Ilustración 70: Configuración despliegue Anchore

Como resultado y tras la descarga de todas las imágenes, se obtiene un *namespace* que contiene todos los *pods* de Anchore funcionando.

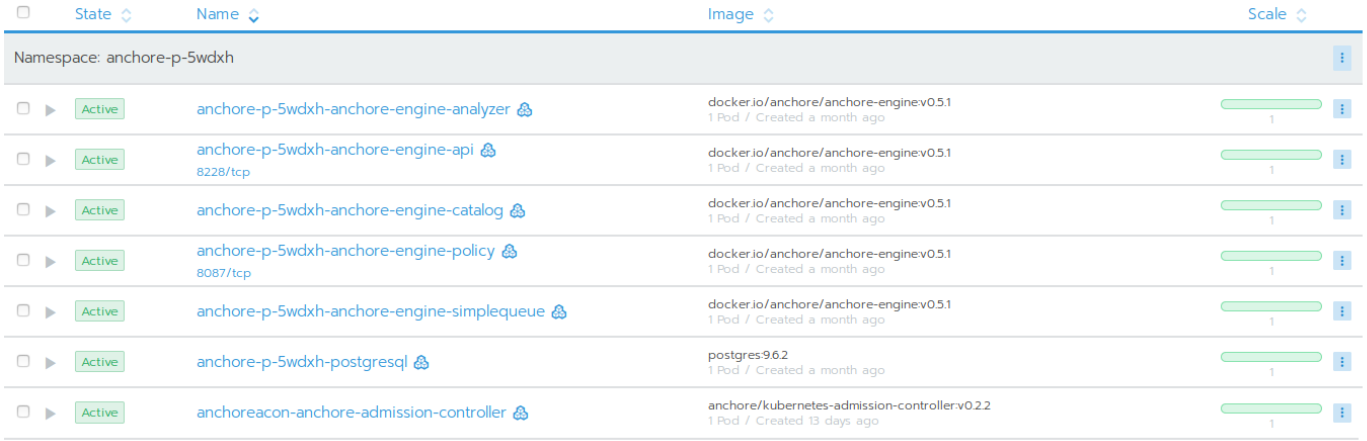


Ilustración 71: Pods de Anchore desplegados en Kubernetes

13.4.2 Publicación de Anchore API Engine

Para poder interactuar con Anchore a través de su cliente, es necesario publicar el servicio que ofrece el *pod* Anchore Engine API a través de su puerto por defecto 8228. Para ello se debe editar la configuración de *pod* Anchore Engine API.

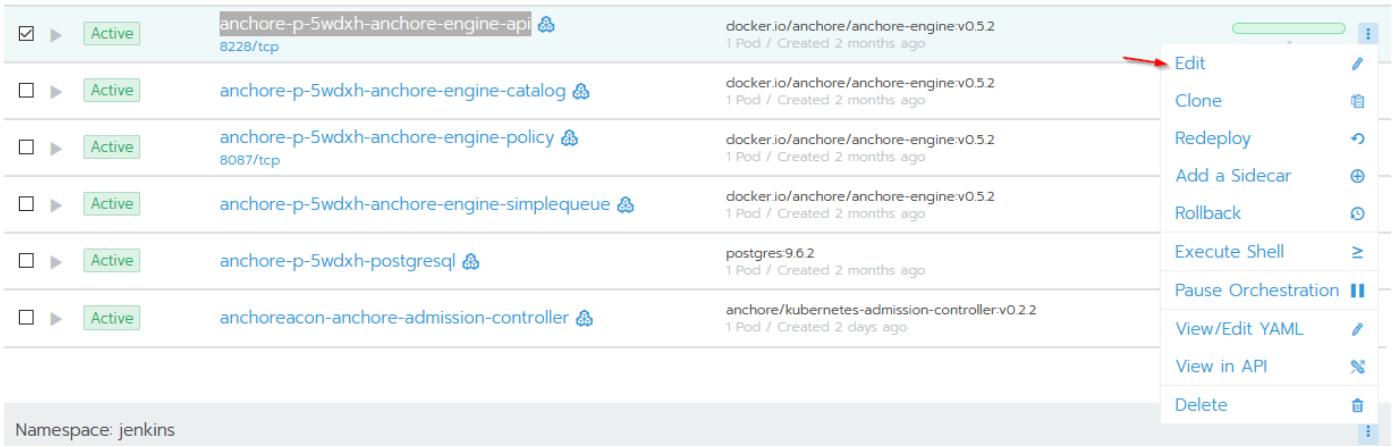


Ilustración 72: Anchore API Engine port mapping

Se procederá a crear el siguiente port mapping.

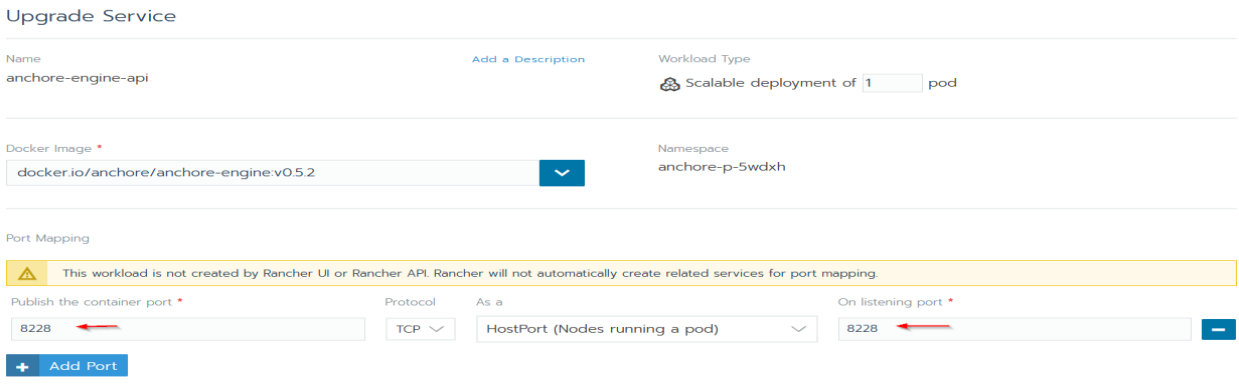
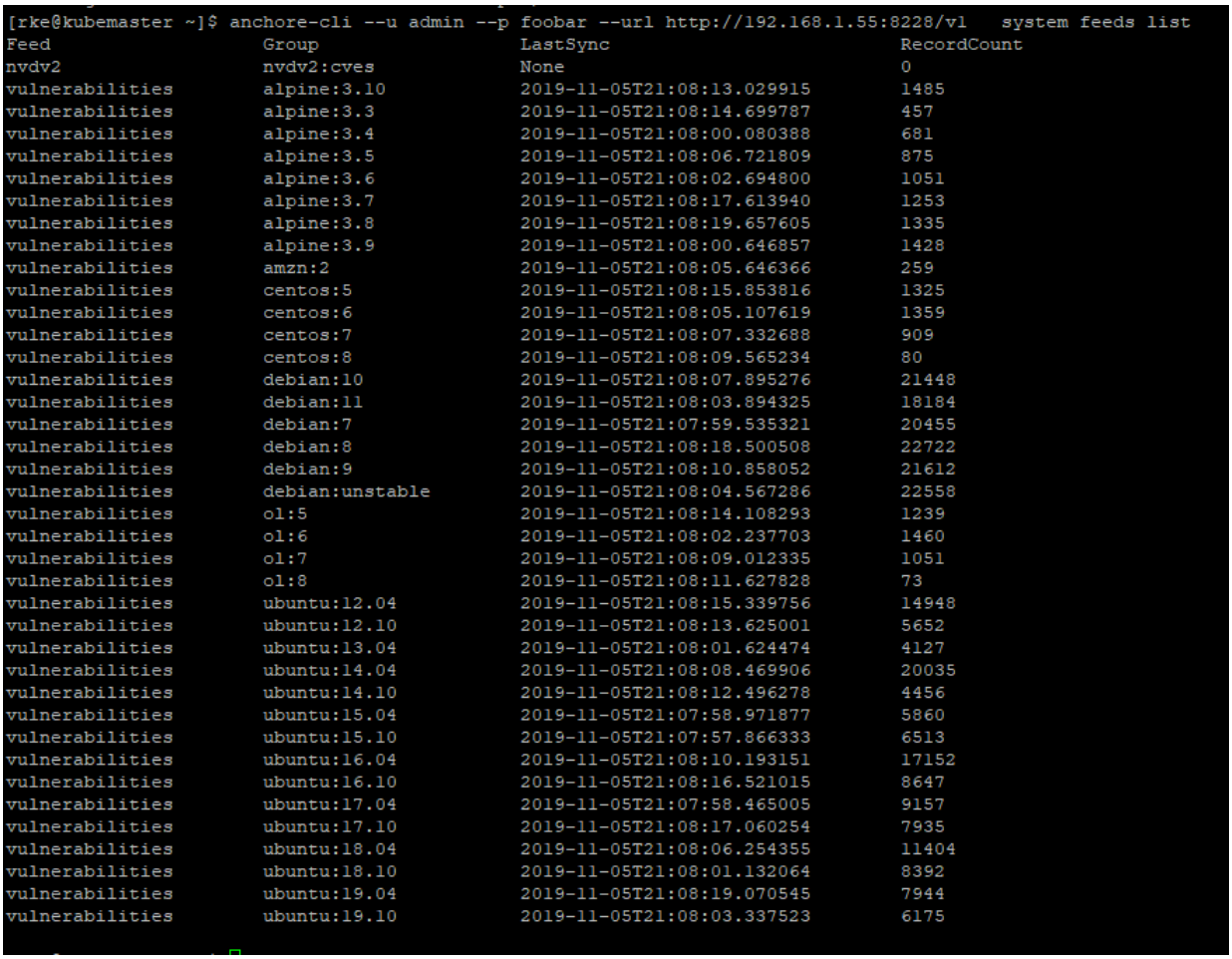


Ilustración 73: Configuración port mapping

13.4.3 Chequeo y actualización de la base de datos de vulnerabilidades

El siguiente comando permite chequear el estado de la base de datos de Anchore, dónde la solución guarda las últimas vulnerabilidades en función del tipo de imágenes que estén en uso en el clúster. Esta DB es la que permite a Anchore realizar la comparación en la fase de análisis entre los paquetes, con sus respectivas versiones, que contiene la imagen de Docker y las vulnerabilidades existentes para dicha versión.

```
# anchore-cli system feeds list
```



```
[rke@kubemaster ~]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   system feeds list
Feed                Group               LastSync                        RecordCount
nvdv2               nvdv2:cves          None                            0
vulnerabilities     alpine:3.10         2019-11-05T21:08:13.029915      1485
vulnerabilities     alpine:3.3          2019-11-05T21:08:14.699787      457
vulnerabilities     alpine:3.4          2019-11-05T21:08:00.080388      681
vulnerabilities     alpine:3.5          2019-11-05T21:08:06.721809      875
vulnerabilities     alpine:3.6          2019-11-05T21:08:02.694800      1051
vulnerabilities     alpine:3.7          2019-11-05T21:08:17.613940      1253
vulnerabilities     alpine:3.8          2019-11-05T21:08:19.657605      1335
vulnerabilities     alpine:3.9          2019-11-05T21:08:00.646857      1428
vulnerabilities     amzn:2              2019-11-05T21:08:05.646366      259
vulnerabilities     centos:5            2019-11-05T21:08:15.853816      1325
vulnerabilities     centos:6            2019-11-05T21:08:05.107619      1359
vulnerabilities     centos:7            2019-11-05T21:08:07.332688      909
vulnerabilities     centos:8            2019-11-05T21:08:09.565234      80
vulnerabilities     debian:10           2019-11-05T21:08:07.895276      21448
vulnerabilities     debian:11           2019-11-05T21:08:03.894325      18184
vulnerabilities     debian:7            2019-11-05T21:07:59.535321      20455
vulnerabilities     debian:8            2019-11-05T21:08:18.500508      22722
vulnerabilities     debian:9            2019-11-05T21:08:10.858052      21612
vulnerabilities     debian:unstable     2019-11-05T21:08:04.567286      22558
vulnerabilities     ol:5                2019-11-05T21:08:14.108293      1239
vulnerabilities     ol:6                2019-11-05T21:08:02.237703      1460
vulnerabilities     ol:7                2019-11-05T21:08:09.012335      1051
vulnerabilities     ol:8                2019-11-05T21:08:11.627828      73
vulnerabilities     ubuntu:12.04        2019-11-05T21:08:15.339756      14948
vulnerabilities     ubuntu:12.10        2019-11-05T21:08:13.625001      5652
vulnerabilities     ubuntu:13.04        2019-11-05T21:08:01.624474      4127
vulnerabilities     ubuntu:14.04        2019-11-05T21:08:08.469906      20035
vulnerabilities     ubuntu:14.10        2019-11-05T21:08:12.496278      4456
vulnerabilities     ubuntu:15.04        2019-11-05T21:07:58.971877      5860
vulnerabilities     ubuntu:15.10        2019-11-05T21:07:57.866333      6513
vulnerabilities     ubuntu:16.04        2019-11-05T21:08:10.193151      17152
vulnerabilities     ubuntu:16.10        2019-11-05T21:08:16.521015      8647
vulnerabilities     ubuntu:17.04        2019-11-05T21:07:58.465005      9157
vulnerabilities     ubuntu:17.10        2019-11-05T21:08:17.060254      7935
vulnerabilities     ubuntu:18.04        2019-11-05T21:08:06.254355      11404
vulnerabilities     ubuntu:18.10        2019-11-05T21:08:01.132064      8392
vulnerabilities     ubuntu:19.04        2019-11-05T21:08:19.070545      7944
vulnerabilities     ubuntu:19.10        2019-11-05T21:08:03.337523      6175
```

Ilustración 74: Comprobación de la base de datos de vulnerabilidades

Mediante el siguiente comando se puede forzar una resincronización de toda la base de vulnerabilidades.

```
# anchore-cli system feeds sync
```

13.4.4 Anchore Admission Controller modificación de comportamiento

En caso de que fuese necesario realizar cambios sobre el fichero *values.yaml*, con la pretensión de cambiar el modo en que actúa el Admission Controller o de cambiar la política de Anchore de la que hará uso, será imprescindible desplegar de nuevo el *pod* con el siguiente comando.

```
helm upgrade anchoreacon anchore/anchore-admission-controller -f values.yaml --force
```

13.4.5 Desactivación del webhook de admisión de Anchore

```
kubectl delete -f validating-webhook.yaml
```

13.4.6 Anchore *Debug*

En esta sección se detallarán varios comandos utilizados para el *debugging* de Anchore y que pueden ser de ayuda al lector en caso de encontrar problemas durante la operación de Anchore.

Verificación del estado general de Anchore:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 system status
```

```
[rke@kubemaster AnchoreAdmissionCotnroller]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 system status
Service policy_engine (anchore-p-5wdxh-anchore-engine-policy-b64864b5c-sw4kw, http://anchore-p-5wdxh-anchore-engine-policy:8087): up
Service analyzer (anchore-p-5wdxh-anchore-engine-analyzer-64846f84d8-l271k, http://anchore-p-5wdxh-anchore-engine-analyzer:8084): up
Service catalog (anchore-p-5wdxh-anchore-engine-catalog-67cf777588-stnfj, http://anchore-p-5wdxh-anchore-engine-catalog:8082): up
Service apiext (anchore-p-5wdxh-anchore-engine-api-684475979c-5cp85, http://anchore-p-5wdxh-anchore-engine-api:8228): up
Service simplequeue (anchore-p-5wdxh-anchore-engine-simplequeue-78dd8d95df-rkbl5, http://anchore-p-5wdxh-anchore-engine-simplequeue:8083): up

Engine DB Version: 0.0.11
Engine Code Version: 0.5.1
```

Ilustración 75: Comprobación de la base de datos de vulnerabilidades

Listado de los últimos eventos:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 event list
```

```
[rke@kubemaster AnchoreAdmissionCotnroller]$ anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1 event list
Timestamp                       Level       Service             Host                                                        Event
   ID
2019-11-25T19:13:27.323218Z     ERROR       catalog             anchore-p-5wdxh-anchore-engine-catalog-67cf777588-stnfj      service_removed
   6966c1064dbd44238d18413ada44a7c2
2019-11-25T19:13:27.280790Z     ERROR       catalog             anchore-p-5wdxh-anchore-engine-catalog-67cf777588-stnfj      service_removed
   ea678970a889484a936ac17a7fc0b8b0
2019-11-25T19:13:27.238336Z     ERROR       catalog             anchore-p-5wdxh-anchore-engine-catalog-67cf777588-stnfj      service_removed
   07a80c377bf34363b2c2246d462492eb
2019-11-25T19:13:27.195525Z     ERROR       catalog             anchore-p-5wdxh-anchore-engine-catalog-67cf777588-stnfj      service_removed
```

Ilustración 76: Listado de los eventos de Anchore

Detalle extendido de un evento en concreto:

```
anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   event get
b74b0f55556b4cf4b3b3bf363a88bf61
```

```
[rke@kubemaster ~]$  anchore-cli --u admin --p foobar --url http://192.168.1.55:8228/v1   event get  b74b0f55556b4cf4b3b3bf363a88bf61
details:
  cause: no heartbeat from service in (300) seconds
  host_id: anchore-p-5wdxh-anchore-engine-policy-7dbcdc8c5b-4kg7n
  service_name: policy_engine
level: ERROR
message: Service down
resource:
  id: http://anchore-p-5wdxh-anchore-engine-policy:8087
  type: service
  user_id: admin
source:
  base_url: http://anchore-p-5wdxh-anchore-engine-catalog:8082
  hostid: anchore-p-5wdxh-anchore-engine-catalog-bf97d487f-q425b
  servicename: catalog
timestamp: '2019-12-06T18:44:04.591486Z'
type: service_down
```

Ilustración 77: Información detallada de un evento de Anchore

Inclusión del parámetro –debug, este parámetro permite una salida detallada de la ejecución de un comando:

```
anchore-cli --debug --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln
docker.io/dockerhubjuliorepository2019/anchoretest:centos6bind os
```

```
[rke@kubemaster AnchoreAdmissionCotnroller]$ anchore-cli --debug --u admin --p foobar --url http://192.168.1.55:8228/v1 image vuln docker.io/dockerhubjulior
epository2019/anchoretest:centos6bind os
INFO:anchorecli.clients.apiexternal:As Account = None
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /v1 HTTP/1.1" 200 5
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /swagger.json HTTP/1.1" 200 167362
INFO:anchorecli.clients.apiexternal:As Account = None
DEBUG:anchorecli.clients.apiexternal:GET url=http://192.168.1.55:8228/v1/images
DEBUG:anchorecli.clients.apiexternal:GET params={'fulltag': u'docker.io/dockerhubjuliorepository2019/anchoretest:centos6bind', 'history': 'false'}
DEBUG:anchorecli.clients.apiexternal:Use get body because detected api version (0, 1, 13) < (0, 1, 6)? False
DEBUG:anchorecli.clients.apiexternal:GET insecure=True
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /v1/images?fulltag=docker.io%2Fdockerhubjuliorepository2019%2Fanchoretest%3Acentos6bind&history=f
alse HTTP/1.1" 200 2074
INFO:anchorecli.clients.apiexternal:As Account = None
DEBUG:anchorecli.clients.apiexternal:GET url=http://192.168.1.55:8228/v1/images/sha256:c0fedb39f69edec4e0be88294a77e40d88c8d1dcdaad1a1640d5b9ab9a389437/vuln
/os?vendor_only=True
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 192.168.1.55:8228
DEBUG:urllib3.connectionpool:http://192.168.1.55:8228 "GET /v1/images/sha256:c0fedb39f69edec4e0be88294a77e40d88c8d1dcdaad1a1640d5b9ab9a389437/vuln/os?vendor
_only=True HTTP/1.1" 200 150
DEBUG:anchorecli.cli.utils:fetched httpcode from response: 200
```

Ilustración 78: Debug en un comando de Anchore

13.5 Instalación y configuración de Jenkins:

13.5.1 Instalación de Jenkins

Para proceder con el despliegue del servidor de Jenkins en este caso se desplegado también dentro del clúster como un *pod* más.

Desde Rancher hipervisor en la categoría de Apps pulsaremos sobre Lunch:

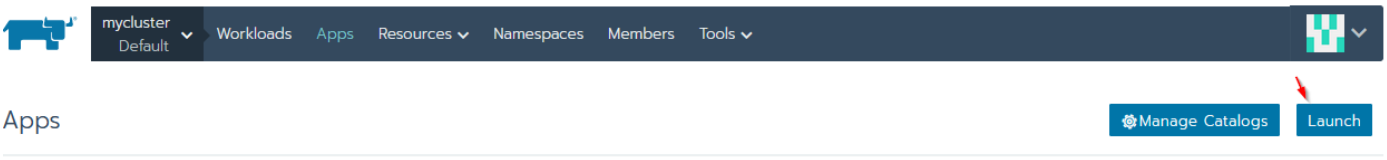


Ilustración 79: despliegue de Jenkins desde Anchore

Filtraremos por Jenkins y pulsaremos en View Details:

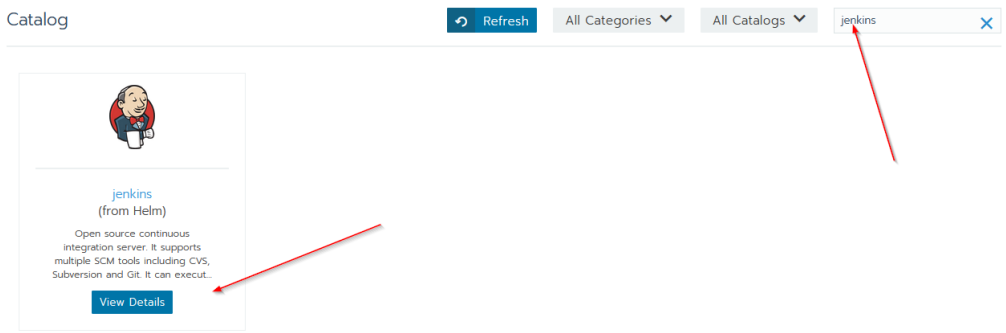


Ilustración 80: Instalación Jenkins

Elegiremos la versión a desplegar y el *namespace* sobre el que se desplegara y pulsaremos lunch:

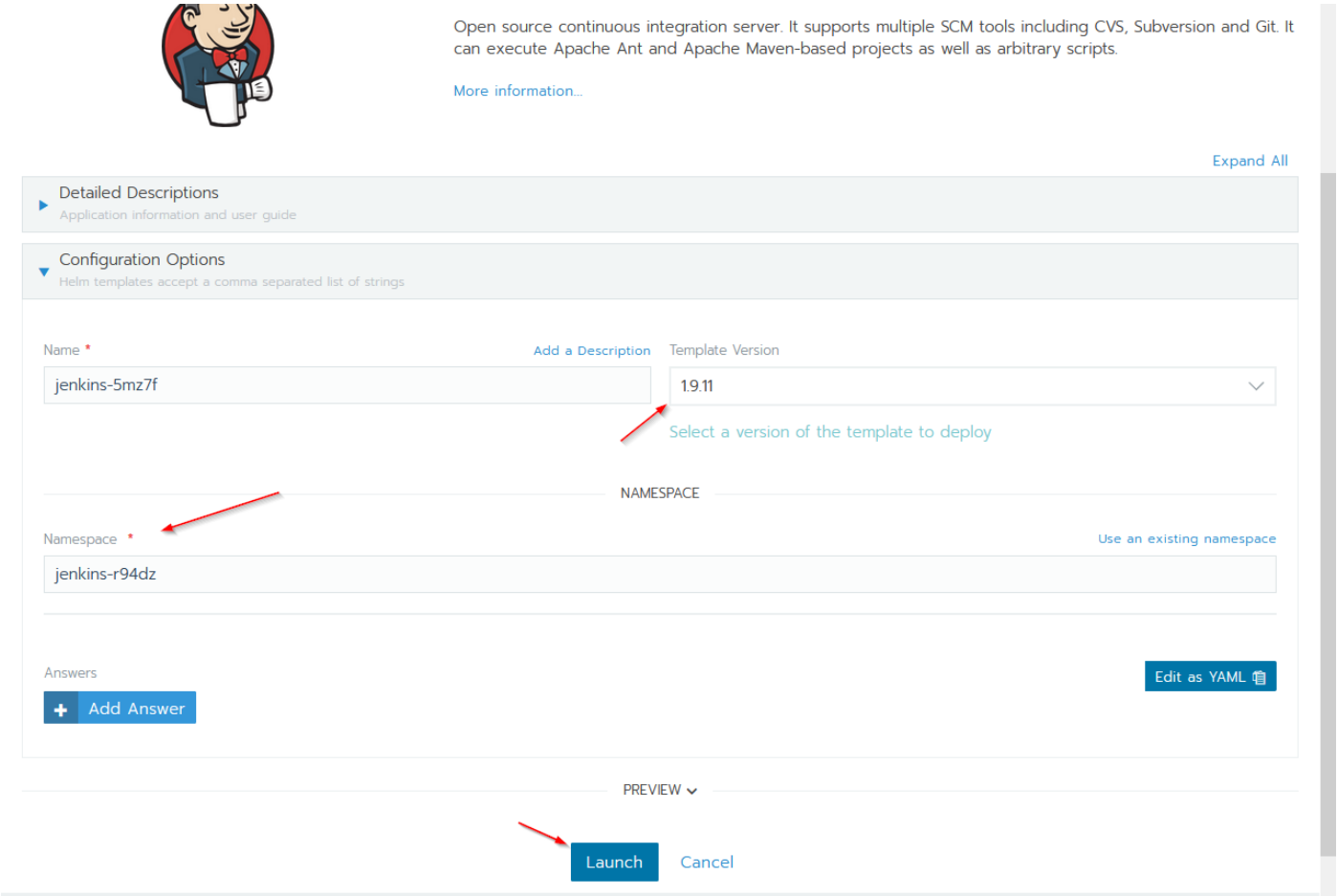


Ilustración 81: campos requeridos para despliegue de jenkins

Una vez finalizado el proceso comprobaremos que los diferentes pods que lo componen esta desplegados, el siguiente pasos era publicar el puerto de acceso a Jenkins:

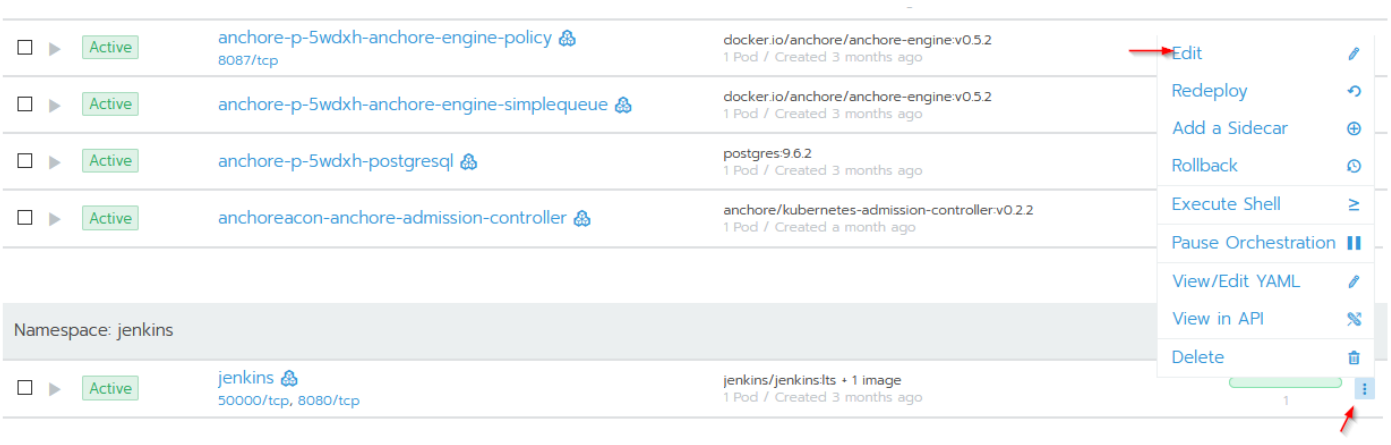


Ilustración 82: *NAT* de los puertos usados por Jenkins

El siguiente pasos es hacer *NAT* de los puertos que expone por defecto Jenkins para que este sea accesible desde el exterior, como se muestra en la captura inferior:

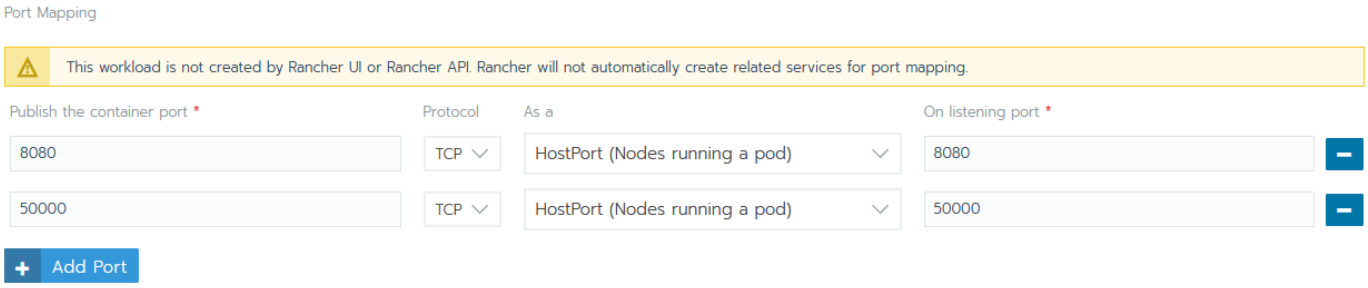


Ilustración 83: *NAT* de los puertos usados por Jenkins II

Una vez publicados deberemos de tener acceso al portal que expone jenkins:

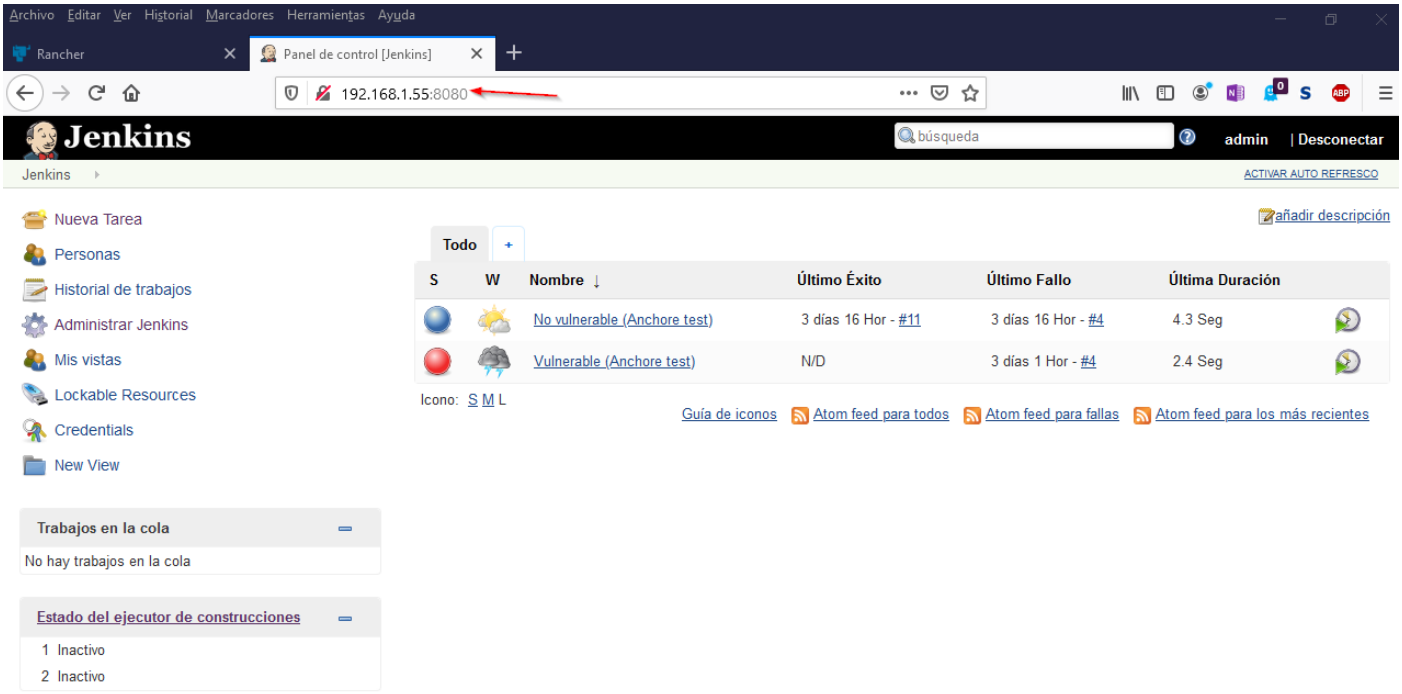


Ilustración 84: Portal de Administración de Jenkins

13.6 Lista simplificada de COTS

- Vmware player
- Máquinas virtuales CentOS 7
- *Hostname*: Kubemaster, kubenode1
- Docker engine 19.03.2
- Kubernetes basado en *Rancher Kubernetes Edition RKE* 1.14.6
- Hipervisor Rancher 2.2.8
- Anchore Engine Analyzer V 0.51
- Anchore Engine API V 0.51
- Anchore Engine Catalog V0.51
- Anchore Engine Policy V 0.51
- Postgres V9.6.2
- Anchore Admission Controller v0.2.2
- Jenkins 1.7.6

14 Bibliografía

Las referencias bibliográficas se han separado en apartados con la intención de mejorar la categorización de la documentación utilizada a lo largo del desarrollo del proyecto. En el apartado documentos de aplicación, se categorizan aquellas referencias documentales necesarias para el desarrollo o cumplimiento de algunas de las secciones del proyecto. En la sección documentos de referencia se incluyen todos los documentos que han servido como base documental al proyecto y que además servirán como fuentes para la ampliación del conocimiento.

14.1.1 Documentos de Aplicación

La siguiente tabla alberga todos los documentos que son de aplicación para el desarrollo del proyecto.

Ref.	Título	Localización
AD-01	Anchore configuración de Registros	*URL*
AD-02	Fichero JSON, Anchore_CIS_1.13.0_MOD.json que contiene la política aplicada en el proyecto.	*URL*
AD-03	Anchore *policy*	*URL*
AD-04	Admission Control in Kubernetes with Anchore	*URL*
AD-05	Kubernetes-admission-controller (Git Hub)	*URL*
AD-06	RHSA-2019:1492 - Security Advisory	*URL*
AD-07	CVE-2018-5743	*URL*
AD-08	Instalación Docker Engine Basado en CentOS	*URL*
AD-09	Script instalación *RKE*	*URL*
AD-10	Fichero ejemplo, despliegue clúster Kubernetes basado en *RKE*	*URL*

Tabla 9: Documentos de Aplicación.

14.1.2 Documentos de Referencia

La siguiente tabla alberga el índice de todas las referencias existentes en el documento:

Ref.	Titulo	Localización
RD-01	Def. Línea Base	*URL*
RD-02	Def. Clúster	*URL*
RD-03	Def. Contenedor Docker	*URL*
RD-04	CPU	*URL*
RD-05	Def. *Hardening*	*URL*
RD-06	Def. Hipervisor	*URL*
RD-07	Def. Integración Continua (CI)	*URL*
RD-08	Def *pod*	*URL*
RD-09	Def. Rollback	*URL*
RD-10	Def. *Snapshoot*	*URL*
RD-11	Conceptos Docker	*URL*
RD-12	Conceptos Kubernetes Componentes de un clúster de Kubernetes	*URL* *URL*
RD-13	Conceptos Rancher Kubernetes Edition	*URL*
RD-14	Arquitectura Rancher	*URL*
RD-15	Conceptos Anchore Página web Anchore Anchore.io Conceptos de despliegue de Anchore Deployment	*URL* *URL* *URL*
RD-16	Docker Hub	*URL*
RD-17	Fran Ramirez, Elias Grande Rafael Troncoso, Docker: SecDevOps Editorial 0xWord, Móstoles (2019) ISBN: 978-84-697-9752-5	*URL*
RD-18	CIS_Docker_1.13.0_Benchmark_v1.0.0.pdf Documento de referencia sobre *hardening* de imágenes de Docker	*URL*
RD-19	CIS_Kubernetes_Benchmark_v1.4.1 Documentod e Referencia para el *hardening* de clústeres de Kubernetes	*URL*
RD-20	Jon-Anders Kabbe, Security analysis of Docker containers in a production enviroment June 2017 NTNU	*URL*
RD-21	Adrian Mouat, Docker Security, editorial O'Reilly Media, Inc ISBN 978-1-491-93661-0	*URL*

Tabla 10: Documentos Referenciados