Aceptación

ARQUITECTURA DE ALTA DISPONIBILIDAD DE CONTENEDORES DOCKER PARA ALOJAR LAS APLICACIONES DE LA PONTIFICIA UNIVERSIDAD JAVERIANA SECCIONAL CALI

JHON HENRY GÓMEZ TABARES

Nota de Aceptación

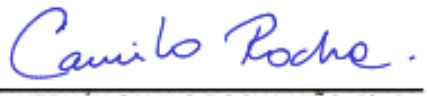
Certificamos que el presente Trabajo de Grado satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

Juan Carlos Martínez Arias
Director

Carlos Alberto Llano Rodríguez
Jurado

Yury Niño Roa
Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en Ingeniería de Software.



HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Santiago de Cali, diciembre 15 de 2021

Acta de correcciones

Maestría en Ingeniería de Software
Facultad de Ingeniería y Ciencias

Acta de Correcciones al Documento de Trabajo de Grado

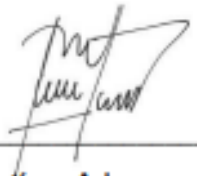
Santiago de Cali, 15 de diciembre de 2021

Autor: JHON HENRY GOMEZ TABARES

Título del Trabajo de Grado: "ARQUITECTURA DE ALTA DISPONIBILIDAD DE CONTENEDORES DOCKER PARA ALOJAR LAS APLICACIONES DE LA PONTIFICIA UNIVERSIDAD JAVERIANA SECCIONAL CALI"

Director:

Como indica el artículo 2. 13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.



Juan Carlos Martínez Arias
Director del Trabajo de Grado

DATOS PERSONALES DEL ESTUDIANTE

1. NOMBRE: Jhon Henry Gómez Tabares
2. DIRECCIÓN: Calle 16ª #49ª-46
3. TELÉFONO: 314 7018180
4. CORREO ELECTRÓNICO: jhonhenry.gomez@javerianacali.edu.co
5. UNIVERSIDAD: Universidad Autónoma de Occidente
6. PROFESIÓN: Ing. En Informática
7. EMPRESA: Rappi
8. CARGO: Senior DBA Engineer - DevOps

ARQUITECTURA DE ALTA DISPONIBILIDAD DE CONTENEDORES DOCKER PARA ALOJAR LAS APLICACIONES DE LA PONTIFICIA UNIVERSIDAD JAVERIANA SECCIONAL CALI

Jhon Henry
Código 20131501

Trabajo de grado para optar al título de
Magister en ingeniería de Software

Director
Juan Carlos Martínez Arias

Codirector
Gilberto Villa Soto

FACULTAD DE INGENIERÍA Y CIENCIAS
MAESTRÍA EN INGENIERIA DE SOFTWARE
SANTIAGO DE CALI, DICIEMBRE DE 2021

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

GLOSARIO

- TIC
  Las siglas como tal significan tecnologías de la información y la comunicación, que combinan temas de informática, electrónica y las telecomunicaciones, con el fin de generar una información y comunicación más eficiente [1].
  El concepto se aplica en todo el documento actual, dado a la influencia a nivel de tecnología de los diferentes temas mencionados.

- Arquitectura de software
  La Arquitectura de Software se refiere a "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos [2].

- IOPS
  Son las siglas de Inputs Outputs Per Second (Entradas Salidas Por Segundo). Es un método utilizado para medir el rendimiento de los discos duros, como SATA, SSD, Nearline, entre otros. Concepto muy importante para análisis, monitoreos y pruebas de infraestructuras, previo, durante y después de una implementación en producción de determinado sistema o arquitectura.

- Patrón arquitectónico
  Un patrón arquitectónico es una solución general y reutilizable a un problema común en la arquitectura de software dentro de un contexto dado. Los patrones arquitectónicos son similares al patrón de diseño de software, pero tienen un alcance más amplio [3].

- DevOps
  El término DevOps, que es una combinación de los términos ingleses development (desarrollo) y operations (operaciones), designa la unión de personas, procesos y tecnología para ofrecer valor a los clientes de forma constante [4].

RESUMEN

La evolución de la arquitectura de TI ha generado cambios importantes en el ámbito de la información; es así como los servicios de TI actuales se prestan basados en infraestructuras de servidores virtualizados a partir de los cuales se generan ahorros en diferentes aspectos. El propósito de este proyecto fue plantear una solución de alta disponibilidad en contenedores Docker para la Pontificia Universidad Javeriana Cali, que permita alojar las diferentes aplicaciones con las que cuenta, disminuir consumo de recursos, estandarizar servidores, entre otras soluciones. Se plantea el uso de diferentes tecnologías que serán evaluadas a lo largo del documento, así como un trabajo detallado en verificación del estado actual y alternativas, para una posterior etapa detallada de diseño, otra de validación mediante un prototipo y una última de evaluación de resultados y desempeño.

ABSTRACT

The evolution of the TI architecture has generated critical changes in the information field; this is how nowadays TI services are provided based on virtualized server infrastructures, from which big savings (from different points of view) are made. The purpose of this project is to propose a high availability solution in Docker containers for the Pontificia Universidad Javeriana Cali, that allows the hosting of the many applications that there are, that reduces the resource consumption, that standardizes servers, among other solutions. The proposal considers the use of different technologies that will be evaluated throughout the document, as well as a detailed work of verification of the current state and the alternatives, in order to reach a later detailed stage of design, other stage of validation by a prototype, and a last stage of results and performance evaluation

INTRODUCCIÓN

Usualmente los arquitectos de TI aprenden a desarrollar soluciones para su entorno de tecnología, basados en la teoría y la experiencia adquirida con los problemas sobre diseño y tecnología. Diseñar una arquitectura de TI no es una tarea fácil, es necesario validar decisiones importantes de este nivel con literatura o con juicio de expertos, lo cual brinde una sensación de alivio al conocer que no es la única persona que se enfrenta a determinado desafío arquitectónico [5].

Un arquitecto de TI se puede encontrar con decisiones arquitecturales trascendentales para el cumplimiento de los objetivos; para este caso, una decisión a la que se han enfrentado muchos arquitectos en los últimos años gira alrededor del uso de virtualización tradicional o de contenedores, siendo el último una de las tecnologías más utilizadas actualmente. Empresas como Google ejecutan la mayoría de sus servicios principales (Gmail, YouTube, Búsqueda, entre otros), bajo arquitecturas basadas en contenedores, justificando que su uso ha permitido que su equipo de desarrollo se mueva rápidamente, implementar software de manera eficaz y operar a una escala sin precedentes; adicionalmente, Google destaca algunas diferencias entre contenedores y la virtualización tradicional, como por ejemplo el tamaño en disco, la baja sobrecarga, el aislamiento, el entorno uniforme que ofrece un contenedor, entre otros [6]. Es tarea del arquitecto de TI revisar lo que actualmente hacen las grandes empresas, conocer sus experiencias, sus casos de éxito y todo el camino tecnológico que han recorrido, lo que permite añadir justificaciones a las diferentes decisiones arquitecturales que se puedan tomar.

La tecnología de contenedores se expande cada vez más, lo que está impulsando el mercado de herramientas de orquestación, que en los próximos años podría crecer a una tasa interanual compuesta del 17,2% [7]. Adicionalmente, se genera la necesidad de contar con esquemas de alta disponibilidad en los diferentes sistemas de información, que complementen las migraciones a contenedores. Usualmente es necesario buscar una herramienta que permita orquestar la cantidad de contenedores que se van a implementar o que han implementado, toda vez que su administración manual será una tarea operativa bastante costosa que es posible automatizar; adicionalmente, se van a ir generando necesidades que un orquestador puede solventar, tales como: monitoreo de contenedores, balanceo de carga, despliegue automático, entre otros. Actualmente, existen muchas herramientas de orquestación, en las que se destacan algunas como: Kubernetes, Docker Swarm, Azure Container Service (AKS), Amazon EX2 Container Service (ECS), entre otros. También es posible utilizar gestores de entornos de contenedores un poco más sencillos, especialmente cuando se inicia en arquitecturas con contenedores; para estos casos se pueden utilizar herramientas como Portainer, que permiten gestionar un entorno de contenedores Docker casi de manera integral.

Actualmente, la Pontificia Universidad Javeriana Cali aloja sus aplicaciones web bajo un entorno virtualizado, generando esto algunas novedades con respecto al consumo de recursos, homogeneidad de servidores, entre otros aspectos que serán detallados a lo largo del documento.

Por lo anterior, se planteó una solución de alta disponibilidad en contenedores Docker para la Pontificia Universidad Javeriana Cali, que permita alojar las diferentes aplicaciones con las que cuenta.

1. DEFINICIÓN DEL PROBLEMA

1.1 Planteamiento del problema

De acuerdo con el informe de tendencias de InfoQ del año 2019, el uso de contenedores y de kubernetes para temas de orquestación y gestión, ha acaparado el mercado de manera significativa [8].

Muchas compañías han decidido migrar sus servicios o aplicaciones a arquitecturas que incluyan contenedores, con el fin de aprovechar las ventajas que ofrece en temas de compatibilidad, mantenibilidad, estandarización, consistencia entre ambientes, ligereza vs una virtualización tradicional, entre otros.

Actualmente, el Centro de Servicios Informáticos de la Pontificia Universidad Javeriana Cali, maneja la mayoría de sus aplicaciones web en el contenedor de servlets Apache Tomcat, las cuáles se encuentran distribuidos en diferentes servidores virtuales (en su mayoría Windows Server), bajo una virtualización tradicional.

Al utilizar una virtualización tradicional, se generan algunas dificultades, tales como:

- Se presenta actualmente un consumo de recursos alto (CPU, RAM, disco duro, entre otros) en todos los servidores de aplicación utilizados para producción, réplica, test, y desarrollo. Actualmente se cuenta con alrededor de 20 servidores que contienen el servicio de Tomcat (entre 4 y 5 instancias por servidor), los cuales alojan las diferentes aplicaciones. Cada servidor mencionado por lo general tiene asignado entre 10gb y 12gb de memoria RAM, con una CPU entre 4 y 8 core, lo cual aporta un consumo importante en los diferentes servidores físicos a los que pertenecen los servidores virtuales. De acuerdo con las estadísticas que genera semanalmente la coordinación de infraestructura del Centro de Servicios Informáticos, los servidores Tomcat normalmente cuentan con un consumo de memoria RAM y recursos superior al 70% en promedio, durante diferentes horas del día.
- Aunque bajo la actual arquitectura es posible crear alta disponibilidad de servicios, se suele realizar para aplicaciones core independientes; sin embargo, otros servicios aún no cuentan con una arquitectura que permita (entre otras cosas), balancear carga o permitir el cambio de aplicaciones en horario productivo sin afectar la operación (tipo técnicas bluegreen deployment o canary release).
  Los administradores de sistemas de la Universidad han contemplado en muchas ocasiones en utilizar la arquitectura mencionada para todos los servidores de aplicación o para

contenedores servlets Apache Tomcat, pero necesariamente multiplicaría la cantidad actual de recursos de una manera considerable

- Como se mencionó anteriormente, se cuenta también con ambientes de réplica, test y desarrollo, para las diferentes etapas de los proyectos. Conservar los servidores de pruebas exactamente iguales a nivel de configuraciones generales y parametrizaciones, no es tarea fácil bajo la virtualización tradicional; aunque actualmente el equipo de infraestructura aplica diferentes estrategias manuales (actualizaciones de S.O programadas, máquina virtual única utilizada como plantilla, etc.), se presentan diferentes novedades en algunas ocasiones cuando se pasan aplicaciones entre ambientes, debido a pequeñas diferencias que pueden generar los sistemas operativos (por ejemplo); adicionalmente, el tiempo de alistamiento y configuración de un nuevo ambiente, genera una carga operativa alta para el equipo de infraestructura.

De acuerdo con las dificultades planteadas anteriormente, se determinó como una solución la definición de una arquitectura de contenedores Docker, que permita la configuración e instalación de los servicios Tomcat y a su vez alojen las diferentes aplicaciones creadas en Java. Al determinar una arquitectura de contenedores Docker, se configuró también una herramienta que permite realizar la administración de los diferentes componentes.

1.2 Formulación del problema

Se propuso el siguiente interrogante:

¿Cómo plantear una arquitectura de alta disponibilidad que permita almacenar las aplicaciones web, disminuya el consumo de recursos y permita tener una homogeneidad entre los diferentes ambientes de desarrollo de software en la Pontificia Universidad Javeriana Cali?

Subpreguntas:

- ¿Cuál es la mejor práctica para aplicar una arquitectura basada en contenedores?
- ¿Cómo plantear una arquitectura con servicios en alta disponibilidad, en contenedores Docker?
- ¿Cómo realizar la validación de un diseño de arquitectura de TI?

2. OBJETIVOS DEL PROYECTO

2.1 Objetivo General

Diseñar una arquitectura de alta disponibilidad implementada en contenedores Docker, que permita almacenar las aplicaciones web de la Pontificia Universidad Javeriana Cali.

2.2 Objetivos Específicos

- Identificar alternativas, para plantear una arquitectura de acuerdo con las necesidades relacionadas con contenedores y de alta disponibilidad de la Pontificia Universidad Javeriana Cali.
- Diseñar una arquitectura en contenedores Docker de alta disponibilidad.
- Validar la arquitectura planteada por medio de un prototipo.
- Comparar el desempeño de la arquitectura prototipo en contenedores Docker vs la arquitectura con virtualización tradicional, mediante pruebas no funcionales de rendimiento.

2.3 Resultados Esperados

A continuación, se relacionan los resultados esperados, basados en los objetivos específicos:

1. Informe detallado de la arquitectura de los servidores de aplicaciones que actualmente tiene la universidad, rendimiento de servidores, estadísticas de uso, entre otros
    a. Identificar y priorizar requisitos basados en la actual arquitectura.
2. Diseño de la arquitectura
    a. Entrega de los diferentes atributos y escenarios de calidad, diagramas del modelo c4 y documentaciones referentes a los patrones de arquitectura y las decisiones arquitecturales tomadas.
3. Entrega de una arquitectura prototipo para realizar la validación del diseño planteado
    a. Entrega de una infraestructura donde se alojará la aplicación prototipo que se probará, realizando una configuración completa de la infraestructura demo y manuales de configuración del prototipo realizado.
4. Informe sobre el análisis del desempeño de la arquitectura virtualizada vs el prototipo creado en contenedores Docker

3. MARCO TEÓRICO Y ANTECEDENTES

3.1 Bases teóricas

3.1.1 Alta disponibilidad

La alta disponibilidad es una técnica, estrategia o conjunto de procesos, que permite que determinado sistema conserve su funcionamiento ante eventuales fallos.

Los sistemas de alta disponibilidad tratan de brindar beneficios alrededor de la tolerancia a fallos, basados en sistemas que cuenten con replicación de elementos (servidores, sistemas de información, entre otros), lo que vendría siendo un clúster de alta disponibilidad [9].

Actualmente las compañías utilizan este tipo de arquitecturas con el fin de satisfacer diferentes atributos de calidad, como lo son la disponibilidad, seguridad, fiabilidad, entre otros; y es que este concepto es posible aplicarlo en empresas con diferentes tipos y tamaños de infraestructuras, pero es necesario conocer y contemplar cada ítem que compone su arquitectura, debido a que aplicar una alta disponibilidad incluye considerar detalles específicos que son diferentes en las empresas.

Por ejemplo, lo que realiza una empresa para contar con una alta disponibilidad con 300 servidores en un solo centro de datos, con aplicaciones educativas vs una compañía global, con múltiples centros de datos alrededor del mundo, con servicios de streaming y con más de 125 millones de usuarios, es totalmente diferente debido a los escenarios y circunstancias de cada compañía. Y sí, el ejemplo se refiere a Netflix. Los artículos del blog de Netflix son interesantes para conocer el trabajo que hacen las grandes compañías y revisar las diferentes situaciones que experimentan cuando quieren aplicar alguna metodología o tecnología, es algo que apoya las bases teóricas para la toma de decisiones que se realicen en el proyecto; y ese es el caso de Netflix, ellos indican que su crecimiento de suscriptores fue muy rápido y tuvieron que enfrentarse a múltiples desafíos. Muchas cosas salieron mal, teniendo hasta incidentes de cara al cliente. Aun así, en muchos de sus artículos indican que aún siguen descubriendo muchas cosas, y es que en sus diferentes blogs siempre dejan información referente a experiencias vividas a lo largo de los procesos que afrontan [10].

3.1.2 Blue-Green deployment

Conocida como implementación A/B, es una técnica muy popular para el lanzamiento de aplicaciones, que permita pasar el tráfico de forma masiva (una vez terminadas las pruebas) de un servidor azul (antiguo entorno) a verde (nuevo entorno). Es una técnica muy usada para mitigar los riesgos que se pueden identificar al realizar determinado paso a producción de una aplicación, ya que, al migrar los usuarios al entorno verde, el azul sirve como medida preventiva ante posibles fallos, siendo (en caso de fallos) solamente necesario enrutar el tráfico de nuevo al azul; para llevar a cabo lo anterior, es necesario tener dos ambientes productivos idénticos (azul y verde).

Es necesario revisar el entorno donde se quiere aplicar este tipo de técnicas, no siempre todos se encuentran preparados o desarrollados para soportar este tipo de actividades [11].

Se puede decir de que este tipo de implementación es posible incluirla en un paquete de técnicas o actividades de un plan completo de alta disponibilidad, toda vez que se analiza desde el punto de vista del software como tal, el cual como se explicaba anteriormente, se va pasando a producción de forma progresiva.

Todos estos conceptos son importantes tenerlos en cuenta, con el fin de formar una estrategia de alta disponibilidad integral en los aspectos que se encuentren en el alcance.

3.1.3 Canary Release

Similar al concepto de Blue-Green Deployment, su diferencia se basa puntualmente en que la enrutada de los usuarios de una versión A a una B, se hace por fases o por determinado bloque de usuarios, quedando la versión anterior como una versión de respaldo y aun respondiendo hasta que se finalice la migración.

El lanzamiento de Canary es un medio para verificar los aspectos de calidad de las nuevas versiones de software en un entorno de producción, manteniendo el riesgo al mínimo [12].
Esta técnica es utilizada para reducir el riesgo al pasar a producción determinado software.

Por ejemplo, Netflix utiliza tres clústers que atienden todo el tráfico, pero subdivido entre las tres partes. Tienen uno de producción, otro de línea base y otro llamado Canary. El primero contiene toda la información productiva y recibe la mayor cantidad de usuarios, el segundo es utilizado como referencia (se compara con el Canary) y el tercero contiene los cambios que se van a implementar a nivel de software [13].

Si bien es una técnica utilizada especialmente para software, se considera también un punto para tener en cuenta en esa formación de una arquitectura integral para una compañía.

3.1.4 Clúster

Es un conjunto de computadoras construidas mediante la utilización de componentes de hardware que se comportan como si fuesen una única computadora [14].

Aunque el concepto como tal de clúster se puede generalizar en conceptos de agrupación de determinadas instancias o granjas de servicios, se identifica como una importante base teórica por el hecho de que es parte fundamental del planteamiento de una arquitectura de alta disponibilidad para el proyecto.

Y es que se considera parte fundamental debido a casos como los de IBM, que indica que toda implementación de su marca en alta disponibilidad requiere configurar un clúster para controlar y gestionar los recursos resilientes [15].

3.1.5 Contenedores

Los contenedores son un ejemplo de un enfoque para implementar la virtualización a nivel de sistema operativo. Son entornos de ejecución autónomos que pueden tener su propia CPU, memoria, entrada / salida (E/S) y recursos de red aislados, y pueden compartir un núcleo de un sistema operativo host.

Los contenedores se pueden aislar unos de otros y de sus servidores (físicos y / o virtuales). Por ejemplo, pueden tener sus propios sistemas de archivos. Es posible que no tengan visibilidad de los procesos de los demás. Sus recursos informáticos (por ejemplo, procesamiento, almacenamiento, redes, etc.) pueden estar limitados. Los contenedores pueden ser más fáciles de construir y configurar que las máquinas virtuales, y debido a que los contenedores se pueden desacoplar de su infraestructura subyacente y de los sistemas de archivos del host, pueden ser altamente portables en varias nubes y distribuciones de sistemas operativos[16].

Se mencionan aspectos en el concepto asociados a servidores físicos y virtuales, toda vez que el proyecto se ejecutó en una infraestructura de un Centro de Datos local, que cuenta con múltiples hosts físicos y servidores virtuales.

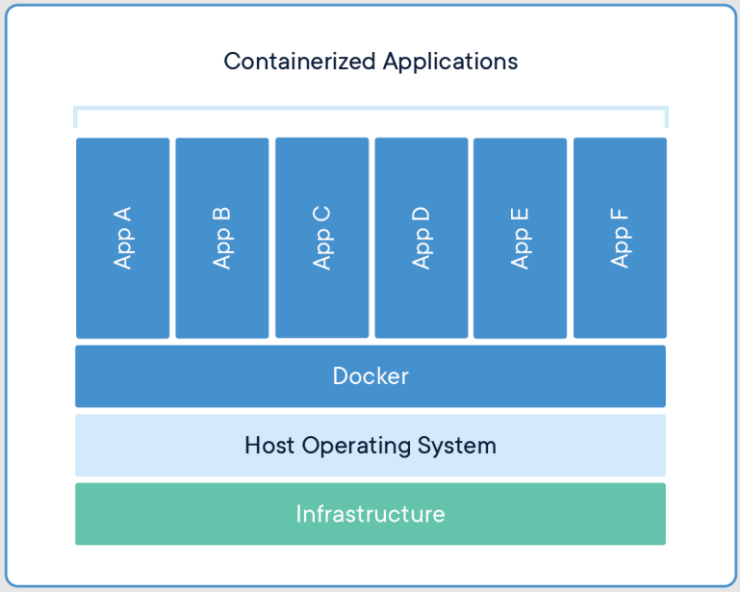


Figura 1. Aplicaciones en contenedores [17]

3.1.6 Docker

La definición del concepto indica que es una tecnología de creación de contenedores, que permite crear y usar contenedores de Linux[18].

Docker trata de resolver el problema de la dependencia en las arquitecturas; las modernas suelen ensamblarse a partir de componentes existentes y dependen de diferentes servicios y aplicaciones.

Cada componente que tenga un sistema de información viene con un conjunto determinado de dependencias que podrían generar conflictos con las de otros componentes. Al empaquetar los diferentes componentes y sus dependencias, Docker resuelve dicha novedad[19].

Para el proyecto, el término Docker es fundamental, dado a que es una tecnología que se utilizó de forma directa, teniendo en cuenta los objetivos de arquitectura planteados. Adicionalmente, se utiliza Docker Desktop para el servidor Windows server donde se configura todo lo referente al demo.

3.1.7 Máquinas virtuales

Las máquinas virtuales son una abstracción de hardware físico que convierte un servidor en muchos servidores [17].
El sistema de software se llama hipervisor, el cual se encarga de separar los recursos de la máquina del sistema de hardware e implementarlos adecuadamente para que las diferentes máquinas virtuales puedan utilizarlos [20].

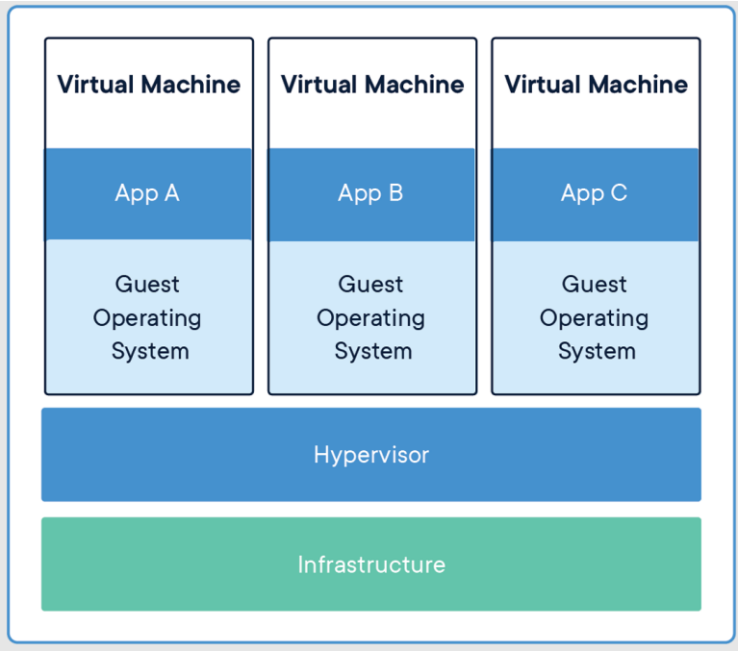


Figura 2. Arquitectura de máquinas virtuales [17]

3.1.8 Contenedores vs máquinas virtuales

Los contenedores y las máquinas virtuales tienen beneficios similares en los aspectos de asignación y aislamiento de recursos, pero funcionan de manera diferente debido a que los contenedores virtualizan el sistema operativo en lugar del hardware [17].

Son entornos informáticos empaquetados que combinan varios elementos de TI y los aíslan del resto del sistema. Las principales diferencias radican en la capacidad de ampliación y la portabilidad de los diferentes elementos que los conforman [21].

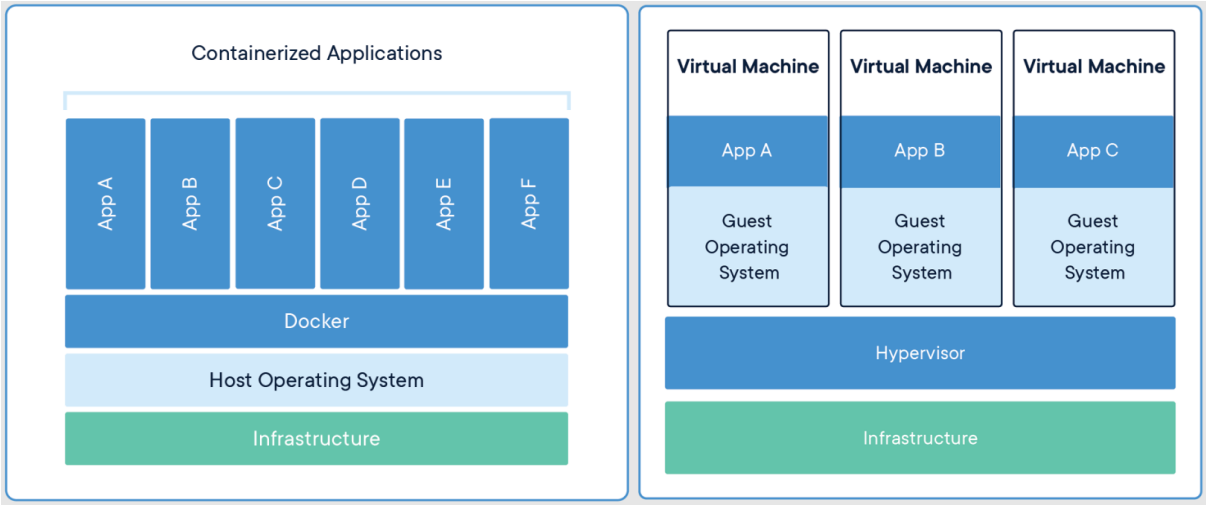


Figura 3. Contenedores vs Máquinas virtuales [17]

3.2 Antecedentes (trabajos previos)

A continuación, se presentan los antecedentes que se consideran más significativos a nivel de aporte en diferentes aspectos

- A Performance Evaluation of Containers running on Managed Kubernetes Services [22]
  El documento describe y brinda información acerca del uso de contenedores y la orquestación con Kubernetes, continuando con un análisis del comportamiento de los contenedores cuando son orquestados por los mismos. En el artículo describen varios puntos que se consideran necesarios resaltar:
  - Es valiosa la información que brinda alrededor de la forma como ha ido ganando terrero el uso de contenedores, y aunque todo lo basa en un ambiente en la nube, los conceptos a nivel general son de mucho aporte para el proyecto
  - La explicación técnica que brinda del funcionamiento de contenedores, la justificación de su uso y algunos detalles, son de gran utilidad para el diseño y la ejecución del prototipo del proyecto
  - Justifica el por qué es necesario utilizar un orquestador o gestor de componentes; esa información y experiencia es importante, y tiene una relación directa con el proyecto
  - El documento se centra en evaluar las características de rendimiento de los contenedores orquestados por kubernetes y muestra información técnica alrededor de la forma como se pueden realizar dichos monitoreos y/o evaluaciones; esta información es importante para futuras implementaciones de orquestadores robustos, en caso de considerase necesario

- La información que brinda sobre antecedentes, citando algunos artículos que contienen datos directamente relacionados con el proyecto, tales como los diferentes escenarios de evaluación que han realizado en otros documentos, comparando por ejemplo máquinas virtuales vs contenedores Docker

La relación del artículo con el proyecto planteado es amplia, ya que en el documento actual se habla del uso de contenedores y del uso de un gestor u orquestador (se plantea Portainer); es por ello que se ha considerado de gran aporte toda la información que brinda el artículo, con aspectos destacados como las métricas que plantean alrededor del tema del funcionamiento de los contenedores a nivel de rendimiento, cuando son orquestados por Kubernetes; adicionalmente, conocer la experiencia y las diferentes justificaciones que brindan ante el aumento del uso de dichas tecnologías, aportó de forma directa a los diferentes objetivo planteados en el documento.

- Design and Implementation of an Edge Computing Platform Architecture Using Docker and Kubernetes for Machine Learning [23]
  En este artículo justifican el uso de contenedores Docker orquestados por kubernetes, pero para utilizar y configurar toda una serie de servicios de Machine Learning. La relación del artículo con el proyecto planteado se basa en las justificaciones del uso de contenedores Docker y de su orquestador, y el aporte directo que realiza es con respecto a la forma como analiza el consumo de recursos teniendo en cuenta el tipo de servicios que va a alojar. Observar esa última parte fue muy importante para plantear en el proyecto temas de justificación del uso de una herramienta de gestión u orquestación, las diferentes ventajas que brinda en lo planteado actualmente y las opciones de otras herramientas que se pueden utilizar ante un futuro crecimiento de la infraestructura de contenedores (Kubernetes, Rancher, etc).

- Docker Containers Across Multiple Clouds and Data Centers [24]
  El documento cuenta con una relación bastante amplia con el proyecto desarrollado; mencionan algo muy importante y es que para entornos empresariales donde se deseen implementar arquitecturas de contenedores, es necesario tener en cuenta algunas consideraciones como la alta disponibilidad, que a su vez trae diferentes desafíos alrededor de la coordinación, programación y adaptación.

  A diferencia de lo que se plantea en este documento, en el artículo proponen un prototipo o un marco de trabajo llamado C-ports, que permitirá la implementación y gestión de los contenedores Docker en múltiples nubes hibridas; aunque es diferente al proyecto, leer

este tipo de secciones permitió asemejarlo con el uso que se les dio a los diferentes componentes de la arquitectura que se plantea. Ese punto y esa experiencia de tener en cuenta esos desafíos, fue muy importante para el proyecto, ya que permite previsualizar diferentes novedades que se puedan presentar alrededor de la alta disponibilidad de contenedores.

- Automatic Deployment of a Network Overlay in an Intelligent Transportation System: Docker and Open Baton Approach [25]

  En este documento cuentan (entre otras cosas), la necesidad de utilizar una herramienta como Portainer para administrar todos los componentes relacionados con Docker; indica también que por medio de estar herramienta logran organizar todos los detalles relevantes y visualizarlos de forma gráfica, logrando de esa forma simplificar de gran manera tanto la configuración como la revisión de los procesos.
  Este documento tiene una relación directa con la arquitectura que se ha planteado, debido al uso que le brindan a la herramienta Portainer, y a la especificación detallada de las ventajas que han aplicado para su proyecto.

3.3 Definición de términos básicos

- Apache
  Es un software de servidor web HTTP muy utilizado desde hace ya varios años. Actualmente, este software hace parte de los servidores Front end que tiene la actual arquitectura de las aplicaciones de la Pontificia Universidad Javeriana Cali, de manera que es vital tener presente su uso actual y el que se planteó [26].

- Portainer
  Aplicación web gratuita orientada a gestionar un entorno de contenedores Docker casi de manera integral. Desde Portainer es posible subir contenedores, gestionar imágenes, manejar volúmenes de datos persistentes y controlar cualquier servidor Docker (local o remoto) que se tenga en la infraestructura [27].

- Kubernetes
  Kubernetes como tal es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios [28]. Se puede pensar en aplicar para muchos aspectos, tales como plataformas de microservicios, contenedores, entre otras; para el contexto actual, se piensa como una futura herramienta, que permita realizar una administración centralizada de contenedores en una arquitectura de alta complejidad o más extensa.

- Prototipo
  El concepto de prototipo a nivel general, indica que es una implementación parcial pero concreta de un sistema o una parte de este, el cual tiene como objetivo principal explorar detalles sobre diferentes aspectos del sistema, durante el desarrollo o implementación de este [29].
  El concepto se interpreta para aplicar en el proyecto durante la fase de implementación, ya que el entregable es una arquitectura prototipo de lo planteado.

- Tomcat
  el término indica que es un contenedor de servlets que se utiliza para que se logre implementar Java Servlets, JavaServer Pages, Java Expression Languaje y Java WebSocket [30].

4. METODOLOGÍA

La metodología utilizada incluye una descripción de actividades basada en los objetivos específicos previamente planteados, en el proceso de diseño de arquitectura de Ian Gorton (Figura 4) y en la descripción general de diseño de arquitectura del arquitecto © Brett Lamb (Figura 5).

La Figura 4 se utiliza como una guía teórica del proceso de tres pasos en el diseño de la arquitectura que plantea el autor, los cuales coinciden en muchos aspectos con los objetivos específicos planteados. La Figura 5 se acota por el hecho de que considera puntos como las funcionalidades, los atributos de calidad y las constantes, para tomar decisiones a nivel del diseño de la arquitectura.

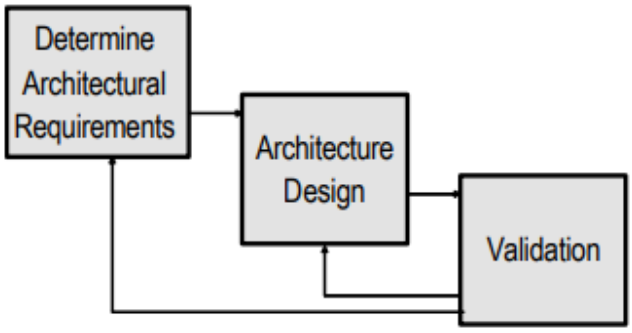


Figura 4. Proceso de arquitectura de software [5]

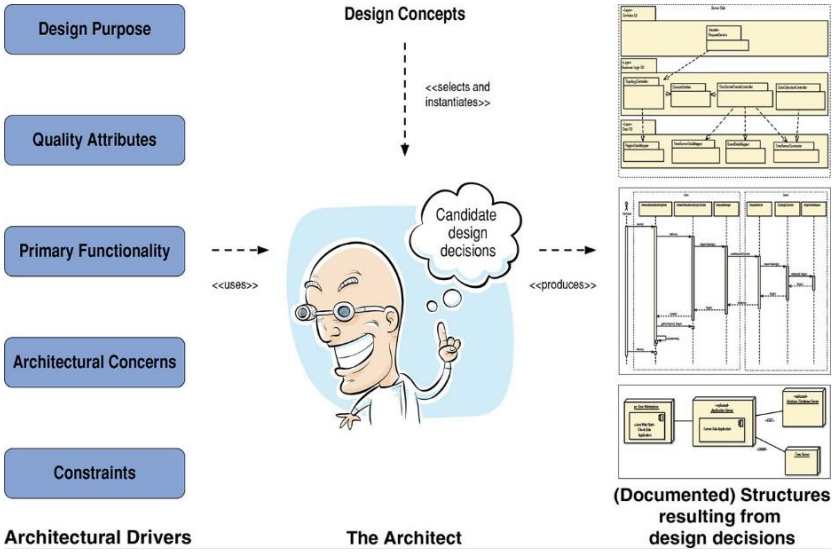


Figura 5. Diseño en Arquitectura de Software [31]

5. IDENTIFICACIÓN DE ALTERNATIVAS PARA PLANTEAR UNA ARQUITECTURA DE ACUERDO CON LAS NECESIDADES IDENTIFICADAS

5.1 Estado actual

Cómo se ha mencionado en diferentes secciones a lo largo del documento, la Pontificia Universidad Javeriana Cali maneja la mayoría de sus aplicaciones web en contenedores de servlets Apache Tomcat, las cuales se encuentran en diferentes servidores virtuales; adicionalmente, la arquitectura de referencia utilizada normalmente es la de monolito, bajo una infraestructura de máquinas virtualizadas, cuyo sistema operativo utilizado para los Tomcat, es Windows Server. Como se maneja una arquitectura monolítica, gran parte de sus aplicaciones se encuentran referenciados a una instancia de base de datos con un motor específico, necesario para el funcionamiento de sus componentes de software.

En la Figura 6, se representa el estado actual de la arquitectura general de un servidor que aloja varias instancias de Tomcat y diferentes aplicaciones:

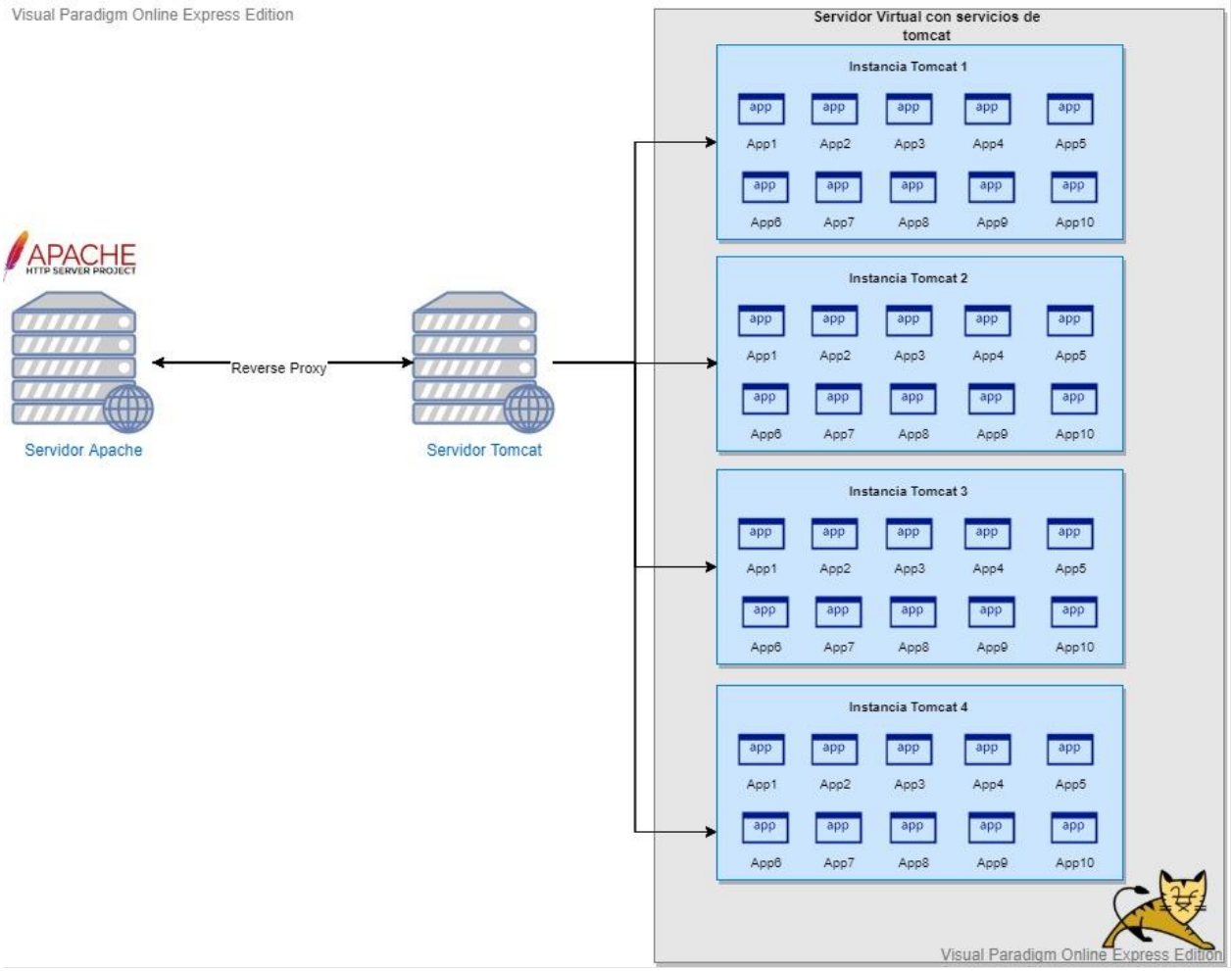


Figura 6. Diagrama de estado actual.

A continuación, se detallan las dificultades que brinda la combinación de una infraestructura virtual para alojar servicios de Tomcat como su principal objetivo, teniendo en cuenta que es la actual arquitectura que utiliza la Universidad.

- Consumo de recurso por parte del servicio Tomcat y sus derivados:
  Uno de los puntos que se identificó, era el del consumo de recursos de las máquinas que soportan estos servicios. Y es que cuando se habla de Tomcat, inmediatamente se deben validar los requisitos para que funcione un servicio como estos; y es en ese momento donde nos encontramos con el JDK, elemento esencial para el funcionamiento del Tomcat. En la máquina virtual de Java para cada instancia de Tomcat, se deben definir diferentes parámetros, entre los cuales se encuentran algunos a nivel de consumo de recursos, de acuerdo con la exigencia que se le realice a la instancia; la siguiente es una imagen de algunas opciones de configuraciones a nivel de JVM que se pueden realizar:

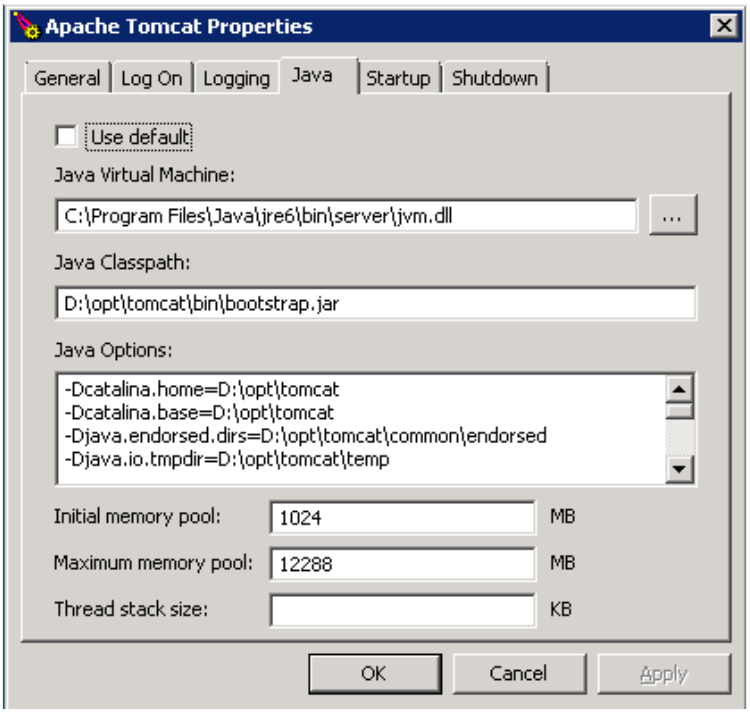


Figura 7. Propiedades de Tomcat [32]

Actualmente, estos parámetros son configurados por parte de los ingenieros de la Universidad, de acuerdo con la cantidad de instancias que contenga el servidor (normalmente 4 instancias máximo), y a la cantidad de recursos asignados (normalmente entre 10 y 12gb de memoria RAM. A nivel de CPU entre 4 y 8 core) y la cantidad de aplicaciones por instancia (normalmente 10).

En la Figura 8 se identifica un ejemplo de un servidor de la Universidad que cuenta con las características mencionadas a nivel teórico, cuyo consumo de recursos es superior al 70% en memoria RAM:

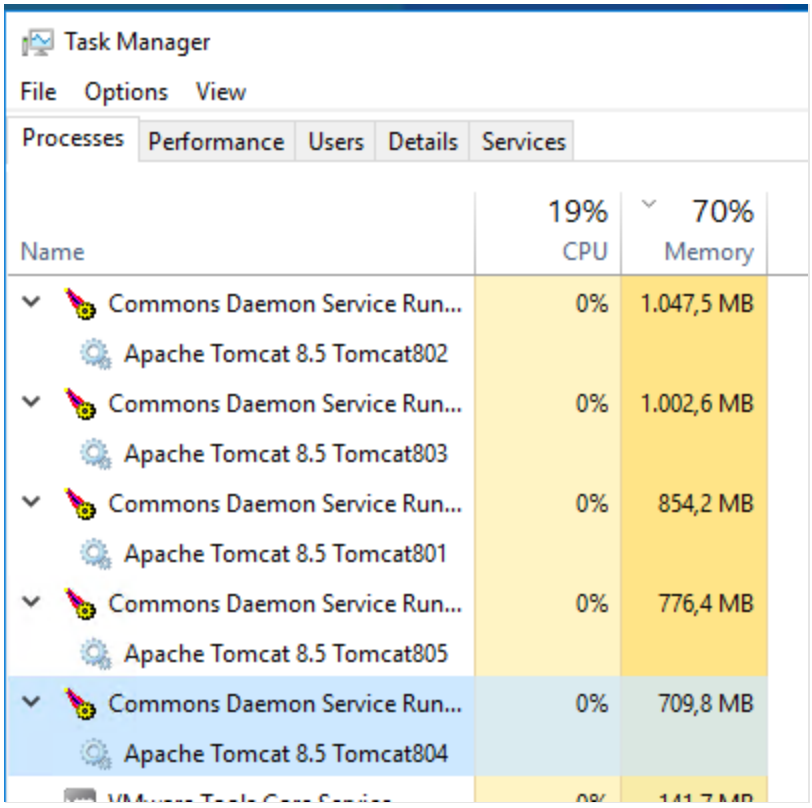


Figura 8. Consumo de recursos

- Otros de los puntos mencionados en el planteamiento del problema, se encuentra asociado a la alta disponibilidad de los diferentes servicios que se alojan en dichas instancias de Tomcat. Y aunque este es un tema que en ocasiones requiere de configuraciones a nivel de aplicación (tema de manejo de sesión, persistencia de estas, etc.), se considera este punto para asociarlo a brindar una alta disponibilidad en el contenedor a nivel de infraestructura (detalles que más adelante se especificarán de forma técnica). Por lo pronto, actualmente la mayoría de las aplicaciones se encuentran alojadas en una única instancia de Tomcat, la cual, en caso de falla, genera indisponibilidad inmediata.

- Otro aspecto mencionado en el planteamiento del problema aborda el punto de los ambientes a nivel de aplicación que actualmente se utilizan, determinados como: réplica, test y desarrollo. A continuación, bajo la Figura 9 se especifican los servidores existentes:

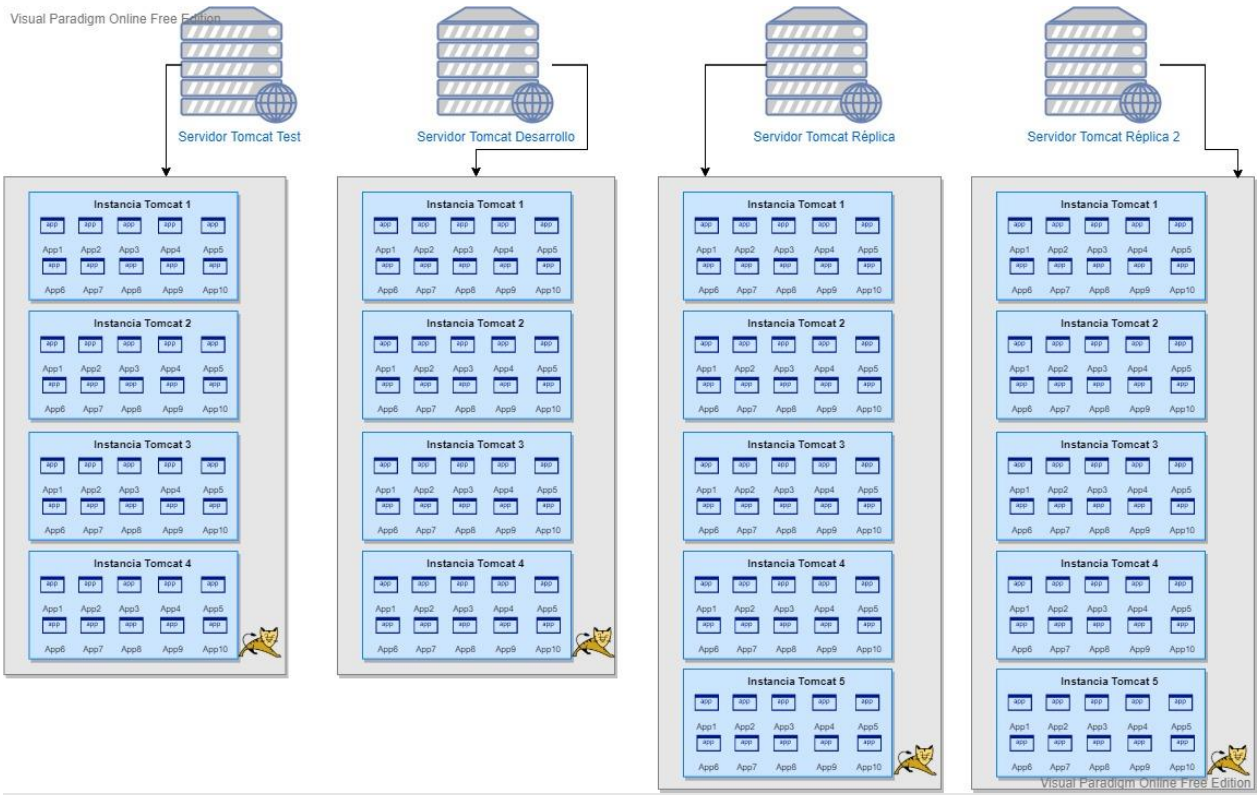


Figura 9. Ambientes de pruebas

- Servidor Tomcat de réplica: como su nombre lo indica, es el servidor de réplica utilizado para realizar las validaciones correspondientes a pruebas con un ambiente muy parecido a producción. Actualmente estas aplicaciones deben conectarse exclusivamente al ambiente de base de datos de réplica
- Servidor Tomcat de test: como su nombre lo indica, es el servidor de pruebas utilizado para realizar las validaciones correspondientes a la implementación de nuevas funcionalidades en determinado desarrollo que se esté realizando. Actualmente estas aplicaciones deben conectarse exclusivamente al ambiente de base de datos de test
- Servidor Tomcat de desarrollo: como su nombre lo indica, es el servidor donde se prueban todas las aplicaciones que estén sufriendo cambios, y que sirve de base para desplegar las mismas mientras se desarrolla. Actualmente estas aplicaciones deben conectarse exclusivamente al ambiente de base de datos de desarrollo, la cual contiene data de pruebas, con información diferente a producción

- Servidor réplica 2: Actualmente es un servidor que cuenta con una versión actualizada de Tomcat, el cual ha servido para realizar las validaciones previas para subir de versión bajo la que se alojará una aplicación.

La novedad que sucede con la anterior infraestructura mencionada se registra a nivel de las diferencias que pueden presentarse cuando se pasa una aplicación de estos ambientes a producción, teniendo en cuenta que, aunque se quiera tener exactamente igual, es una labor compleja debido a diferentes factores como: las actualizaciones de sistema operativo, cambios en instancias de Tomcat (jvm, certificados, etc.), entre otros factores. Actualmente se trata de conservar de una manera manual los ambientes muy parecidos a los productivos, sin embargo, cuando se presentan novedades que funciona en todos los ambientes de pruebas, pero no producción, se validan configuraciones a nivel detallado en servidor, instancia de Tomcat, máquina virtual de Java, entre otros, lo cual genera demoras en entregas a producción, indisponibilidad, aumento de carga operativa, entre otros.

5.2 Identificación de requisitos

5.2.1 Requisitos no funcionales

ID	Descripción del requisito
RNF1	El Sistema debe garantizar la comunicación entre todos los componentes de un contenedor
RNF2	El sistema debe contar con contenedores que brinden una disponibilidad del 99.44% en sus servicios
RNF3	Los servicios alojados en los contenedores deben estar disponibles las 24 horas del día, 7 días a la semana, durante los 365 días del año, teniendo en cuenta que se considera un periodo de mantenimiento de 4 horas mensual. (24hrs x 30 días = 720 – 4 hrs de mantenimiento = 716hrs.). Adicionalmente, se contemplan 4 horas de rollback ante activación del DRP. Se utiliza la formula Disponibilidad = (Horas totales – Horas de mantenimiento / Horas totales) X 100

	CA= (720 – 4/720) * 100 = 99.44%
RNF4	Los contenedores del sistema deben incluir todos los elementos que permitan el correcto funcionamiento de este, cuando se realice una migración, clon o traslado
RNF5	Los Tomcat alojados en los contenedores, deben contar con una parametrización de uso de máquina virtual de Java limitada, de acuerdo con los recursos totales del servidor
RNF6	El sistema debe soportar un entorno gráfico robusto, para administrar los contenedores de forma adecuada
RNF7	El sistema debe contar con la documentación necesaria para implementar y extender la arquitectura en caso de ser necesario
RNF8	El sistema debe utilizar una versión de Tomcat estable, teniendo en cuenta las recomendaciones de seguridad que brinda el área encargada
RNF9	El sistema no debe superar el 70% en consumo de recursos (CPU, RAM), bajo circunstancias similares a las actuales
RNF10	El sistema debe contar con las reglas de acceso local a los contenedores correctamente afinadas a nivel de seguridad
RNF11	El sistema debe ser escalable de manera horizontal y vertical
RNF12	El sistema debe dejar registro a nivel de logs de los cambios ejecutados en los diferentes componentes
RNF13	El sistema debe contar con contenedores que se encuentren estructurados para realizar un adecuado manejo de fallos de forma automática (en caso de realizar la implementación de estas estrategias despliegue automático)
RNF14	El sistema debe brindar una alta disponibilidad a nivel de los servicios de sus contenedores, por medio de estrategias que permitan disponer los mismos
RNF15	El sistema y sus contenedores deben poder desplegarse en otros ambientes que soporten la tecnología planteada

Tabla 1. Requisitos no funcionales

5.2.2 Identificación de restricciones

ID	Descripción del requisito
RESTR01	La arquitectura debe contener versiones Tomcat y Java estables
RESTR02	La versión demo debe configurarse bajo la arquitectura y política de servidores del centro de Datos utilizado por la Universidad
RESTR03	La arquitectura debe ser contemplada para configurar bajo un sistema operativo base Windows Server 2019, y con contenedores que tengan como base una determinada distribución de Linux

Tabla 2. Identificación de requisitos

5.2.3 Priorización de requisitos

Se utilizará la técnica de tres niveles [33]:

- Alta prioridad
- Media prioridad
- Baja prioridad

Requisitos	Prioridad
RNF01	Alta
RNF02	Alta
RNF03	Alta
RNF04	Alta
RNF05	Media
RNF06	Baja
RNF07	Media
RNF08	Media
RNF09	Alta
RNF10	Media
RNF11	Alta
RNF12	Baja
RNF13	Media
RNF14	Alta
RNF15	Alta

Tabla 3. Priorización de requisitos

5.3 Atributos de calidad y requisitos

5.3.1 Atributos de Calidad

A continuación, se agruparán los requisitos en los diferentes atributos de calidad identificados, se observarán requisitos no funcionales en más de una clasificación ya que los requisitos podrán corresponder a más de un atributo al mismo tiempo.

Atributo	Requisito
Disponibilidad	RNF1, RNF2, RNF3, RNF9, RNF13, RNF14
Portabilidad	RNF4, RNF11, RNF15
Usabilidad	RNF6
Mantenibilidad	RNF6, RNF13
Seguridad	RNF8, RNF12
Performance	RNF9
Sin clasificación	RNF5

Tabla 4. Atributos de Calidad

6. ARQUITECTURA EN CONTENEDORES DOCKER DE ALTA DISPONIBILIDAD

6.1 Patrones, tácticas y arquitecturas de referencia

6.1.1 Patrones y arquitecturas de referencia

Normalmente cuando se inicia un diseño de una arquitectura para sistemas de información se estructura para resolver una variedad de problemas a nivel general. Usualmente, las estructuras arquitectónicas planteadas deben estar basadas en uno o más patrones.

Por eso, uno de los primeros pasos es la selección de el o los patrones que permitirán estructurar la arquitectura, basado en los problemas que se necesitan solventar.

Cuando se realiza la revisión detallada de los patrones a utilizar, se ejecuta una revisión precisa de tres aspectos claves: contexto, problema y solución.

Basados en los problemas planteados en el capítulo correspondiente, y teniendo en cuenta que el proyecto consiste en plantear una arquitectura de alta disponibilidad en contenedores para la Universidad Javeriana Cali, se utiliza como referencia el patrón de arquitectura de microservicios; es importante dejar claro que se usa como referencia, toda vez de que no se desarrollará ni se ajustará una aplicación de software determinada o se separarán las bases de datos, pero si se dejará el diseño adaptado ante una posible migración de aplicaciones a microservicios, teniendo en cuenta de que se adoptará una arquitectura basada en contenedores Docker.

6.1.1.1 Microservicios

Dado el contexto anterior, a continuación, se explica un poco el uso de la arquitectura microservicios a nivel general.

Normalmente, cuando se toca el tema de una arquitectura de microservicios, se plantean como referencias algunos diseños como el de la Figura 10.

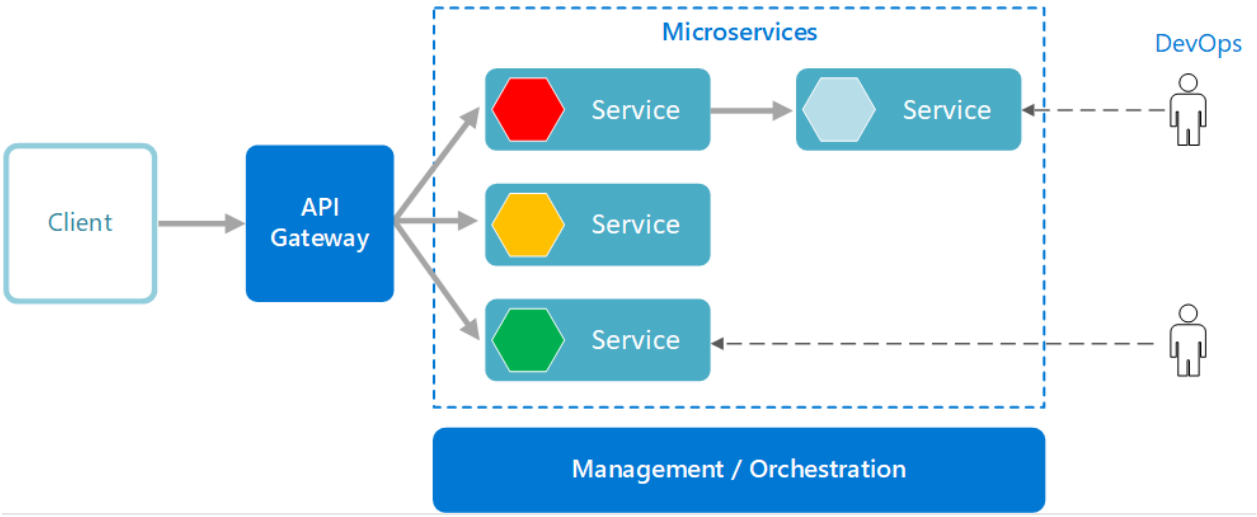


Figura 10. Estilo de arquitectura de Microservicios [34]

Por lo general, una arquitectura microservicios contiene una colección de servicios autónomos pequeños e independientes. Cada uno de ellos se encarga de implementar determinada funcionalidad del negocio. Sin embargo, como se comentaba anteriormente, se ha acotado esta arquitectura de referencia, toda vez de que para el diseño que se plantea en el presente documento, se utiliza como base el uso de contenedores Docker, elemento que es utilizado usualmente para las arquitecturas de microservicios.

En la sección del modelo c4 y en la de la solución propuesta, se plantean varios diagramas que indican el uso que se le da al anterior modelo de referencia, adaptado al contexto del proyecto (no utilizará microservicios, pero si contenedores Docker).

Como se puede visualizar en dichos diagramas, este tipo de arquitecturas permite que en el futuro se puedan alojar sistemas de información diseñados en un modelo de microservicios.

6.1.1.2 Arquitectura Docker

Docker como tal, tiene una arquitectura de cliente servidor, donde al mencionar cliente se hace referencia al programa que ejecuta el equipo y el servidor es equivalente al *daemon* o servicio que se encarga de crear los contenedores. En la Figura 11 se representa la arquitectura de referencia de Docker.

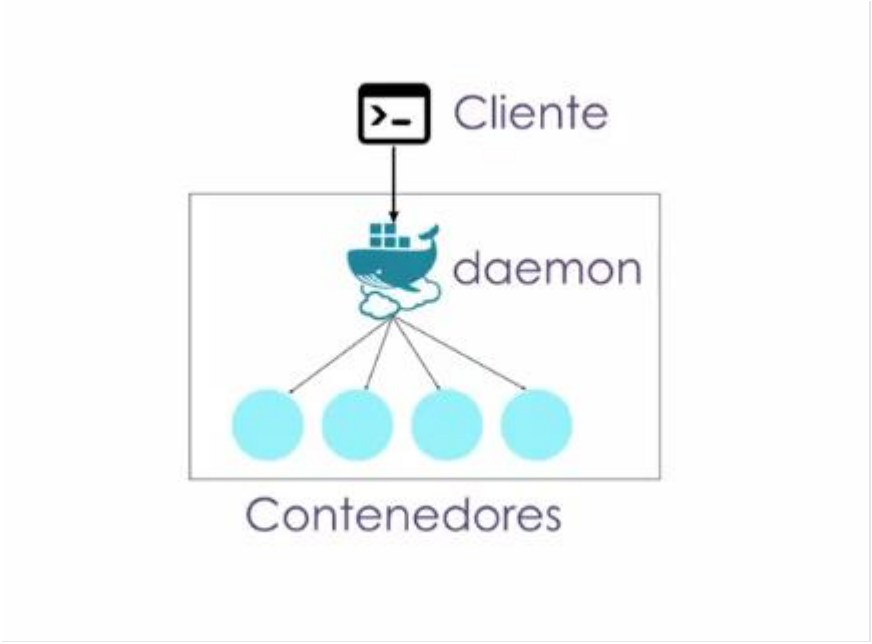


Figura 11. Arquitectura Docker [35]

6.1.1.3 Contenedores

Cuando se habla de Docker, es fundamental involucrar el concepto de contenedores. Un contenedor como definición plana, es una entidad lógica y una agrupación de procesos que se ejecuta de forma nativa, los cuales comparten con el host o la máquina sobre la que corran, el kernel del sistema operativo.

El siguiente es el flujo de construcción de Docker:

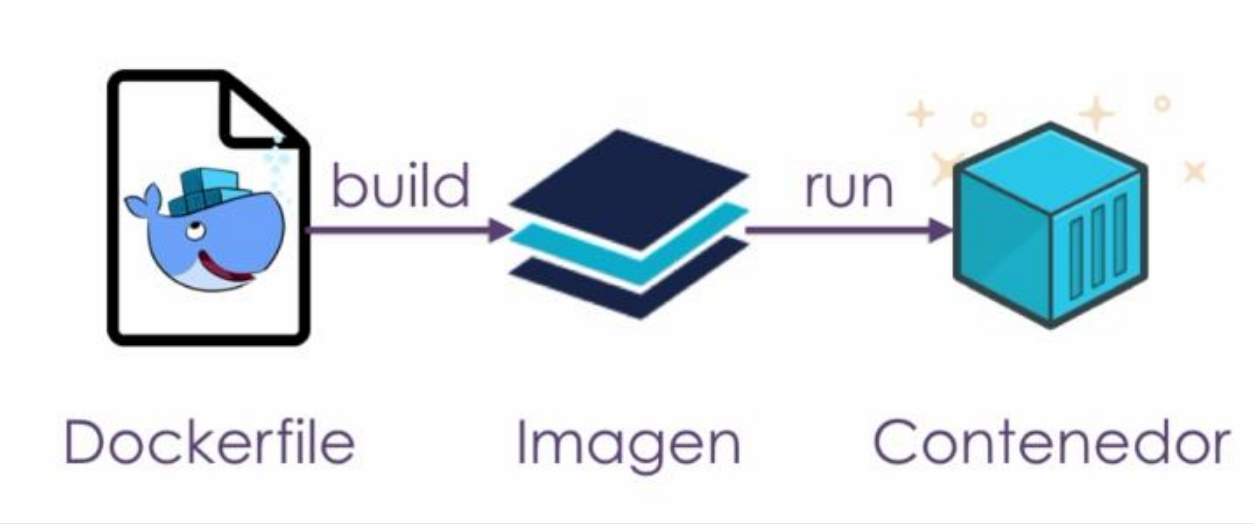


Figura 12. Flujo de construcción de Docker [35]

6.1.1.4 Portainer

Esta herramienta cumple la función en la arquitectura planteada, de permitir administrar y gestionar las diferentes opciones que incluye todo lo referente a contenedores Docker (imágenes, despliegues, conexiones, entre otros). Se contempla esta herramienta para uso administrativo, sin embargo, se deja la posibilidad de que cuando la compañía se encuentre madura en temas de contenedores o cuente con una cantidad alta de estos elementos, se puede complementar con otras herramientas como Kubernetes o Rancher.

6.2 Escenarios de calidad

6.2.1 Atributos de calidad de la arquitectura

A continuación, se indica el concepto de los atributos de calidad seleccionados, previamente listados en la sección de requisitos

Disponibilidad: Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere [36].

Portabilidad: Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro [36].

Usabilidad: Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones [36].

Mantenibilidad: Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas[36].

Seguridad: Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos [36].

Performance: El atributo de calidad Performance se refiere a la capacidad de responder, ya sea el tiempo requerido para responder a eventos determinados, o bien, la cantidad de eventos procesados en un intervalo de tiempo dado. La Performance caracteriza la proyección en el tiempo de los servicios entregados por el sistema.

6.2.2 Tácticas

Los patrones abordan muchos atributos de calidad, pero una táctica aborda un atributo de calidad particular. La arquitectura de software de un sistema es una colección de decisiones de diseño, y esto va directamente relacionado con las tácticas o decisiones que se tomen para abordar determinado escenario o atributo de calidad.

Las tácticas se van a detallar en el formato de escenario de calidad planteado, el cual se realiza por cada atributo de calidad determinado; se compone de constantes decisiones que permite cumplir los diferentes atributos de calidad que se seleccionen, a esas decisiones se le llama tácticas.
Las tácticas influyen de forma directa en el manejo que se brinde a las respuestas en los atributos de calidad, de acuerdo con los diferentes estímulos; la Figura 13 indica lo mencionado.

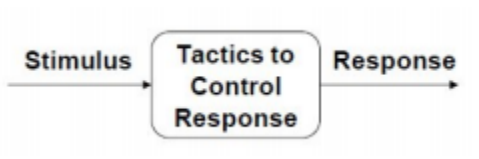


Figura 13. Respuesta - atributos de calidad [37]

6.2.3 Formato de escenario de calidad

En la Figura 14 se indican las partes que normalmente componen un escenario de calidad.

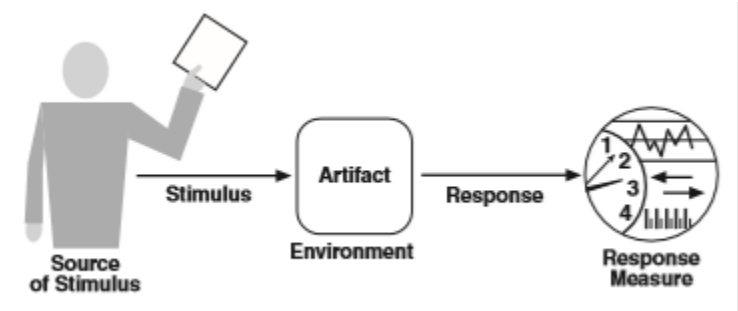


Figura 14. Partes de un Escenario de Calidad [37]

En la tabla 5 se indica el formato de escenario de calidad que se ha definido para el proyecto.

Parte	Valores
Origen	Fuente del estímulo: alguna entidad (un ser humano, un sistema informático o cualquier otro actuador) que generó el estímulo.
Estímulos	Ocurre una falla; El estímulo es una condición que requiere una respuesta cuando llega a un sistema.
Artefacto	Recurso que recibe el estimulo
Ambiente	El estímulo ocurre bajo ciertas condiciones. El sistema puede estar en una condición de sobrecarga o en funcionamiento normal, o en algún otro estado relevante. Para muchos sistemas, el funcionamiento "normal" puede referirse a uno de varios modos. Para este tipo de sistemas, el entorno debe especificar en qué modo se está ejecutando el sistema.
Respuesta	Es la actividad que se realiza como resultado de la llegada del estímulo.
Medición de la respuesta	Cuando se produce la respuesta, debe ser medible de alguna manera para que el requisito se pueda probar.
Tácticas	Toma de decisiones para brindar respuestas a los atributos de calidad

Tabla 5. Formato de Escenario de Calidad [37]

6.2.4 Detalle de escenarios de calidad

El detalle de los escenarios de disponibilidad, se especificará un caso por cada atributo de calidad definido previamente.

Disponibilidad

El sistema debe contar con contenedores que brinden una disponibilidad del 99.44% en sus servicios

Parte	Valores
Origen	Hardware y software
Estímulos	Falla en los componentes del contenedor
Artefacto	Servicios instalados en el contenedor
Ambiente	Operación normal
Respuesta	Notificación con la falla (sistema de monitoreo de infraestructura), con herramientas como zabbix o server alive, las cuales permiten notificaciones al equipo de infraestructura o noc, vía herramientas como Telegram o correo electrónico. Registrar en el log (novedad en los servicios)
Medición de la respuesta	- Denegación de servicio el menor tiempo posible (< 5 minutos)
Tácticas	- Uso de sistema de monitoreo para componentes del contenedor - Reinicio del contenedor en caso de falla, por medio de un script que corre como crontab que valida el estado de los servicios y reinicia en caso de estar abajo - Realizar reinicio de servicios de forma periódica, para temas de refrescar caché (sucede con aplicativos que usan la JVM). Lo anterior se puede programar con un script en bash, Python o en perl, para que el mismo sea automatizado - Reinicio de servicio de forma manual por parte del ingeniero encargado - Múltiples contenedores con servicios - Balanceadores de carga (ej: HAProxy) - Sí hay servidores con consumo de contenedores superiores al 80% de memoria RAM, se debe realizar un crecimiento horizontal o vertical de la infraestructura

Tabla 6. Escenario de calidad de Disponibilidad

Portabilidad

Los contenedores del sistema deben incluir todos los elementos que permitan el correcto funcionamiento de este, cuando se realice una migración, clon o traslado

Parte	Valores
Origen	Hardware y software
Estímulos	Necesidad de migrar el contenedor a otra infraestructura
Artefacto	Software
Ambiente	Migración
Respuesta	Despliegue de contenedor en otra infraestructura, de forma manual o automática
Medición de la respuesta	- Despliegue del contenedor en otra infraestructura o sistema de forma correcta - Cero errores críticos al migrar
Tácticas	- Migración de forma manual por parte de alguno de los ingenieros encargados - Migración de forma automática con notificaciones al correo del proceso que se está realizando, con scripts hechos en bash, perl o Python, que identifiquen caída de un contenedor y procedan a realizar el correspondiente despliegue

Tabla 7. Escenario de calidad de Portabilidad

Usabilidad

El sistema debe contar con un entorno gráfico para administrar los contenedores

Parte	Valores
Origen	Hardware y software
Estímulos	Administración de contenedores desde un entorno gráfico
Artefacto	Software
Ambiente	Operación normal
Respuesta	Experiencia amigable e intuitiva para administrar contenedores
Medición de la respuesta	- Actividades correspondientes a administración de contenedores de forma rápida

	- Personal de soporte podría utilizar la pantalla, con una capacitación básica por parte del equipo de infraestructura
Tácticas	- Utilizar herramientas como Portainer - Utilizar una interfaz de usuario web como Portainer, para administrar los contenedores docker

Tabla 8. Escenario de calidad de Usabilidad

Mantenibilidad

El sistema debe contar con contenedores que se encuentren estructurados para realizar un adecuado manejo de fallos de forma automática

Parte	Valores
Origen	Hardware y software
Estímulos	Novedades con el funcionamiento de un contenedor
Artefacto	Software
Ambiente	Operación normal
Respuesta	- Notificación con la falla (sistema de monitoreo de infraestructura), con herramientas como zabbix o server alive, las cuales permiten notificaciones al equipo de infraestructura o noc, vía herramientas como Telegram o correo electrónico. - Registro en el log de las aplicaciones, servidores de base de datos y administrador de contenedores - Intento de recuperación automática
Medición de la respuesta	- Se puede utilizar un script en bash, perl o Python, que identifique el down de algún contenedor y realice el correspondiente despliegue - Denegación de servicio el menor tiempo posible
Tácticas	- Utilizar herramientas como Portainer para revisar la recuperación automática o manual por parte del equipo de Monitoreo

Tabla 9. Escenario de calidad de Mantenibilidad

Seguridad

El sistema debe utilizar la última versión más actualizada de tomcat aplicada por la Universidad Javeriana Cali

Parte	Valores
Origen	Software
Estímulos	Conservar versión de Tomcat actualizada
Artefacto	Software
Ambiente	Operación normal
Respuesta	- Instalación inicial de última versión - Alerta de lanzamientos de nuevas versiones
Medición de la respuesta	- El Tomcat debe contar con las últimas actualizaciones a nivel de binarios
Tácticas	- Programar actualizaciones de Tomcat con el equipo de desarrollo y seguridad - Realizar un plan de pruebas para validar las versiones a instalar (cuando sean liberadas) - Validar compatibilidad de versiones de Tomcat con aplicaciones actuales

Tabla 10. Escenario de calidad de Seguridad

El sistema debe utilizar una herramienta de doble factor de autenticación, para el acceso vía https al administrador de aplicaciones y balanceador de carga

Parte	Valores
Origen	Software
Estímulos	Acceso a los componentes
Artefacto	Software
Ambiente	Operación normal
Respuesta	- Acceso con herramienta de doble factor
Medición de la respuesta	- El acceso debe permitirse exclusivamente con la herramienta de doble factor
Tácticas	- Configurar SSO para acceso a estos componentes - Configuración de diferentes métodos de doble factor, por medio de herramientas como MobilityGuard o FortiClient

Tabla 11. Escenario de calidad de Seguridad

Performance

El sistema no debe superar el 80% en consumo de recursos, bajo circunstancias similares a las actuales

Parte	Valores
Origen	Software y hardware
Estímulos	Múltiples conexiones a los servicios de los contenedores
Artefacto	Software
Ambiente	Operación normal
Respuesta	- Entrega de información en los tiempos adecuados - Bajo consumo de recursos - Sin denegación de servicio
Medición de la respuesta	- El consumo de recursos de los servidores debe ser inferior al 80% durante la transmisión de información - La pérdida de paquetes con el servidor debe ser inferior a los 30ms
Tácticas	- Controlar el uso de memoria de los diferentes servicios del servidor - Aumentar memoria RAM, CPU o mover a un almacenamiento de SSD. Crecimiento vertical - Sí se identifican horarios determinados donde se generan los altos consumos, se debe realizar una revisión detallada de todos los ambientes que influyen en la arquitectura - Se debe realizar revisión de reportes de rendimiento de la arquitectura, con el fin de determinar posibles mejoras que se deban realizar con ventanas de mantenimiento programadas

Tabla 12. Escenario de calidad de Performance

6.3 Modelo C4

Habitualmente en el área de TI se manejan diferentes lenguajes para modelar ideas, sin embargo, esos lenguajes en ocasiones no comunican de forma efectiva temas de arquitectura. Es por ello que se crea el modelo c4, cuyo objetivo es ayudar a los equipos de arquitectura y desarrollo, a entender a un alto y bajo nivel diferentes aspectos con importantes niveles de detalle, permitiendo así comunicar de manera efectiva los objetivos de cada particularidad de la arquitectura.

Basado en lo anterior, se ha realizado el diseño de tres niveles del modelo c4, con el fin de comunicar de manera efectiva la arquitectura planteada. Tener en cuenta, que cada nivel se deja con dos versiones, toda vez que la misma ha sufrido cambios durante el desarrollo técnico y se consideró ideal dejar esa trazabilidad del diseño inicial versus el final.

Nivel 1 (Versión 1):

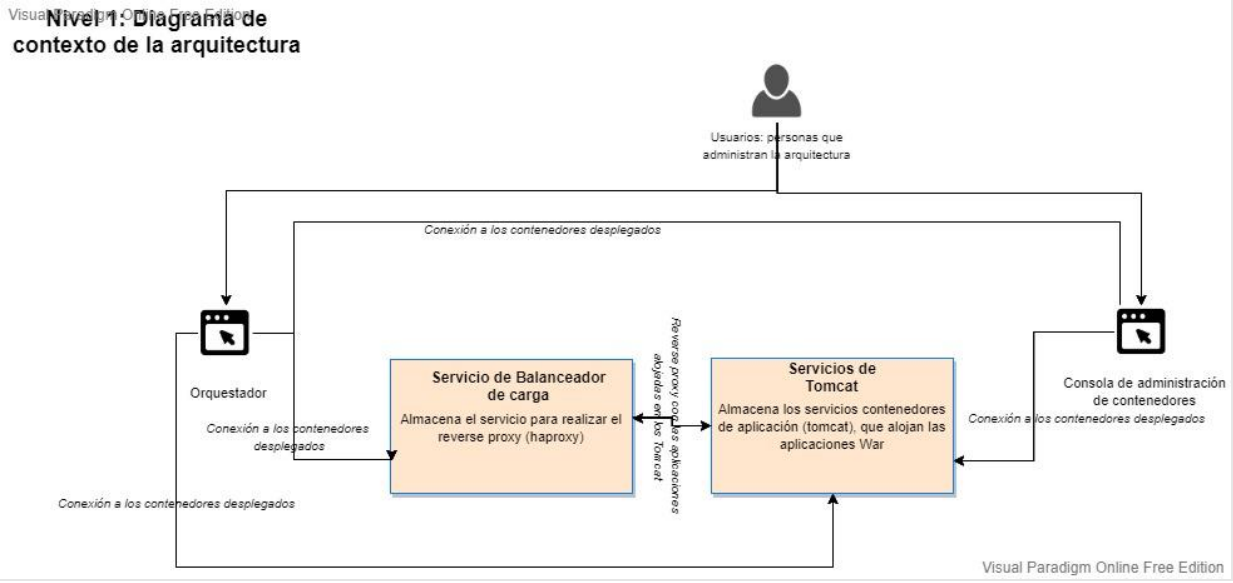


Figura 15. Nivel 1 (Versión 1)

Nivel 1 (Versión final):

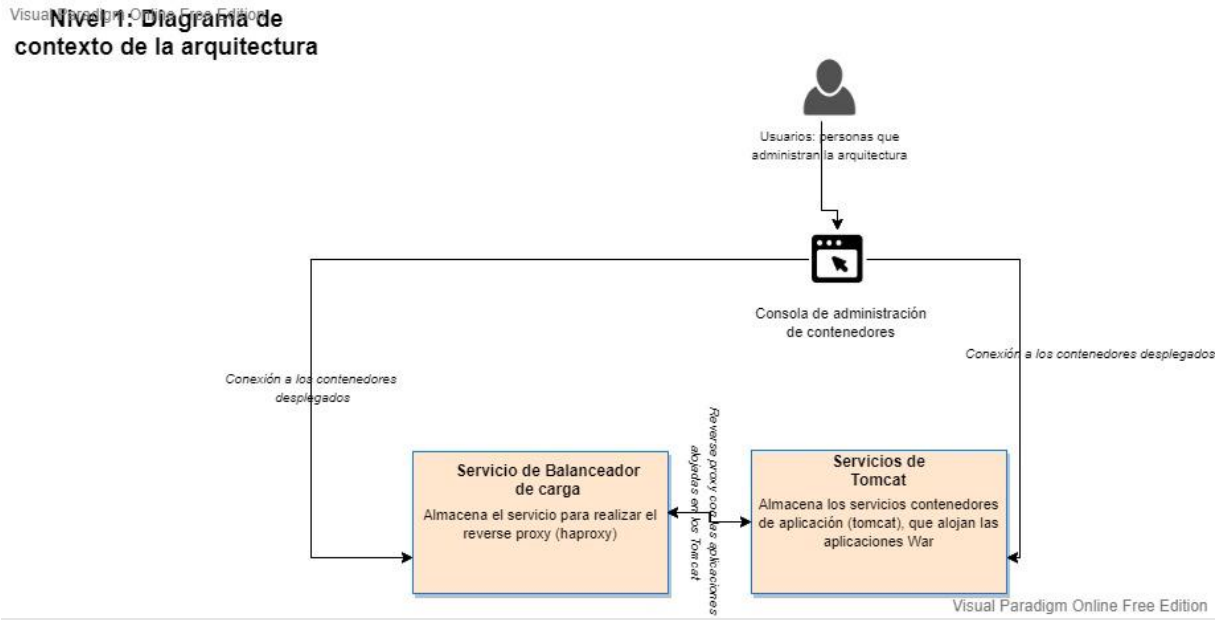


Figura 16. Nivel 1 (Versión final)

Con respecto de la versión inicial vs la versión final, el principal cambio pasa por la definición de una consola de administración de contenedores, dejando el tema del orquestador como posibles mejoras o trabajo futuro, toda vez que la infraestructura actual con contenedores estaría iniciando y el uso de este tipo de elementos se suele realizar en arquitecturas de múltiples servicios y con equipos que cuentan con experiencia en el campo de docker y temas de DevOps.

Nivel 2 (Versión 1):

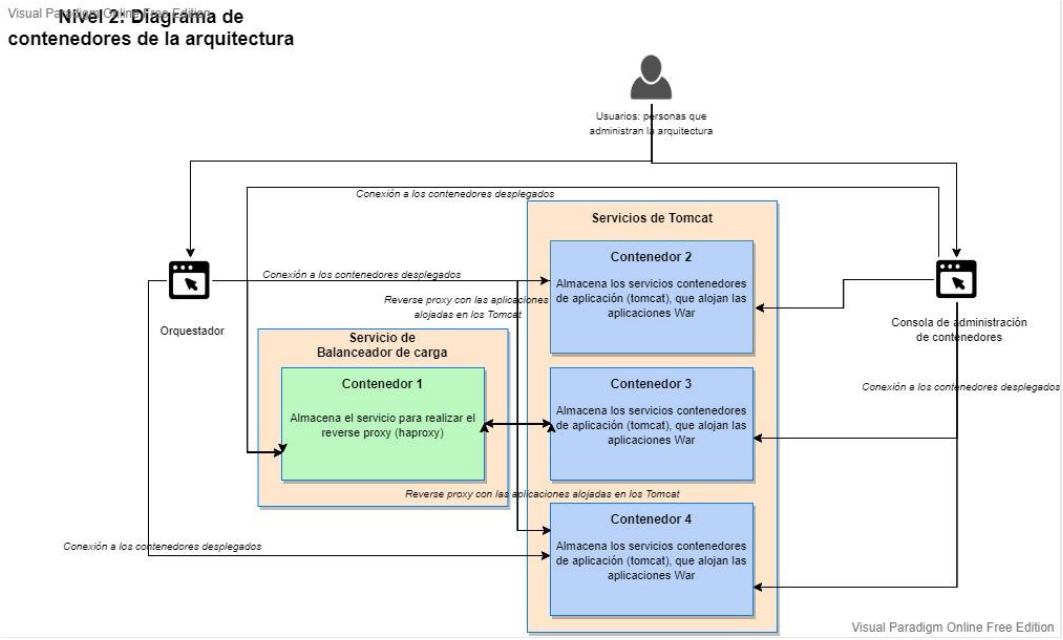


Figura 17. Nivel 2 (Versión 1)

Nivel 2 (Versión final):

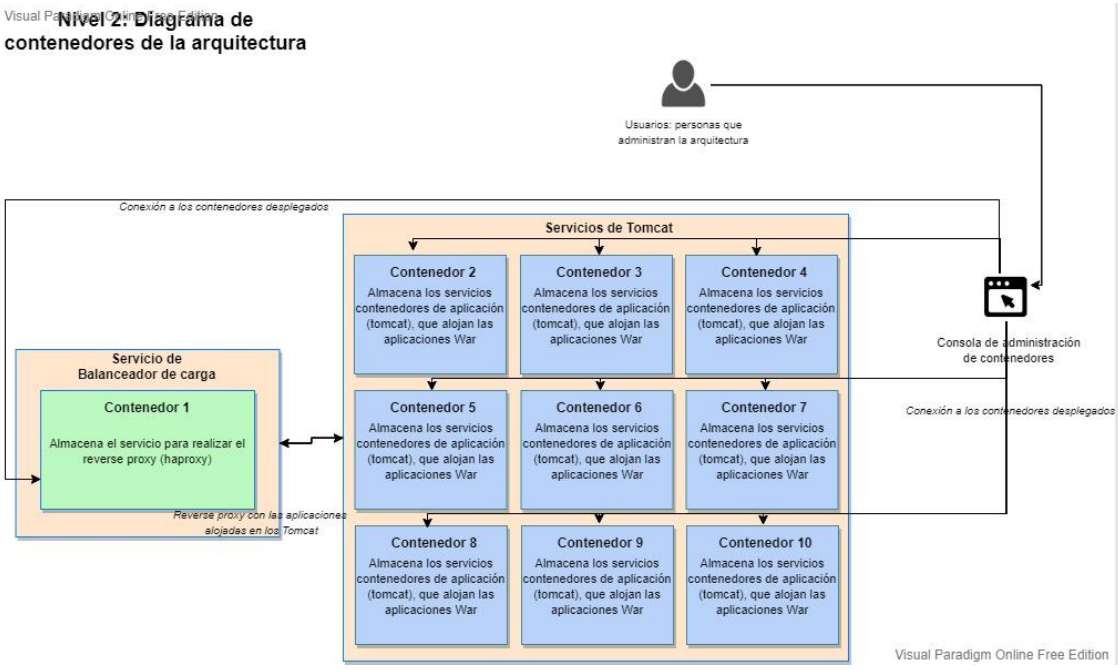


Figura 18. Nivel 2 (Versión final)

A medida que los niveles aumentan, se van detallando diferentes aspectos de la arquitectura. Por tal motivo, en este nivel se detallan temas referentes a los contenedores que soportarán la infraestructura y los servicios que tienen los mismos. El principal cambio entre la primera versión y la final pasa por el uso una consola de administración y el cambio de la cantidad de contenedores, teniendo en cuenta que la documentación recomienda un servicio por contenedor.

Nivel 3 (Versión 1):

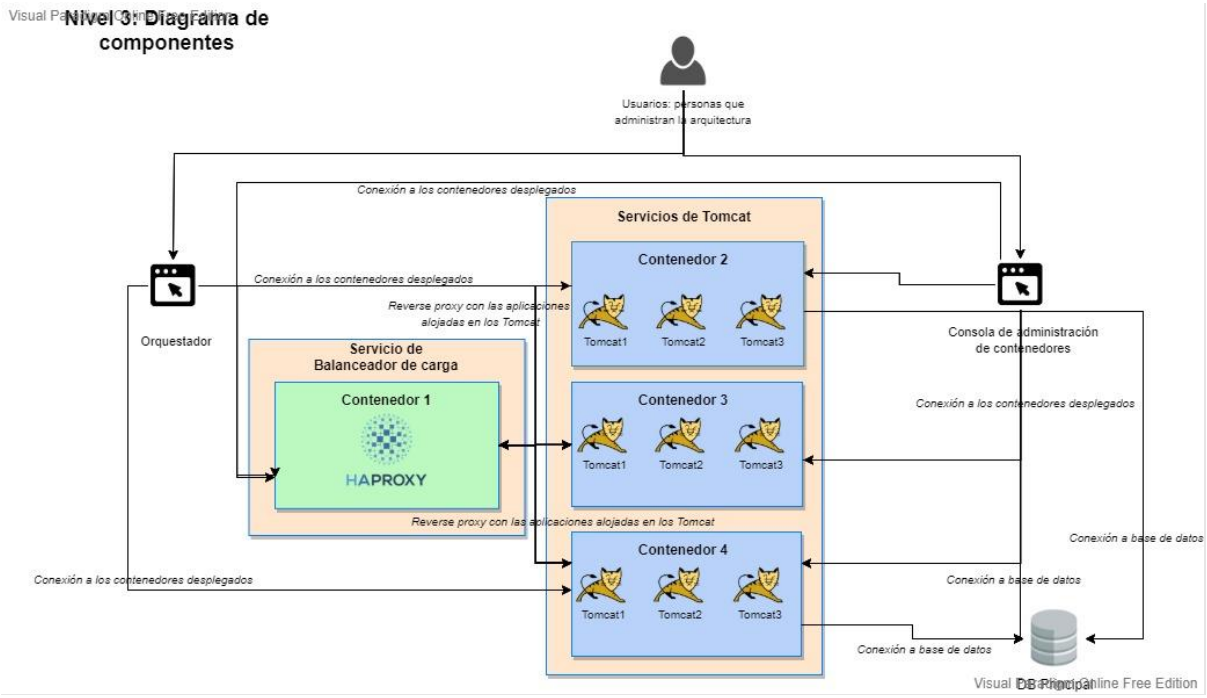


Figura 19. Nivel 3 (Versión 1)

Nivel 3 (Versión final):

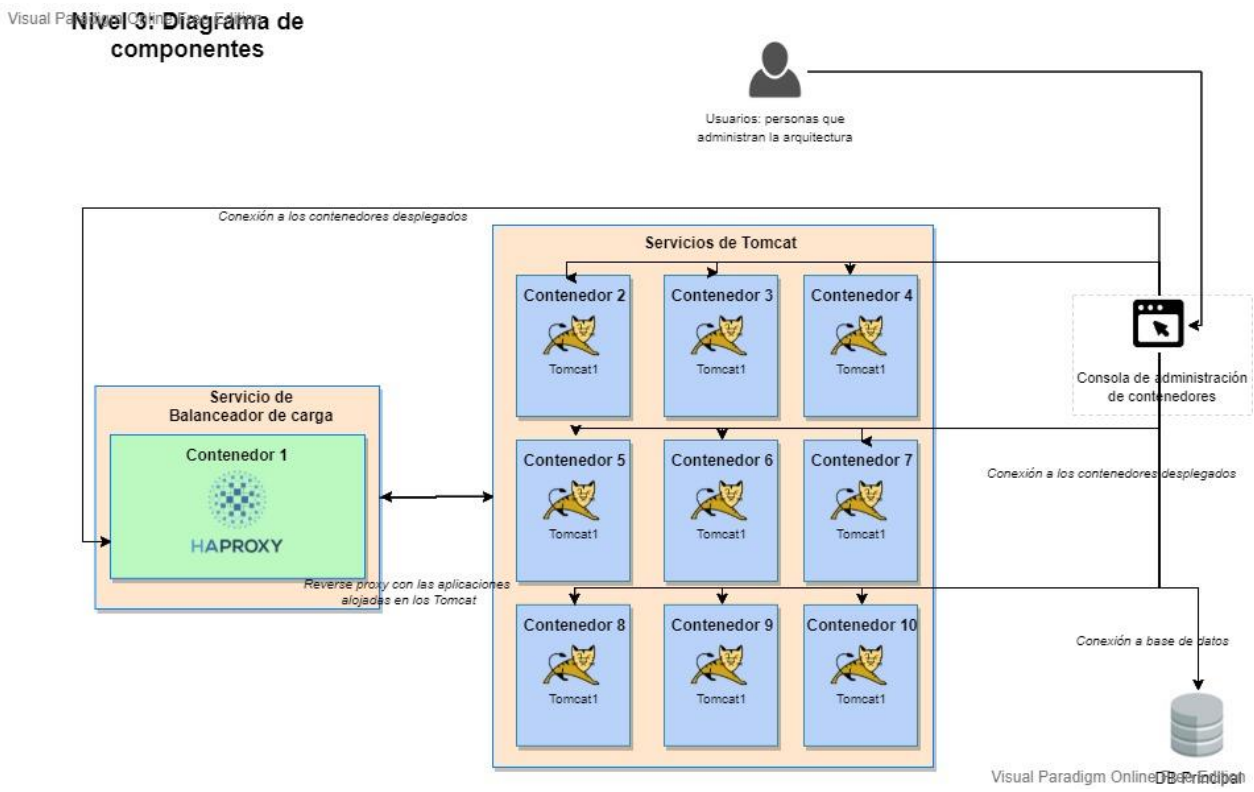


Figura 20. Nivel 3 (Versión final)

En este nivel se realiza una descomposición un poco más detallada, donde se especifican aspectos de servicios por contenedor y base de datos. Con respecto a los cambios entre la primera versión y la final, pasan por aspectos mencionados previamente, que hacían referencia a la consola de administración y a la cantidad de contenedores (cambio heredado).

6.3.1 Conclusiones modelo C4

Durante el desarrollo del proyecto, se planteó inicialmente algunos diagramas basados en el modelo C4, sin embargo, durante el desarrollo técnico fue necesario tomar algunas decisiones arquitecturales adicionales a las planteadas en la etapa teórica, lo cual generó cambios en los diagramas.
Por lo anterior, se deja documentado en cada nivel del diagrama una versión 1 y una versión final, teniendo como grandes cambios a nivel general los siguientes

Se cuenta solo con una consola de administración, no se utilizará para el proyecto un orquestador
La distribución de los servicios por contenedor cambia, teniendo en cuenta que luego de revisar documentación técnica se observó que como buena práctica se recomienda no dejar más de un servicio por contenedor.
Como se mencionó inicialmente, la necesidad de generar una versión adicional nació durante el

desarrollo técnico del proceso, teniendo en cuenta que en ocasiones en los laboratorios se identifican algunos aspectos que por diferentes razones no son posibles aplicar (tiempo, recursos, dinero, alcance, entre otros).

6.4 Decisiones arquitecturales

Para llegar a los anteriores diagramas expresados bajo el modelo C4, fue necesario realizar diferentes análisis para identificar los tipos de componentes a utilizar, de acuerdo con el contexto planteado.

Decisiones	Justificación
Arquitectura de microservicios	Como se indicaba en el capítulo anterior, se toma como referencia la arquitectura de microservicios, dejando claridad de que no se modificará nada a nivel de aplicación, y que el uso de este patrón es referente a la utilidad y al diseño de la arquitectura con contenedores Docker, las cuales contienen servicios para alojar aplicaciones web tipo war. Entre los motivos por tomar la decisión de usar como base la arquitectura de microservicios, se encuentran todos los atributos de calidad (ya mencionados) que este tipo de patrones permite utilizar. Al aplicar este tipo de arquitecturas, se deja la base (contenedores) para que posteriormente la compañía pueda incorporar las aplicaciones como microservicios, y ayude en temas como el de la cultura DevOps, que son tendencia en la actualidad. Se descarta el uso de una arquitectura monolítica, teniendo en cuenta que es la que se utiliza actualmente; aunque no se dividen las aplicaciones en microservicios, sí se deja una arquitectura bajo contenedores Docker, donde quedarán alojados los diferentes War; lo anterior permite que en el futuro se pueda

	realizar una división de aplicaciones, contando ya con las bases arquitecturales para alojarlas sin inconvenientes.
Balanceador de carga	Es necesario contar con un balanceo de carga a nivel de los servicios que se utilizarán; de manera que se utilizará HAProxy para realizar este proceso y una herramienta de gestión de contenedores llamada Portainer, para la administración de estos componentes y servicios
Servicios por contenedor	Cada contenedor contará con 1 servicio Tomcat o HAProxy, los cuales se afinarán de acuerdo con las recomendaciones de la documentación oficial y a la experiencia del equipo de trabajo

Figura 21. Decisiones arquitecturales

Complementando el modelo c4, se desarrolla el diagrama general de la arquitectura.

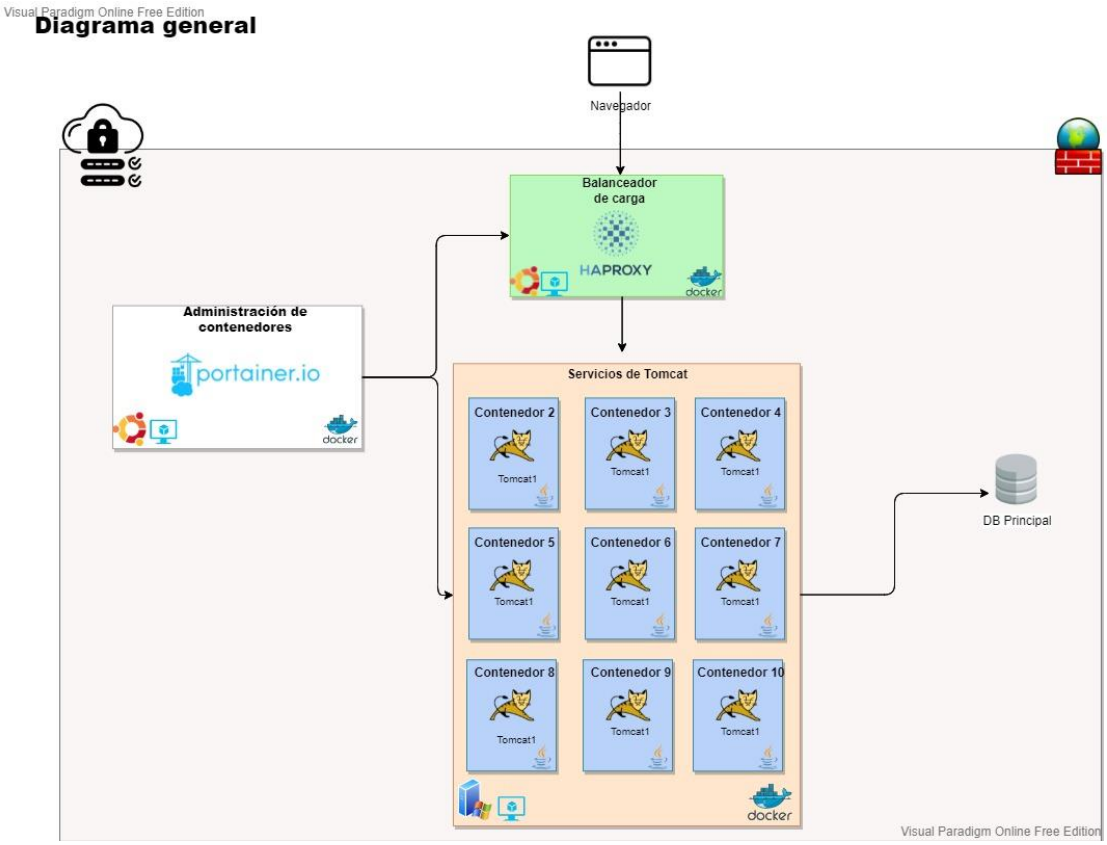


Figura 22. Diagrama general

7. ARQUITECTURA PLANTEADA POR MEDIO DE UN PROTOTIPO

Como se planteó de manera teórica en las anteriores secciones del documento, se trabaja con diferentes componentes de infraestructura, servicios, sistemas operativos, entre otros.
Para sintetizar un poco, se mencionan algunos grandes pasos y en cada uno de ellos se cuenta un poco lo realizado (a nivel general), grandes novedades o experiencias que se consideren relevantes, retos técnicos, cambios en decisiones arquitecturales, entre otros; todo esto teniendo en cuenta que se entregará como anexo un manual con comandos, configuraciones detalladas, referencias, entre otros, que detallará el contenido técnico y las formas como se realizó cada procedimiento de la construcción del prototipo o el demo.

- **¿Dónde se realiza el laboratorio o construcción de arquitectura demo?**
  Se realizó una solicitud formal de dos servidores, uno con sistema operativo Ubuntu 18.04 y otro con Windows Server 2019, al equipo de infraestructura de la Universidad. La solicitud de un servidor Windows server, se realiza teniendo en cuenta que la actual infraestructura de los servicios Tomcat de la Universidad se encuentra alojada en estos sistemas operativos, de manera que se quería realizar la prueba configurando el servicio de Docker en este sistema operativo y validando el funcionamiento de alguna aplicación.
  El servidor Linux se solicita para configurar otros servicios y realizar una validación de la forma como se puede migrar un contenedor Docker entre diferentes sistemas operativos sin afectar el funcionamiento de la aplicación, realizar algunas comparaciones de rendimiento, mostrar la forma como se configura el servicio Docker en Windows y en Linux y mencionar algunas ventajas y desventajas.

- **Contenedores y Docker sobre sistema operativo Windows Server**
  Como se mencionó previamente, se solicitó el servidor de esta forma teniendo en cuenta que la actual infraestructura que aloja estos servicios (Tomcat), se encuentran en este sistema operativo, y se podría dar el caso que algunas aplicaciones war tuvieran alguna ruta o path de Windows seteado y esto generara algunas novedades; adicionalmente, se consideraba también necesario probar este tipo de servicios (Docker) sobre este sistema operativo, teniendo en cuenta de que aún es algo muy nuevo.
  Sobre la instalación del servicio, se debe mencionar que se presentaron diferentes novedades y fue necesario realizar algunas configuraciones adicionales, teniendo en cuenta que los servidores entregados se encuentran virtualizados; de manera que con apoyo del equipo de infraestructura que administra los servidores, fue necesario habilitar algunas opciones en la configuración de la virtualización del servidor. En cuanto a la

instalación, se vivieron diferentes experiencias técnicas sobre las variadas formas en las que se puede instalar este servicio en estos sistemas operativos. Finalmente, luego de varias pruebas (documentadas), se logra dejar funcional el servicio de Docker sobre sistema operativo Windows Server, teniendo como gran conclusión que de igual forma este tipo de servicios emulan un poco el sistema operativo Linux.

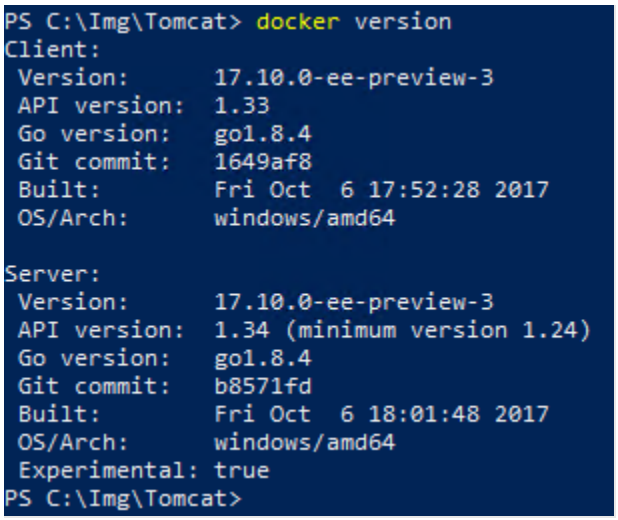


Figura 23. Servicio Docker sobre Windows Server

- **Servicio Tomcat**

  Este servicio fue instalado sobre los contenedores Docker configurados en el servidor Windows Server. Fue necesario sobre este servicio tener en cuenta diferentes aspectos, tales como:

  - Versión del Tomcat (se utiliza la versión 8)
  - Configuración de Dockerfile para la imagen
  - Parametrización de la máquina virtual de Java
  - Despliegue de War (probe) de administración de aplicaciones
  - Despliegue de aplicaciones de prueba entregadas por el equipo de desarrollo de la PUJ
  - Despliegue de múltiples contenedores con diferentes puertos

Los anteriores fueron algunos detalles generales de lo referente a los contenedores que cuentan con el servicio de Tomcat.

Fue necesario validar de forma detallada la documentación del Tomcat, con el fin de realizar una parametrización adecuada del Dockerfile, máquina virtual de Java, archivos war y probe (war de administración de aplicaciones desplegadas en Tomcat).
Referente al despliegue de las aplicaciones de prueba que entregó el equipo de desarrollo, fue necesario solicitarlos previamente, generar permiso a nivel de firewall para llegar a la base de datos de pruebas a donde apuntan las aplicaciones y revisar el string de conexión, teniendo en cuenta que en los primeros intentos de despliegue se presentaron diferentes novedades en estos aspectos.

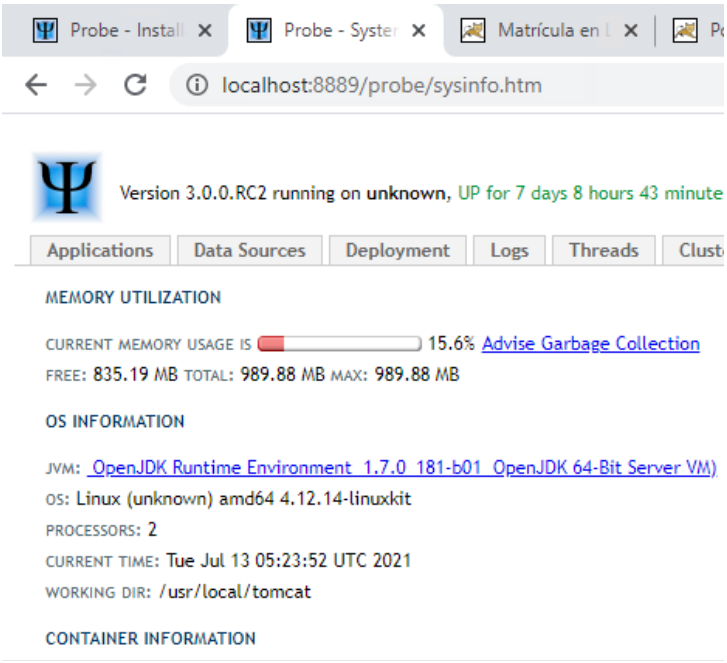


Figura 24. Servicio Tomcat

- **Servicio HAProxy y Docker sobre Linux**
  Como se mencionó, se solicitó también un servidor Linux, con el fin de instalar algunos servicios también sobre este tipo de sistema operativo y validar el funcionamiento del servicio Docker, para documentar esa experiencia en caso de que posteriormente se pueda considerar utilizar solo este sistema operativo para todos los servicios que se alojen en contenedores Docker.
  Con respecto a la instalación fue mucho más práctica que sobre el sistema operativo Windows Server, los detalles se dejan documentados en el anexo A.

Sobre el servicio HAProxy, se presentaron diferentes retos técnicos en la creación de la imagen del servicio, la configuración de este y el despliegue; se debe tener en cuenta que este servicio se instala para balancear el acceso a los servicios Tomcat instalados en los contenedores del servidor Windows Server, de manera que fue necesario realizar diferentes ajustes para los accesos, exposición de puertos, entre otros.

- **Administrador de componentes (Portainer)**

  Este servicio fue revisado de manera detallada a nivel técnico, teniendo en cuenta las diferentes configuraciones necesarias para su funcionamiento. De manera que se revisó la documentación para habilitar el acceso API al servicio Docker en Windows Server y en Linux, proceso en el cual fueron necesarias diferentes pruebas y cuya documentación detallada y experiencias queda alojada en el documento anexo. Luego de configurados esos prerrequisitos, se procede a realizar el despliegue del servicio en un contenedor, sobre el servidor Linux.

  Sobre este servicio se realizaron algunas otras parametrizaciones, como usuario de acceso, conexión con el servidor remoto (Windows) para administrar los contenedores, entre otras configuraciones documentadas.

  Como se comentaba y para concluir, los principales retos pasaron por la forma como se habilitaba el acceso API (puerto 2375) en los dos diferentes sistemas operativos de los servidores de la arquitectura demo.

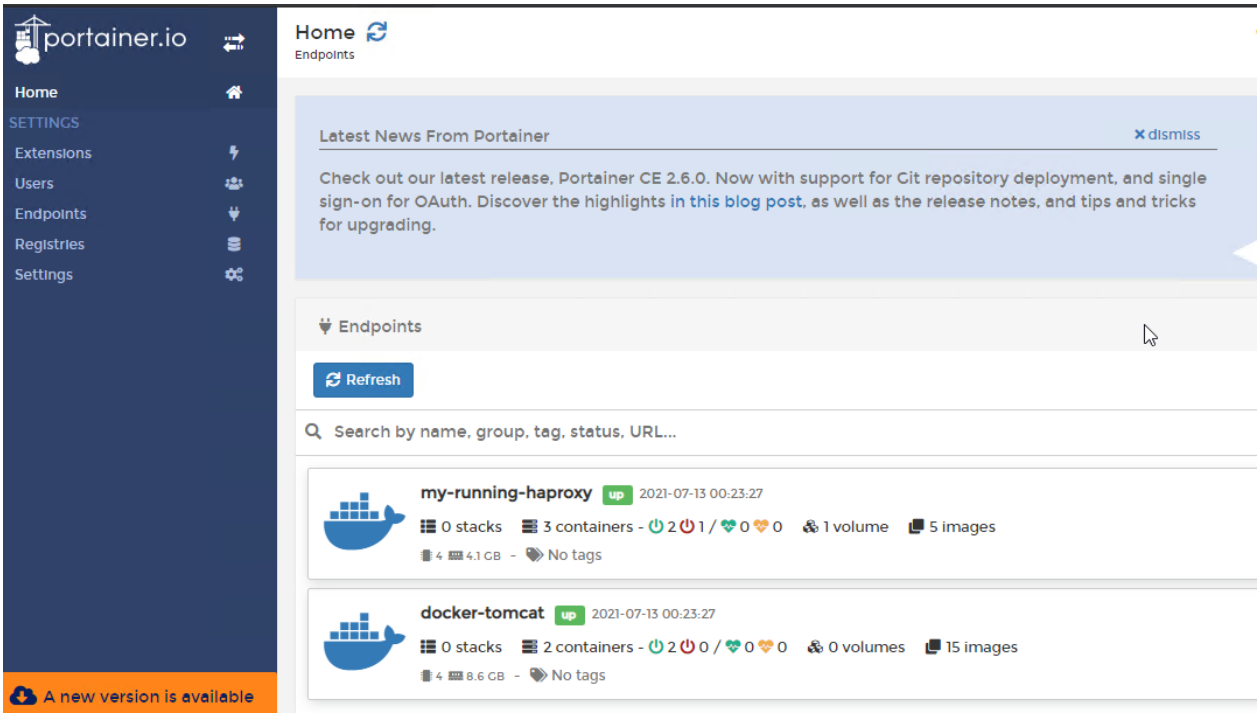


Figura 25. Consola de administración Portainer

7.1 Prototipo y atributos de calidad

En la sección anterior se explican varios detalles del proceso técnico realizado para el diseño de la arquitectura demo y los diferentes retos presentados durante el proceso. Ahora es necesario mencionar las labores técnicas realizadas para abordar algunos atributos de calidad mencionados en la sección del diseño de arquitectura.

- Disponibilidad: como se especificaba en los escenarios de calidad, son muchas las tácticas que se pueden utilizar ante este tipo de atributos. Para el caso de la arquitectura demo, se han creado múltiples contenedores, cada uno con un servicio Tomcat (por buena práctica). Como elemento front, se ha configurado un contenedor independiente con un servicio HAProxy, para que sea el encargado de balancear entre los servicios Tomcat disponibles y así brindar una alta disponibilidad en el servicio a nivel general.
- Portabilidad: una de las ventajas que se utiliza en una arquitectura de contenedores, tiene que ver justo con los temas de portabilidad. Por tal razón, en la arquitectura se plantea que los contenedores desplegados en el servidor Windows Server para el escenario demo, pueden ser portados a otro servidor (el servidor Linux para nuestro caso). Se realiza una prueba y se deja documentada en el informe detallado adjunto, indicando con esto la forma como un contenedor desplegado sobre un servicio Docker alojado en Windows Server, puede ser llevado o desplegado en un servidor Linux. Como se mencionaba, para

lo anterior se utilizaron los dos servidores entregados, donde se alojó todo lo referente a la arquitectura demo.

- Usabilidad: para atender todo lo referente a este atributo de calidad, se configura la herramienta Portainer, la cual permite hacer toda la gestión de los contenedores de toda la arquitectura planeada. La Figura 25 es una referencia de la herramienta
- Seguridad: siguiendo las recomendaciones que brindó la Universidad, se conserva la actual versión que utilizan de Tomcat (8) y la del Java (7). Adicionalmente, se contempla continuar el uso de la herramienta de doble factor llamada OneGate, la cuál es una capa expuesta de doble factor de autenticación, que entre sus ventajas incluye esa doble seguridad en temas de ingreso a las diferentes aplicaciones o sistemas de información
- Performance: aunque este atributo se debe validar en un escenario productivo donde se cuente con todos los recursos para estos ambientes, en la sección de revisión de las pruebas de desempeño realizadas, se detallan los resultados y se explica que el actual servidor Windows Server, donde se trabajaron las pruebas, si consume un poco más del porcentaje indicado, toda vez que fue utilizado para el laboratorio como punto de acceso a la infraestructura disponible por la Universidad. Sin embargo, para un futuro servidor con solamente los servicios Docker operando, se dejan algunas configuraciones útiles para estos escenarios (consumo de memoria del Java ajustado, un servicio por cada contenedor, entre otros). Algunos otros detalles son especificados en la comparación y análisis de la siguiente sección.

7.2 Documentación del prototipo

La documentación técnica de las actividades realizadas con el prototipo se deja en el Anexo A, el cual brinda toda la información técnica necesaria de forma detallada.

8. DESEMPEÑO DE LA ARQUITECTURA PROTOTIPO EN CONTENEDORES DOCKER VS LA ARQUITECTURA CON VIRTUALIZACIÓN TRADICIONAL

8.1 Comparación y análisis

Como se mencionó en capítulos anteriores, se realizó una arquitectura prototipo o demo sobre dos servidores entregados por la Universidad Javeriana para realizar este proyecto.

Para realizar todo lo referente a pruebas y validaciones de la arquitectura, se ejecutaron diferentes pruebas funcionales y no funcionales:

- Se plantean casos de pruebas (manuales) tales como
  - Verificación del despliegue del servicio tomcat
  - Carga de aplicación war demo
  - Revisión de balanceo de aplicaciones por medio del HAProxy
  - Funcionamiento del Portainer
  - Entre otras (detalladas en el documento anexo de pruebas manuales)
- Se plantean pruebas no funcionales de rendimiento
  - Se realizan dos pruebas con diferentes características
  - Se utiliza la herramienta jmeter y la documentación de estas y evidencias técnicas queda alojada en el anexo C.

De acuerdo con las pruebas detalladas, se plantean diferentes conclusiones de la arquitectura planteada:

Pruebas funcionales:

- Todo lo realizado o planteado en el documento técnico tuvo retos en diferentes aspectos, los cuales fueron superados y documentados en el documento adjunto ya especificado. En el momento de realizar esas validaciones manuales, se confirmó un correcto funcionamiento de cada punto especificado (despliegue de aplicación demo, uso del HAProxy, Portainer, etc.). Cada herramienta o esquema detallado cumplía con su correcto funcionamiento y permitía completar todo lo referente a una aplicación demo funcionando bajo una arquitectura de contenedores Docker.
  Es importante mencionar o destacar, que la arquitectura con el servicio Tomcat fue configurada sobre Windows Server; y tal como se mencionaba en la sección del prototipo, esto trajo retos técnicos, teniendo en cuenta lo nuevo que puede ser el servicio Docker

para servidores Windows Server. Sin embargo, en las pruebas realizadas, se comprueba un correcto funcionamiento del Docker, Tomcat y la aplicación demo.

Pruebas de carga:

- Como se detalló en otro párrafo, para lo referente a estas pruebas se utiliza la herramienta JMeter y se solicita el acompañamiento de una persona del equipo de infraestructura de la Universidad, con el fin de poder obtener información del rendimiento de los servidores directamente del virtualizador. Aunque las pruebas de carga fueron 2 y se presentaron diferentes novedades debido a que el JMeter se debía configurar en el mismo servidor donde están los contenedores con el servicio Tomcat (comparten recursos), lo cual generó que en la última prueba el JMeter genera un bloqueo como aplicación propia. Sin embargo, con los datos realizados se podía evidenciar que la prueba realizada operaba sin novedad 200 usuarios en 200 segundos y el rendimiento de los servicios Tomcat no se elevó (manteniéndose en promedio en 700mb de memoria ram).
  Aunque se quería realizar una prueba superior, por temas de permisos no se contaba con otro equipo desde donde se pudiera realizar la prueba de carga. Es importante tener en cuenta, que este tipo de pruebas con JMeter, dependen mucho del equipo origen desde donde se ejecuten las mismas, por eso cuando se ha realizado esta actividad, fueron utilizados varios equipos (con acceso al servicio), y ejecutado la misma prueba en todos, con el fin de jugar un poco con los recursos propios de cada máquina.
  Sin embargo, lo anterior deja unos resultados sobresalientes, teniendo en cuenta aspectos como:

  - Solo se configuraron 4 contenedores Docker desplegados con el servicio Tomcat, ya que como se compartían recursos con JMeter y aparte no es un servidor con las mismas características de uno de producción habitual (más RAM, más CPU). Por tal motivo, la prueba fue controlada por este aspecto
  - El consumo de los 4 servicios desplegados no superó el valor de promedio de 700mb en memoria RAM
  - No hubo reportes de indisponibilidad reportados en los logs de VMWare
  - El HAProxy realizó un correcto balanceo de carga
  - La CPU del balanceador de carga tuvo solo picos de CPU alta
  - El servidor Windows server generó algunos picos en consumo de red
  - A nivel de lectura y escritura en disco sobre el servidor Windows Server, se generó una latencia habitual al promedio

Comparando un poco los resultados con el estado habitual de la arquitectura de los servicios Tomcat de la Universidad, se puede mencionar lo siguiente:

- Como se planteó en la sección del estado actual, los servicios Tomcat habitualmente se encuentran en un promedio entre 900 MB y 1 GB de consumo de memoria RAM; lo que a priori puede mostrar poca diferencia en consumo versus lo planteado. Sin embargo, es necesario tener en cuenta que estos servicios Tomcat de la Universidad Javeriana, son reiniciados por una tarea programada en una hora determinada de la noche y se realiza una limpieza de logs, lo que permite que el consumo de memoria pueda ser diferente a la arquitectura planteada (contenedores desplegados desde hace varias semanas)
- Teniendo en cuenta el item anterior, es necesario mencionar que los servicios Tomcat fueron desplegados bajo contenedores Docker (con sistema operativo base Linux), pero el sistema operativo del servidor principal es Windows Server 2019; lo anterior se realiza acordado con el equipo de infraestructura de la Universidad, acatando recomendaciones de funcionamiento. Sin embargo, se deja constancia de que estos contenedores Docker pueden ser migrados a un servidor que por naturaleza consuma menos recursos (alguna distribución de Linux), ya que la portabilidad de los contenedores es una de las grandes ventajas de este servicio. Es necesario mencionar, que se debe realizar una prueba detallada de funcionamiento sobre un sistema operativo Linux como servidor principal del servicio Docker, ya que, en las pruebas realizadas de este tema, fueron para atributos de portabilidad expuestos a lo largo del documento.
- Aunque el servidor Windows cuenta (a diferencia de uno habitual de la Universidad) con el servicio Docker, no fue un valor relevante en el consumo de recursos; sin embargo, si hubo otros factores que podrían generar aumento, tales como: Jmeter y Chrome. Lo anterior fue utilizado durante las pruebas, teniendo en cuenta que era la máquina que tiene privilegios para ejecutar las validaciones necesarias
- Con respecto al consumo de servicios como el HAProxy, habitualmente este componente no tiene un desgaste alto, ya que sus actividades se centran en realizar la distribución de la carga. En este tipo de componentes se debe tener en cuenta son las parametrizaciones de máximas conexiones, temas de caché, entre otros
- Otro punto por mencionar es que se tuvo acceso a la evidencia técnica de las pruebas y a la información de experiencias vividas en estos aspectos del equipo de infraestructura, referente a pruebas que se han realizado del servicio del War demo utilizado. Sobre este punto se encontraron algunos detalles de pruebas realizadas, con algunos resultados y condiciones como:
    - Ejecutado desde diferentes equipos de alto rendimiento de una sala de computo

- Un caso de prueba integro (autenticación, consulta y escritura)
- Logran emular un número máximo de conexiones hasta llevar la CPU de la base de datos al 100%
- Es necesario tener en cuenta que no se conoce la cantidad de hilos, periodo de subida y repeticiones de las pruebas que habitualmente ejecutan.

Anexo al presente escrito, se encuentra toda la documentación referente a las pruebas realizadas, manuales técnicos de herramientas de pruebas utilizadas e informes detallados que se pueden detallar.

9. CONCLUSIONES

La etapa de la definición del problema y el análisis del estado actual permitió recopilar información fundamental para identificar de forma detallada los puntos críticos a abordar durante el desarrollo del proyecto. Adicionalmente, toda esta información inicial determinó la base para tomar las diferentes decisiones arquitecturales del diseño de la solución.

Cuando se comprende de manera integral el estado actual de los componentes que se abordan en el proyecto, es claro que este tipo de soluciones son prototipos para iniciar en todo lo referente a un cambio de arquitectura, tratando de dar pasos para llegar a temas de DevOps, alejándose de monolitos, con cambios a nivel general desde puntos particulares del diseño de la arquitectura; y justo era eso lo que generaba mucha expectativa, ya que, al contar con una aplicación ya desarrollada y funcional en una arquitectura virtualizada, no era claro el funcionamiento de la misma en contenedores Docker. Adicionalmente, los componentes funcionaban bajo un sistema operativo Windows Server, y aunque se utilizó ese tipo de sistema operativo para alojar el servicio, los contenedores nativamente cuentan con sistemas operativos de base Linux y eso podría generar algunas novedades.

Otro aspecto considerado como un buen reto a nivel técnico y de diseño, fue lo referente a la configuración del servicio Docker sobre sistemas operativos Windows Server, teniendo en cuenta que habitualmente estos componentes son utilizados o configurados bajo sistemas operativos Linux; aunque para Windows Server es oficial el uso del servicio Docker, no hay tanta experiencia a nivel de documentación, como si lo hay para sistemas operativos Linux. Esta dinámica fue realizada para conservar parte de la base de la actual infraestructura, aunque se ha dejado como recomendación que es ideal utilizar los servicios bajo sistemas operativos Linux y la migración o portabilidad del prototipo no sería un proceso extenso.

El servicio para balancear carga y el administrador de los contenedores, fueron otros componentes que generaron retos a nivel técnico, toda vez que no dejaba de ser un prototipo que se estaba trabajando y muchos aspectos eran conocidos, pero otros eran trabajos exploratorios que se quería validar su respectivo funcionamiento.

A nivel de documentación, se habían planteado los diagramas del modelo c4, sin embargo, durante el desarrollo técnico fue necesario ajustar algunos detalles que se observaron en esta parte práctica; por tal razón, se dejan documentadas dos versiones de cada nivel del modelo, explicando las diferencias o los cambios que hay en cada uno de ellos.
Es importante mencionar, que durante toda la etapa práctica se fueron ajustando algunos detalles teóricos o procedimentales, cuyo conocimiento lo iban dando los retos que se enfrentaban y la documentación asociada; se debe detallar que el dialogo con expertos, la experiencia laboral y académica, fueron puntos claves que permitieron realizar una toma de decisiones adecuada ante cada situación que se iba presentando.

10. TRABAJOS FUTUROS

Este tipo de arquitecturas planteada cuenta con muchas posibilidades de crecimiento en diferentes aspectos. Se mencionan a continuación, algunos trabajos futuros que se consideran se pueden ejecutar.

- Orquestador: al contar con una arquitectura de contenedores Docker que pueda estar en constante crecimiento, y la misma por la cantidad de objetos involucrados tome una carga operativa alta, es posible considerar añadir a la arquitectura componentes para orquestar los contenedores que pertenezcan a la infraestructura; durante varios puntos del documento, se menciona este aspecto como un posible elemento que permita continuar por el camino de la cultura DevOps. Sobre este aspecto de orquestación, se pueden mencionar diferentes herramientas como Kubernetes, Docker Swarm, que a medida de irlas implementando permitirán el crecimiento completándose con otras como Rancher, generando esto un crecimiento a nivel de arquitectura
- Migración de servicio Docker a Linux: durante el documento se mencionaba que es posible migrar los contenedores que se alojan sobre el servicio Docker, los cuales están en el servidor Windows Server; esta recomendación es ideal para bajar temas de consumo de recursos, como se lograba visualizar con el uso del contenedor Docker sobre Linux con el servicio de HAProxy.
- Arquitectura de microservicios: durante la sección de las arquitecturas de referencia, se mencionaba que, para el prototipo del presente documento se planteaba una arquitectura que utiliza como referencia la de microservicios; sin embargo, el mismo solamente se utiliza como base, toda vez de que hay muchos ítems de un diseño de microservicios que no se aplican por temas de alcance del proyecto y estado actual de la arquitectura. De manera que solo se utiliza o se hace referencia a la arquitectura de microservicios, por el uso de componentes Docker.

  Como trabajo futuro la arquitectura planteada permite utilizar de forma integral un diseño de microservicios, trabajando los cambios desde un enfoque de desarrollo sobre los componentes de software de la organización
  - La anterior división a nivel de componentes de software incluye la separación de los componentes de base de datos, teniendo en cuenta el estado actual de los diferentes sistemas de información en este aspecto
- Migración a nube: aunque bajo los contextos actuales de la organización este no es tema que se tenga a corto plazo en los servicios core, se deja referencia de que la arquitectura planteada es posible portarla a los equivalentes en nube vs on premise.

- Acceso a los servicios: como se planteaba en la arquitectura, el acceso a los servicios se realiza por medio de un balanceador; es posible que se pueda agregar el uso de una ip virtual, con el fin de contar con varios balanceadores en caso de querer subir a otro nivel en el tema de disponibilidad en este aspecto

11. BIBLIOGRAFÍA

[1] Red+ Noticias, "¿Qué son las TIC? Y ¿Por qué son tan importantes?," *Claro*, 2019. https://www.claro.com.co/institucional/que-son-las-tic/.

[2] H. Cervantes, "Arquitectura de Software," *Software Guru*.

[3] W. Ccori, "Los 10 patrones comunes de arquitectura de software," *Medium*, 2018. https://medium.com/@maniakhitoccori/los-10-patrones-comunes-de-arquitectura-de-software-d8b9047edf0b.

[4] "¿Qué es DevOps?," *AZURE*. https://azure.microsoft.com/es-es/overview/what-is-devops/.

[5] I. Gorton, *Essential software architecture*, vol. 49, no. 05. 2012.

[6] Google Cloud, "Google Cloud," *CONTENEDORES EN GOOGLE*, 2020. https://cloud.google.com/containers/?hl=es-419.

[7] itTrends, "itTrends," *El mercado de orquestación de contenedores crecerá hasta 2026*, 2019. https://www.ittrends.es/software-y-apps/2019/12/el-mercado-de-orquestacion-de-contenedores-crecera-hasta-2026.

[8] D. Bryant, C. Swan, S. Opel, H. Beal, and M. Pais, "DevOps and Cloud InfoQ Trends Report - February 2019," *Febrero 2019*, 2019. https://www.infoq.com/articles/devops-cloud-trends-2019/?itm_source=articles_about_InfoQ-trends-report&itm_medium=link&itm_campaign=InfoQ-trends-report.

[9] J. P. Paredes, "Alta disponibilidad para Linux," pp. 1–14, 2019.

[10] A. Glover and K. Probst, "Tips for High Availability," *Netflix Technology Blog*, 2018. https://medium.com/@NetflixTechBlog/tips-for-high-availability-be0472f2599c.

[11] "¿Qué es la implementación azul-verde?," *Red Hat*, 2018. https://www.redhat.com/es/topics/devops/what-is-blue-green-deployment.

[12] D. Ernst, A. Becker, and S. Tai, "Rapid Canary Assessment Through Proxying and Two-Stage Load Balancing," *Proc. - 2019 IEEE Int. Conf. Softw. Archit. - Companion, ICSA-C 2019*, pp. 116–122, 2019, doi: 10.1109/ICSA-C.2019.00028.

[13] M. Graff and C. Sanden, "Automated Canary Analysis at Netflix with Kayenta," *THE NETFLIX TECH BLOG*, 2018. https://netflixtechblog.com/automated-canary-analysis-at-netflix-with-kayenta-3260bc7acc69.

[14] B. Fekade, T. Maksymyuk, and M. Jo, "Clustering hypervisors to minimize failures in mobile cloud computing," *Wirel. Commun. Mob. Comput.*, vol. 16, no. 18, pp. 3455–3465, 2016, doi: 10.1002/wcm.2770.

[15] "Configuración de la infraestructura de alta disponibilidad," *IBM Knowledge Center*. https://www.ibm.com/support/knowledgecenter/es/ssw_ibm_i_73/rzaig/rzaigtaskconfig.htm.

[16] K. Inamdar, G. Salgueiro, R. M. Ravindranath, and S. Jeuk, "CANARY RELEASE VALIDATION MECHANISMS FOR A CONTAINERIZED APPLICATION OR SERVICE MESH," 2020.

[17] "What is a Container?," *Docker*, 2020. https://www.docker.com/resources/what-container.

[18] "¿Qué es DOCKER?," *Red Hat*. https://www.redhat.com/es/topics/containers/what-is-docker.

[19] D. Merkel, "Docker : Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, pp. 2–7, 2014, [Online]. Available: http://delivery.acm.org.ezproxy.library.wisc.edu/10.1145/2610000/2600241/11600.html?ip=128.104.46.196&id=2600241&acc=ACTIVE SERVICE&key=066E7B0AFE2DCD37.066E7B0AFE2DCD37.4D4702B0C3E38B35.4D4702B0C3E38B35&__acm__=1557803890_216b4a0168a6b29b8f2e7a74.

[20] "¿Qué es una máquina virtual?," *Red Hat*, 2021. https://www.redhat.com/es/topics/virtualization/what-is-a-virtual-machine.

[21] "Los contenedores en comparación con las máquinas virtuales," *Red Hat*, 2021. .

[22] A. Pereira Ferreira and R. Sinnott, "A performance evaluation of containers running on managed kubernetes services," *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2019-Decem, pp. 199–208, 2019, doi: 10.1109/CloudCom.2019.00038.

[23] Y. Huang, R. Zong, K. Cai, and Y. Mao, "Design and implementation of an edge computing platform architecture using docker and kubernetes for machine learning," *ACM Int. Conf. Proceeding Ser.*, pp. 29–32, 2019, doi: 10.1145/3318265.3318288.

[24] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker Containers across Multiple Clouds and Data Centers," *Proc. - 2015 IEEE/ACM 8th Int. Conf. Util. Cloud Comput. UCC 2015*, pp. 368–371, 2015, doi: 10.1109/UCC.2015.58.

[25] R. Onet, R. Burian, C. M. Campeanu, I. A. Ivanciu, D. Zinca, and V. Dobrota, "Automatic Deployment of a Network Overlay in an Intelligent Transportation System: Docker and Open Baton Approach," *Proc. - RoEduNet IEEE Int. Conf.*, vol. 2019-Octob, 2019, doi: 10.1109/ROEDUNET.2019.8909588.

[26] The Apache Software Foundation, "APACHE HTTP SERVER PROJECT." https://httpd.apache.org/.

[27] "Ciberseguridad total," *Portainer: gestión sencilla de Docker a través de la web*, 2020. https://ciberseguridadtotal.com/portainer-gestion-sencilla-de-docker-a-traves-de-la-web/.

[28] "¿Qué es Kubernetes?," *Kubernetes*, 2020. https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/.

[29] T. Granollers, "¿Qué es un prototipo?," *Curso de Interacción Persona-Ordenador*. https://mpiua.invid.udl.cat/fases-mpiua/prototipado/que-es-un-prototipo/.

[30] The Apache Software Foundation, "Apache Tomcat." http://tomcat.apache.org/index.html.

[31] H. Cervantes and R. Kazman, "Architectural Design," 2016, [Online]. Available: https://www.informit.com/articles/article.aspx?p=2738304&seqNum=2.

[32] "steakrecords," *¿Pasar argumentos de JVM a Tomcat cuando se ejecuta como un servicio?* https://steakrecords.com/es/426098-passing-jvm-arguments-to-tomcat-when-running-as-a-service-java-environment-variables-jvm-arguments-tomcat.html.

[33] J. B. Karl Wiegers, "First thing first. Setting requirements priorities. Wiegers," in *Software Requirements.*, Microsoft Press, 2013.

[34] Microsoft, “Estilo de arquitectura de microservicios.” https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices.
[35] M. Goig, “Arquitectura de Docker.” https://docs.mikelgoig.com/docker/arquitectura-de-docker.html.
[36] “ISO/IEC 25010.” https://iso25000.com/index.php/normas-iso-25000/iso-25010?limit=3&limitstart=0.
[37] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice Second Edition Third Edition*. 2013.

11. Anexos

Anexo A. Manual de configuración e instalación

El presente anexo cuenta técnicamente los procesos realizados para instalar y configurar los diferentes componentes del prototipo. Adicionalmente, se cuentan las diferentes opciones que se exploraron en casos donde se generaban novedades en las pruebas o validaciones iniciales.

Recursos

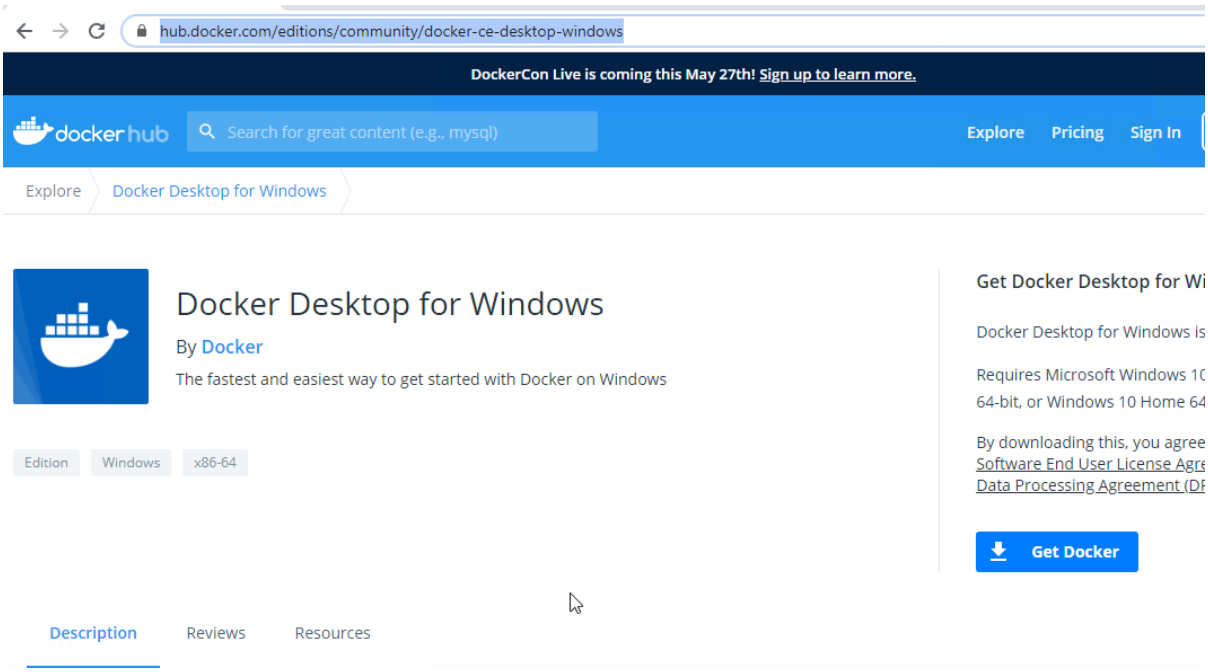
Servidor Windows

Servidor Linux

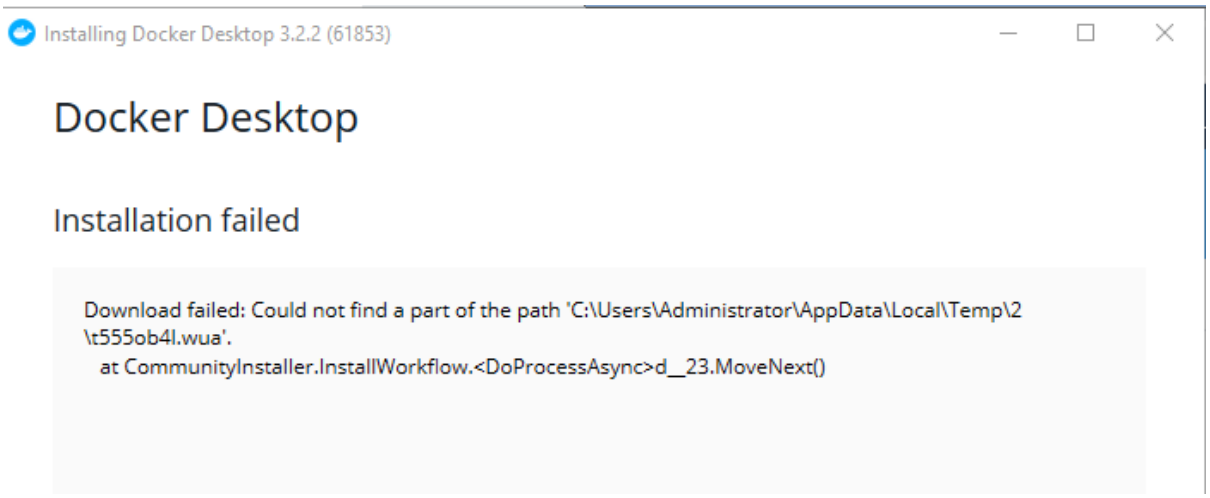
Docker

- Se debe descargar instalador en la siguiente ruta:

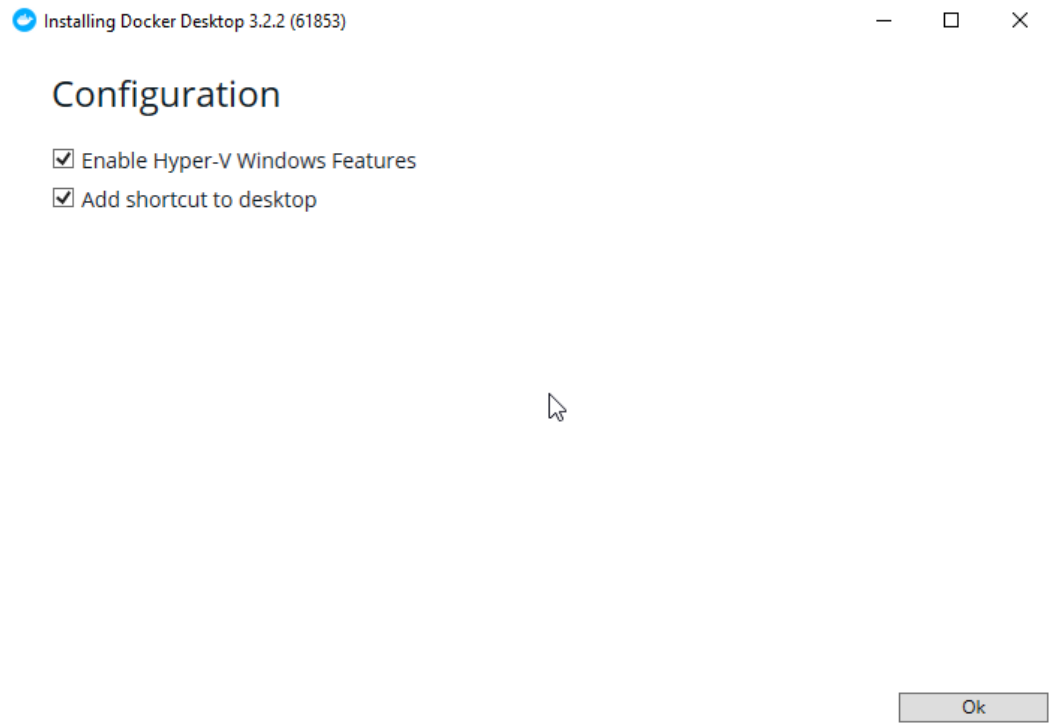
https://hub.docker.com/editions/community/docker-ce-desktop-windows



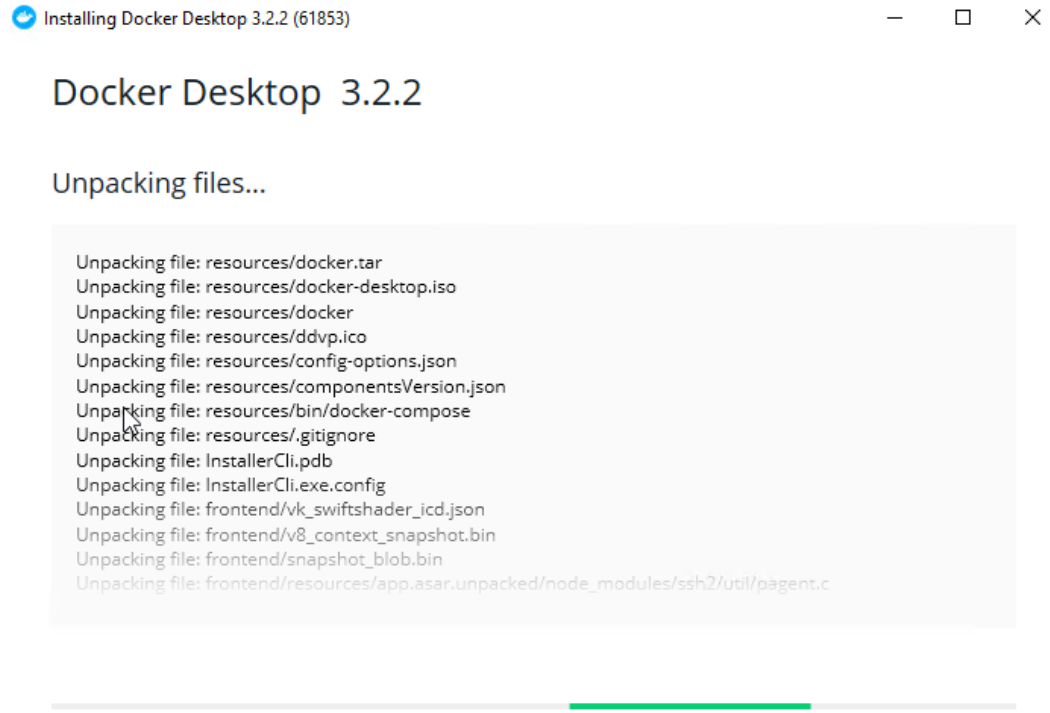
- Ejecuta el archivo descargado
- Ante el posible error:



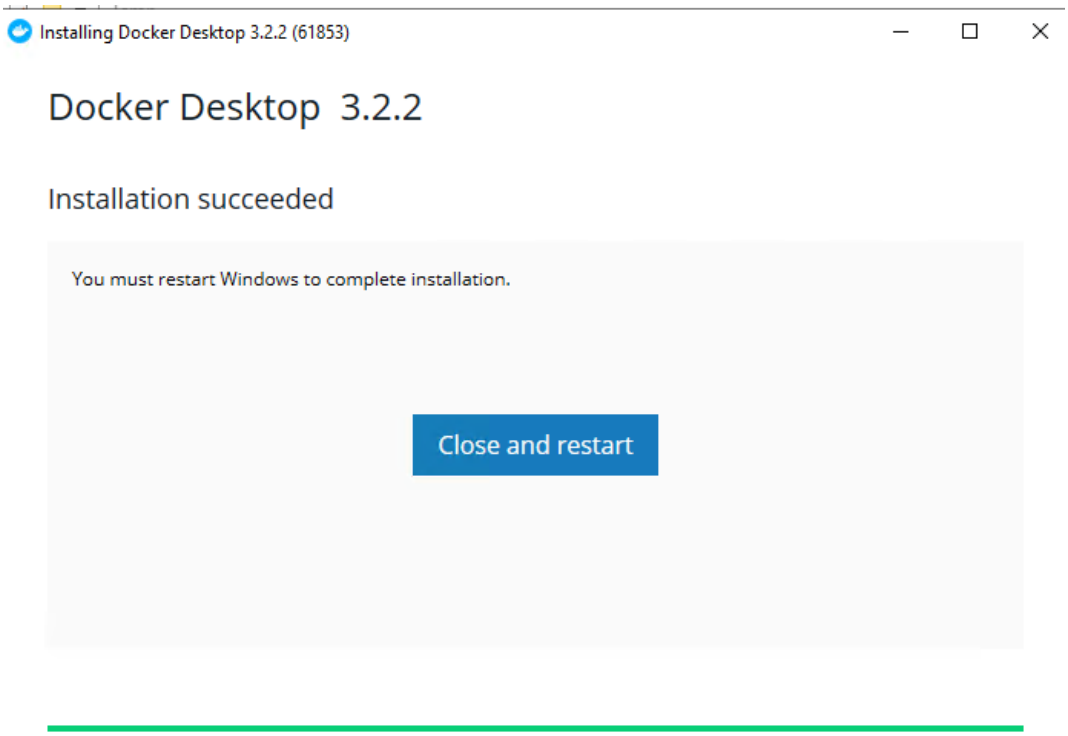
  - se debe borrar registros de la ruta indicada, para este caso fue: C:\Users\Administrator\AppData\Local\Temp
  - Borrar carpeta 2
- Ok al siguiente mensaje



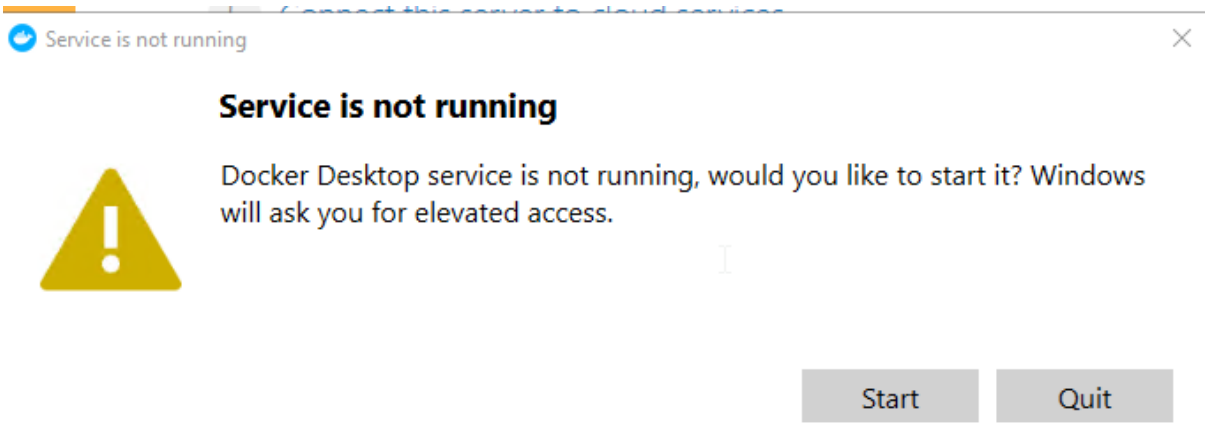
- Instalando



- Finaliza el proceso



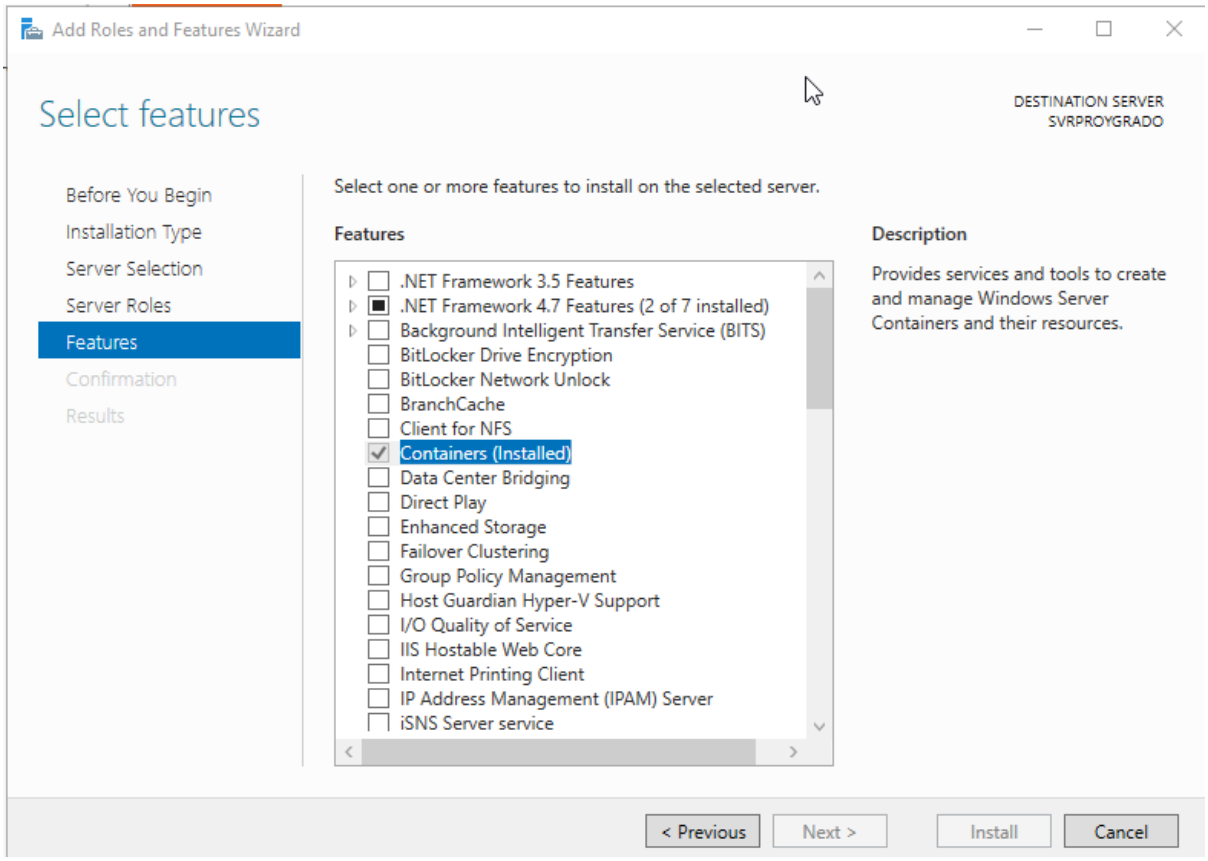
- Se debe reiniciar el sistema
- Posteriormente, ante el siguiente mensaje hacer clic en Start



NOTA: como la máquina virtual se encuentra sobre VMware, se puede probar otra instalación o ajustar temas de VMware.

Forma 2 de instalar Docker

- Activar característica



- Abrir PowerShell como administrar
- Luego validar con el comando

Fin-module –name docker*

```
PS C:\Users\Administrator> find-module -name docker*

Version    Name                                Repository           Description
-------    ----                                ----------           -----------
1.0.0.8    DockerMsftProvider                  PSGallery            PowerShell module with commands for discovering,...
0.0.0.3    DockerProvider                      PSGallery            PowerShell module with commands for discovering,...
0.0.3      dockeraccesshelper                  PSGallery            Allow a user account to access the docker engine...
1.3.2      Docker                              PSGallery            This module helps with development using Docker ...
1.2010.... DockerCompletion                    PSGallery            Docker command completion for PowerShell.
1.0.0.2    DockerMsftProviderInsider           PSGallery            PowerShell module with commands for discovering,...
0.7.2      Docker-CI                           PSGallery            PowerShell module to build, test and publish doc...
0.0.0.2    DockerDsc                           PSGallery            This module contains resources to deal with Dock...
1.29.0.... DockerComposeCompletion             PSGallery            Docker Compose command completion for PowerShell.
1.0.154    DockerPowershell                    PSGallery            Adds cmdlets to work with the Docker cli.
0.3.3      Docker-Ops                          PSGallery            Some utils for docker operations
0.4.8      Docker.Build                        PSGallery            PowerShell module to build, test and publish doc...
0.16.2.... DockerMachineCompletion             PSGallery            Docker Machine command completion for PowerShell.
0.11.54880 DockerStack                         PSGallery            Module to serve the purpose of interfacing with ...
0.7.7.14   DockerRegistry                      PSGallery            Cmdlets for talking to Docker Engine and Docker ...
0.9.0      DockerMachine                       PSGallery            Create new or load existing Docker Machine config
0.1.3      DockerHelpers                       PSGallery            Cmdlets to extend and enhance the docker cli
0.0.2      DockerInstall                       PSGallery            Powershell DSC Resource to install Docker for Wi...
```

- Instalar DockerMsftProvider de PSGallery

Install-Module -Name DockerMsftProvider -Repository PSGallery -Force

```
PS C:\Users\Administrator> install-module -name  DockerMsftProvider -Repository PSGallery -force
```

- Luego instalar el paquete

  Install-Package -Name docker -ProviderName DockerMsftProvider

```
PS C:\Users\Administrator> install-Package -Name docker -ProviderName DockerMsftProvider

The package(s) come(s) from a package source that is not marked as trusted.
Are you sure you want to install software from 'DockerDefault'?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y

Name                           Version          Source           Summary
----                           -------          ------           -------
Docker                         20.10.0          DockerDefault    Contains Docker EE for use with Windows Server.


PS C:\Users\Administrator> docker
```

- En ocasiones es necesario reiniciar
- Se valida versión

```
PS C:\Users\Administrator> docker --version
Docker version 20.10.0, build 6ee42dc
PS C:\Users\Administrator>
```

- Se debe iniciar el servicio

  Start-Service docker

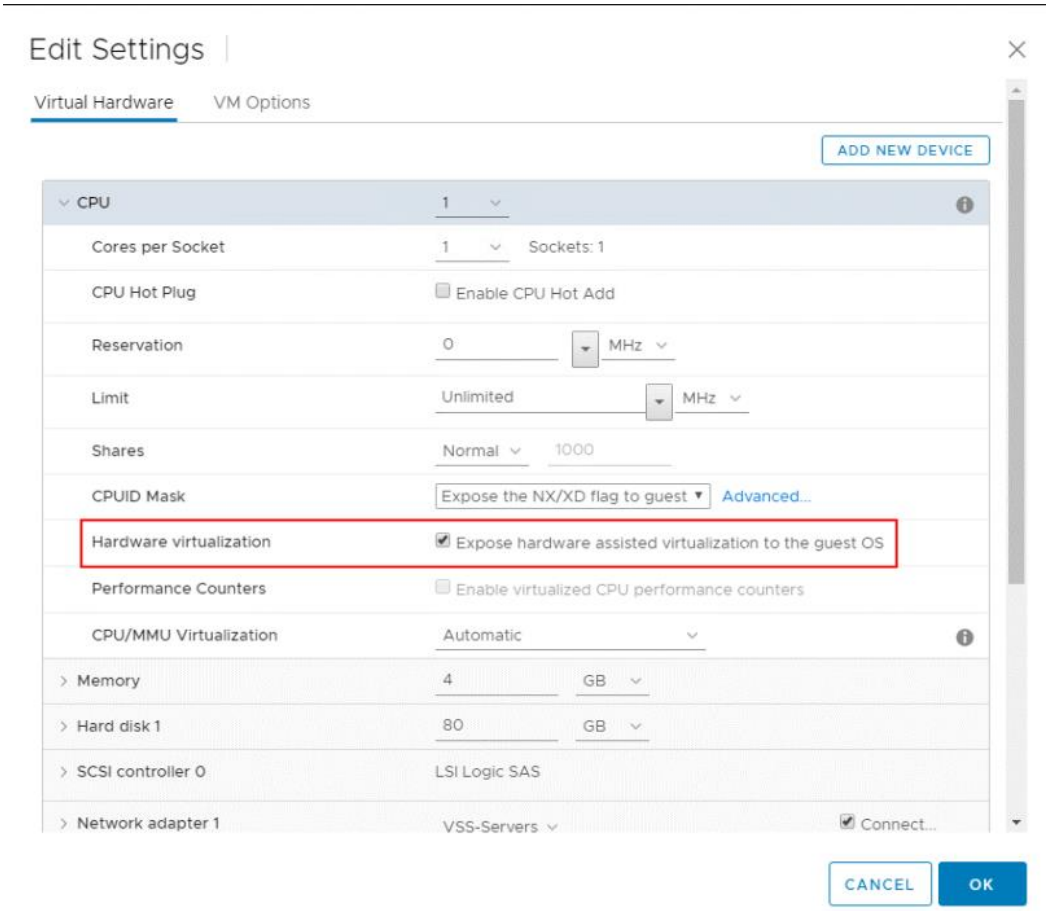
- Se validar servicio ok

```
 Experimental:      false
PS C:\Users\Administrator> Get-Service docker

Status   Name               DisplayName
------   ----               -----------
Running  docker             Docker Engine
```

Novedades

Al realizar la instalación anterior, funciona todo ok, sin embargo, cuando se quiere instalar una imagen en Linux, es necesario realizar unos ajustes en la configuración. Para el laboratorio, tenemos una máquina virtual en VMWare, por lo cual fue necesario solicitar a los administradores de la plataforma modificar unas configuraciones

- Modificar virtualización invitado OS para VMware.



- Desinstalar lo anterior

Uninstall-Package -Name docker -ProviderName DockerMSFTProvider

- Habilite la virtualización anidada si está ejecutando contenedores Docker con una máquina virtual Linux que se ejecuta en Hyper-V.

Get-VM WinContainerHost | Set-VMProcessor -ExposeVirtualizationExtensions $true

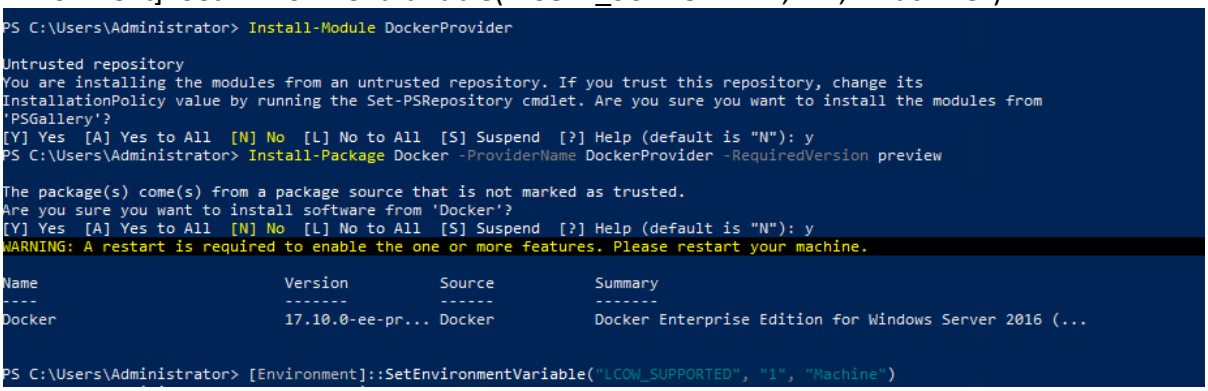
- instale la versión de vista previa actual de Docker EE.

Install-Module DockerProvider

Install-Package Docker -ProviderName DockerProvider -RequiredVersion preview

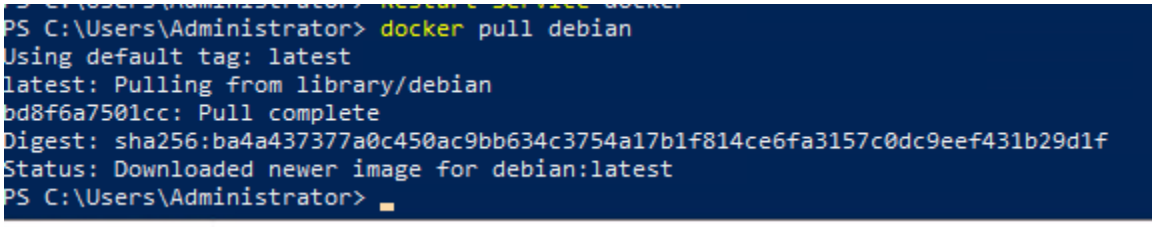
- Habilite el sistema LinuxKit para ejecutar contenedores Linux

[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", "1", "Machine")



- Reiniciar servicio

  Restart-Service docker
- Se prueba bajando img docker



Nota: en caso de querer hacer switch para instalar imágenes Windows, se debe utilizar el comando
[Environment]::SetEnvironmentVariable("LCOW_SUPPORTED", "$null", "Machine")

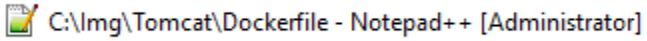
Dockerfile

- Se construye Dockerfile
- Ruta de tomcat que se utilizará

https://hub.docker.com/layers/tomcat/library/tomcat/8.0/images/sha256-3c45e165dc72e3fc0f147dfa0c4712145cde00c2efc78d6df50ca33437542079?context=explore

Imagen Docker

- Se construye imagen creando previamente un Docker file, el cual queda alojado en la ruta:



- La imagen incluye temas como:
  - Paso del probe.war
    - El probe es un gestor muy utilizado por la organización para realizar despliegues
  - Es necesario reemplazar el archivo server.xml, para ajustar el tema de puertos
    - Tienes algunos cambios como:
      - Puerto, definir 8080, 8090, 8100 o 8110, según corresponda

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

      - Modificar puerto AJP 8009,8010,8011,8012,8013

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
  <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

  - Reemplazar archivo tomcat-users.xml, modificar los datos para acceso desde el probe

```
<user username="admin" password="Ponti2021" roles="manager-gui" />
```

- Se debe añadir una línea que copie los parámetros del tomcat-users, con el fin de reemplazar el archivo con uno que indique los datos de acceso con el probe

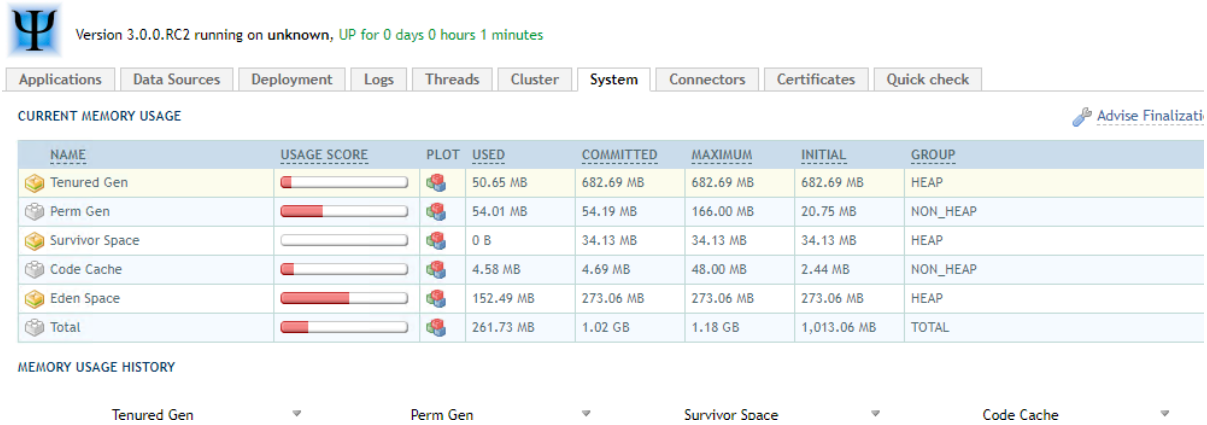
copy tomcat-users.xml /usr/local/tomcat/conf

Ese archivo tomcat-users.xml cuenta con el usuario para acceder al probe

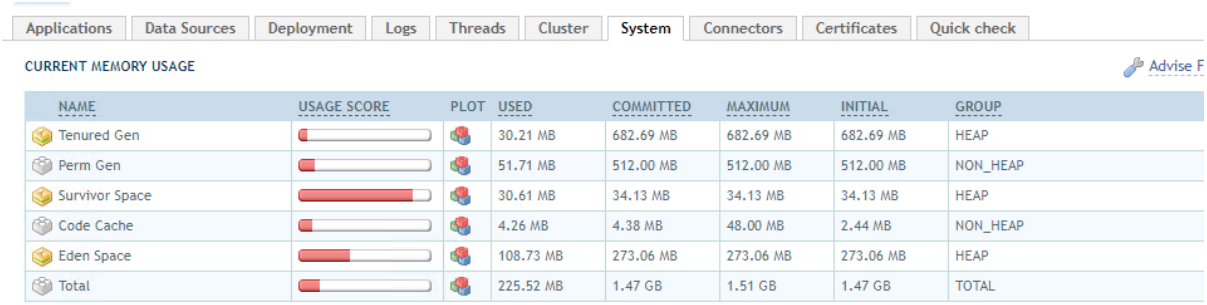
<user username="admin" password="Ponti2021" roles="manager-gui" />

- Configuración de la MV de java y parámetros en la imagen docker

Se realiza una prueba parametrizando
ENV JAVA_OPTS="-XX:PermSize=512m -XX:MaxPermSize=512m"
Antes Perm Gen:



Luego del parámetro en el Docker file



- Se puede ajustar archivo server.xml, con el fin de parametrizar puerto de apagado, puerto interno del tomcat y puerto ajp
    - Puerto apagado

<Server port="8005" shutdown="SHUTDOWN">
    - Puerto interno

<Connector port="8080" protocol="HTTP/1.1"
    - Puerto ajp

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

Para aplicar lo anterior, se debe configurar un archivo server.xml, el cual se añade a la imagen para que copie el fichero cada que se compile la imagen. Aunque para el presente documento se dejará un solo archivo de configuración, toda vez de que el puerto de acceso (puerto interno), se parametriza apenas se sube el contenedor (se mapea el que expone); de manera que de forma interna cada contenedor (independiente) puede utilizar el 8080, pero de forma externa al crear el contenedor se expone de otra forma (8888 y 8889 para el siguiente ejemplo):
Ejemplo:
docker run -d --name tomcat1 -p 8888:8080 tomcat8/test:1.0
docker run -d --name tomcat2 -p 8889:8080 tomcat8/test:1.0

```
PS C:\Img\Tomcat> docker run -d --name tomcat2 -p 8889:8080 tomcat8/test:1.0
086c289d3c7d47bd971bdd3dc89e2d825c3dd83cccb1e58b8374048a4934ad39
PS C:\Img\Tomcat> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
     NAMES
086c289d3c7d        tomcat8/test:1.0    "catalina.sh run"   18 seconds ago      Up 4 seconds        0.0.0.0:8889->8080/t
cp   tomcat2
d6dc1e59a942        tomcat8/test:1.0    "catalina.sh run"   32 minutes ago      Up 32 minutes       0.0.0.0:8888->8080/t
cp   tomcat1
PS C:\Img\Tomcat>
```

De tal manera que este punto es OPCIONAL, para esta arquitectura se deja como una posibilidad en caso de necesitar parametrizar los puertos de otra forma.

Servicio tomcat

Se inicia creación de imagen que alojará el servicio tomcat.

- Se construye imagen y se despliega contenedor

  docker build --tag tomcat8/test:1.0 .

  docker run -it --rm -p 8888:8080 tomcat10/test:1.0

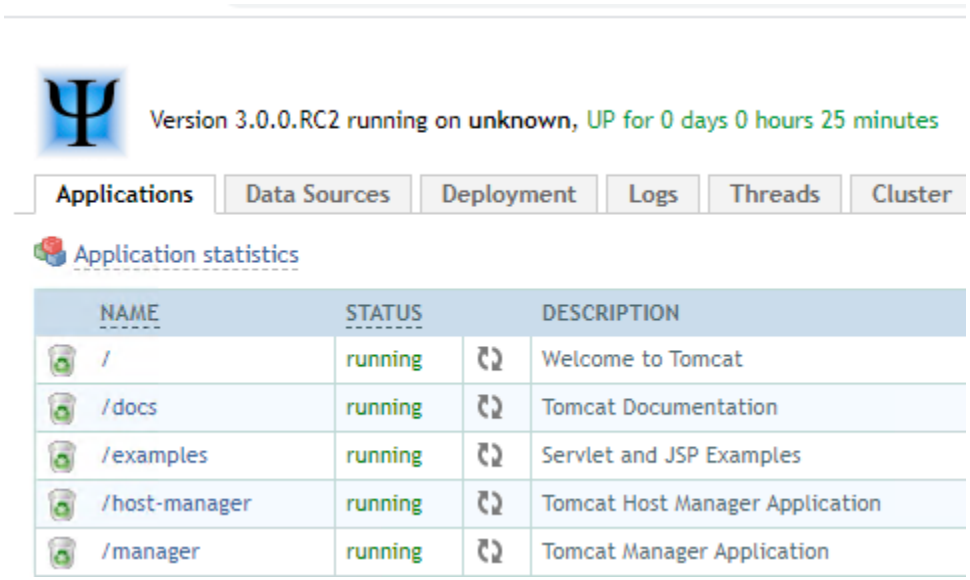
  Como la anterior forma de desplegar el contenedor era temporal (por el –rm), era necesario pensar en otra manera de subirlo para dejarlo ejecutando de fondo:

  - El contenedor también se puede desplegar de otra forma, que permita colocarle un nombre a la imagen y correr en segundo plano

docker run -d --name tomcat1 -p 8888:8080 tomcat8/test:1.0

```
PS C:\Img\Tomcat> docker run -d --name tomcat1 -p 8888:8080 tomcat8/test:1.0
d6dc1e59a94221f0ec336af538f10f34060b7cc594061898411c967989b62b25
PS C:\Img\Tomcat> docker ps
PS C:\Img\Tomcat> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
d6dc1e59a942        tomcat8/test:1.0    "catalina.sh run"   58 seconds ago      Up 44 seconds       0.0.0.0:8888->8080/tcp   tomcat1
PS C:\Img\Tomcat>
```

- Se realiza un test de prueba de tomcat arriba y probe arriba



Algunos comandos necesarios

- Acceso al contenedor

  docker exec -i -t 665b4a1e17b6 /bin/bash
  docker exec -i -t tomcat1 /bin/bash

```
PS C:\Img\Tomcat> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                    NAMES
d6dc1e59a942        tomcat8/test:1.0    "catalina.sh run"   3 minutes ago       Up 3 minutes        0.0.0.0:8888->8080/tcp   tomcat1
PS C:\Img\Tomcat> docker exec -i -t tomcat1 /bin/bash
root@d6dc1e59a942:/usr/local/tomcat# ls -ltr
total 120
-rw-r--r-- 1 root root  16258 Jun 29  2018 RUNNING.txt
-rw-r--r-- 1 root root   6794 Jun 29  2018 RELEASE-NOTES
-rw-r--r-- 1 root root   1446 Jun 29  2018 NOTICE
-rw-r--r-- 1 root root  57011 Jun 29  2018 LICENSE
drwxr-xr-x 2 root root   4096 Sep 12  2018 lib
drwxr-sr-x 3 root staff  4096 Sep 12  2018 native-jni-lib
drwxr-sr-x 3 root staff  4096 Sep 12  2018 include
drwxr-xr-x 2 root root   4096 Sep 12  2018 bin
drwxr-xr-x 1 root root   4096 May 20 06:19 conf
drwxr-xr-x 1 root root   4096 May 20 06:19 temp
drwxrwxrwx 1 root root   4096 May 20 06:19 work
drwxr-xr-x 1 root root   4096 May 20 06:19 webapps
drwxrwxrwx 1 root root   4096 May 20 06:19 logs
root@d6dc1e59a942:/usr/local/tomcat#
```

- Copiar un war en un contenedor desplegado

docker cp C:\Img\Tomcat\war\probe.war container_name:/usr/local/tomcat/webapps

- Copiar en Docker file Windows

copy C:\Img\Tomcat\war\probe.war /usr/local/tomcat/webapps

- Eliminar contenedor

docker rm my-running-haproxy

- Elimina imagen

  Docker rmi nombre:tag

HAPROXY

Instalación Docker en servidor ubuntu

- Actualiza los repos

  sudo apt-get update

```
Hit:1 http://archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://archive.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Fetched 177 kB in 1s (125 kB/s)
Reading package lists... Done
```

- Instalar utilidades

  sudo apt-get install apt-transport-https ca-certificates curl software-properties-common –y

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'apt' instead of 'apt-transport-https'
ca-certificates is already the newest version (20210119~18.04.1).
curl is already the newest version (7.58.0-2ubuntu3.13).
software-properties-common is already the newest version (0.96.24.32.14).
The following additional packages will be installed:
  apt-utils libapt-pkg5.0
Suggested packages:
  apt-doc aptitude | synaptic | wajig dpkg-dev
The following packages will be upgraded:
  apt apt-utils libapt-pkg5.0
3 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
Need to get 2,216 kB of archives.
After this operation, 1,024 B of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 libapt-pkg5.0 amd64 1.6.13 [808 kB]
Get:2 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 apt amd64 1.6.13 [1,201 kB]
Get:3 http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 apt-utils amd64 1.6.13 [206 kB]
Fetched 2,216 kB in 2s (1,209 kB/s)
(Reading database ... 138280 files and directories currently installed.)
Preparing to unpack .../libapt-pkg5.0_1.6.13_amd64.deb ...
Unpacking libapt-pkg5.0:amd64 (1.6.13) over (1.6.12ubuntu0.2) ...
Setting up libapt-pkg5.0:amd64 (1.6.13) ...
(Reading database ... 138280 files and directories currently installed.)
Preparing to unpack .../archives/apt_1.6.13_amd64.deb ...
Unpacking apt (1.6.13) over (1.6.12ubuntu0.2) ...
Setting up apt (1.6.13) ...
(Reading database ... 138280 files and directories currently installed.)
Preparing to unpack .../apt-utils_1.6.13_amd64.deb ...
Unpacking apt-utils (1.6.13) over (1.6.12ubuntu0.2) ...
Setting up apt-utils (1.6.13) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.4) ...
```

- Agregar el gpg

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

```
OK
```

- Agregar el repo

  sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

```
Get:1 https://download.docker.com/linux/ubuntu bionic InRelease [64.4 kB]
Get:2 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages [18.1 kB]
Hit:3 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:4 http://archive.ubuntu.com/ubuntu bionic-security InRelease
Hit:5 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Fetched 82.6 kB in 1s (71.6 kB/s)
Reading package lists... Done
```

- Actualizar de nuevo

  sudo apt-get update

```
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:3 http://archive.ubuntu.com/ubuntu bionic-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Reading package lists... Done
```

- Instalar Docker

  sudo apt-get install docker-ce

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  containerd.io docker-ce-cli docker-ce-rootless-extras docker-scan-plugin libltdl7
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
Recommended packages:
  pigz slirp4netns
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-scan-plugin libltdl7
0 upgraded, 6 newly installed, 0 to remove and 45 not upgraded.
Need to get 107 MB of archives.
After this operation, 465 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://download.docker.com/linux/ubuntu bionic/stable amd64 containerd.io amd64 1.4.4-1 [28.3 MB]
Get:2 http://archive.ubuntu.com/ubuntu bionic/main amd64 libltdl7 amd64 2.4.6-2 [38.8 kB]
Get:3 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce-cli amd64 5:20.10.6~3-0~ubuntu-bionic [41.4 MB]
Get:4 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce amd64 5:20.10.6~3-0~ubuntu-bionic [24.8 MB]
Get:5 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce-rootless-extras amd64 5:20.10.6~3-0~ubuntu-bionic [9,067 kB]
Get:6 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-scan-plugin amd64 0.7.0~ubuntu-bionic [3,884 kB]
Fetched 107 MB in 4s (29.4 MB/s)
Selecting previously unselected package containerd.io.
(Reading database ... 138280 files and directories currently installed.)
Preparing to unpack .../0-containerd.io_1.4.4-1_amd64.deb ...
Unpacking containerd.io (1.4.4-1) ...
Selecting previously unselected package docker-ce-cli.
Preparing to unpack .../1-docker-ce-cli_5%3a20.10.6~3-0~ubuntu-bionic_amd64.deb ...
Unpacking docker-ce-cli (5:20.10.6~3-0~ubuntu-bionic) ...
Selecting previously unselected package docker-ce.
Preparing to unpack .../2-docker-ce_5%3a20.10.6~3-0~ubuntu-bionic_amd64.deb ...
Unpacking docker-ce (5:20.10.6~3-0~ubuntu-bionic) ...
Selecting previously unselected package docker-ce-rootless-extras.
Preparing to unpack .../3-docker-ce-rootless-extras_5%3a20.10.6~3-0~ubuntu-bionic_amd64.deb ...
Unpacking docker-ce-rootless-extras (5:20.10.6~3-0~ubuntu-bionic) ...
Selecting previously unselected package docker-scan-plugin.
Preparing to unpack .../4-docker-scan-plugin_0.7.0~ubuntu-bionic_amd64.deb ...
Unpacking docker-scan-plugin (0.7.0~ubuntu-bionic) ...
Selecting previously unselected package libltdl7:amd64.
Preparing to unpack .../5-libltdl7_2.4.6-2_amd64.deb ...
Unpacking libltdl7:amd64 (2.4.6-2) ...
Setting up containerd.io (1.4.4-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/containerd.service → /lib/systemd/system/containerd.service.
```

- Iniciarlo con el sistema

  sudo systemctl enable docker

```
Synchronizing state of docker.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
```

- Imagen de prueba

  docker run hello-world

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
b8dfde127a29: Pull complete
Digest: sha256:5122f6204b6a3596e048758cabba3c46b1c937a46b5be6225b835d091b90e46c
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

```
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    d1165f221234   2 months ago   13.3kB
```

HAPROXY Imagen

- Se crea carpeta para alojar imagen

  mkdir docker-images

- *Se crea dockerfile*

```
FROM haproxy:2.3
COPY haproxy.cfg /usr/local/etc/haproxy/haproxy.cfg
```

```
otal 20
rw-r--r-- 1 root root  119 Jun  2 06:26 Dockerfile
rw-r--r-- 1 root root 4005 Jun  2 06:41 haproxy.cfg_2021_06_02
rw-r--r-- 1 root root 3388 Jun  9 06:13 haproxy.cfg_vjhon
rw-r--r-- 1 root root 4109 Jun  9 07:16 haproxy.cfg
```

- *Se crea archivo haproxy.cfg en la ruta de la imagen (documentación: https://github.com/haproxytech/haproxy-docker-ubuntu/blob/master/2.3/haproxy.cfg)*

```
#---------------------------------------------------------------------,
# Example configuration for a possible web application. See the
# full configuration options online.
#
# https://www.haproxy.org/download/2.3/doc/configuration.txt
# https://cbonte.github.io/haproxy-dconv/2.3/configuration.html
#
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# Global settings
#---------------------------------------------------------------------
global
# to have these messages end up in /var/log/haproxy.log you will
# need to:
#
# 1) configure syslog to accept network log events. This is done
# by adding the '-r' option to the SYSLOGD_OPTIONS in
# /etc/sysconfig/syslog
#
# 2) configure local2 events to go to the /var/log/haproxy.log
# file. A line like the following can be added to
# /etc/sysconfig/syslog
#
# local2.* /var/log/haproxy.log
#
log 127.0.0.1 local2

chroot /var/lib/haproxy
pidfile /var/run/haproxy.pid
maxconn 4000
user haproxy
group haproxy
# daemon

# turn on stats unix socket
stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
```

```
#---------------------------------------------------------------------
defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 3000

#---------------------------------------------------------------------
# example how to define user and enable Data Plane API on tcp/5555
# more information: https://github.com/haproxytech/dataplaneapi and
# https://www.haproxy.com/documentation/hapee/2-0r1/configuration/dataplaneapi/
#---------------------------------------------------------------------
# userlist haproxy-dataplaneapi
# user admin insecure-password mypassword
#
# program api
# command /usr/bin/dataplaneapi --host 0.0.0.0 --port 5555 --haproxy-bin /usr/sbin/haproxy --
config-file /etc/haproxy/haproxy.cfg --reload-cmd "kill -SIGUSR2 1" --reload-delay 5 --userlist
hapee-dataplaneapi
# no option start-on-reload

#---------------------------------------------------------------------
# main frontend which proxys to the backends
#---------------------------------------------------------------------
frontend main
bind *:80
# bind *:443 ssl # To be completed ....

acl url_static path_beg -i /static /images /javascript /stylesheets
acl url_static path_end -i .jpg .gif .png .css .js
```

use_backend static if url_static
default_backend app

#---
 # static backend for serving up images, stylesheets and such
#---
backend static
balance roundrobin
server static1 127.0.0.1:4331 check
server static2 127.0.0.1:4332 check

#---
 # round robin balancing between the various backends
#---
backend app
balance roundrobin
server app1 127.0.0.1:5001 check
server app2 127.0.0.1:5002 check
server app3 127.0.0.1:5003 check
server app4 127.0.0.1:5004 check

- Despliegue de imagen

```
Sending build context to Docker daemon  2.048kB
Step 1/1 : FROM haproxy:2.3
2.3: Pulling from library/haproxy
69692152171a: Pull complete
4a113359dddc: Pull complete
eb3f7a3e9532: Pull complete
55f06dddfab5: Pull complete
78a4c0ac4427: Pull complete
Digest: sha256:befd197ad0b554bfb0e2c1f3481bf92d66d963346a5261f00b2983d74fd3dbc5
Status: Downloaded newer image for haproxy:2.3
 ---> 397cf3d55fac
Successfully built 397cf3d55fac
Successfully tagged haproxy:latest
```

creando imagen
docker build --tag myhaproxy .

- Check archivo de configuración

docker run -it --rm --name haproxy-syntax-check my-haproxy haproxy -c -f /usr/local/etc/haproxy/haproxy.cfg

```
Configuration file is valid
```

docker run -it --rm --name haproxy-syntax-check myhaproxy haproxy -c -f /usr/local/etc/haproxy/haproxy.cfg

- Se ejecuta contenedor

#se despliega exponiendo el puerto 8080
docker run -d --name my-running-haproxy --sysctl net.ipv4.ip_unprivileged_port_start=0 -p 8080:80 myhaproxy

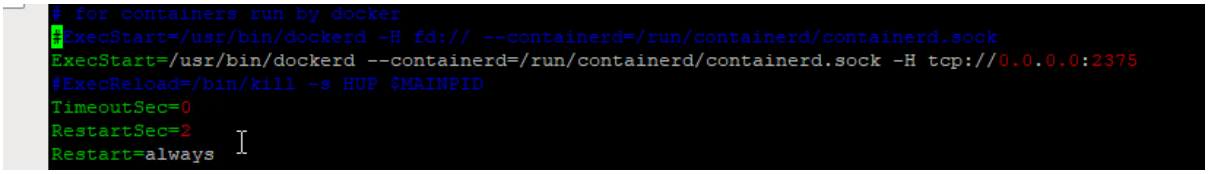
Portainer

Acceso por el puerto 2375

Este punto es importante, teniendo en cuenta de que es el puerto API por el cual se expone el daemon del Docker, con el fin de permitir el acceso al orquestador.
Para lo anterior se realiza lo siguiente en el servidor ubuntu

- Modificar archivo docker.service
- Cambiar la línea

ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
Por la siguiente
ExecStart=/usr/bin/dockerd --containerd=/run/containerd/containerd.sock -H tcp://0.0.0.0:2375



- sudo systemctl daemon-reload
- sudo systemctl restart docker.service
- sudo systemctl status docker.service
- Posteriormente, es necesario declara docker_host, teniendo en cuenta que ahora cualquier comando con docker no me funciona por tener el tcp activo

export DOCKER_HOST="tcp://0.0.0.0:2375"

- Luego ya funcionará el docker ps (por ejemplo)

Instalando Portainer

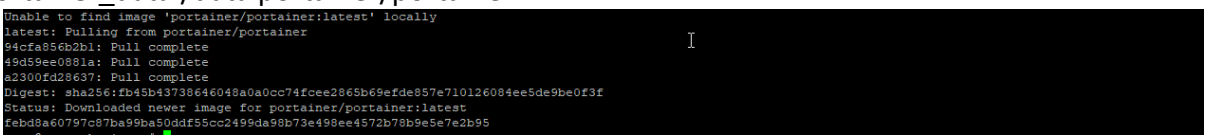
El Portainer se instala y configura en el servidor linux entregado para el laboratorio

- Creando volumen

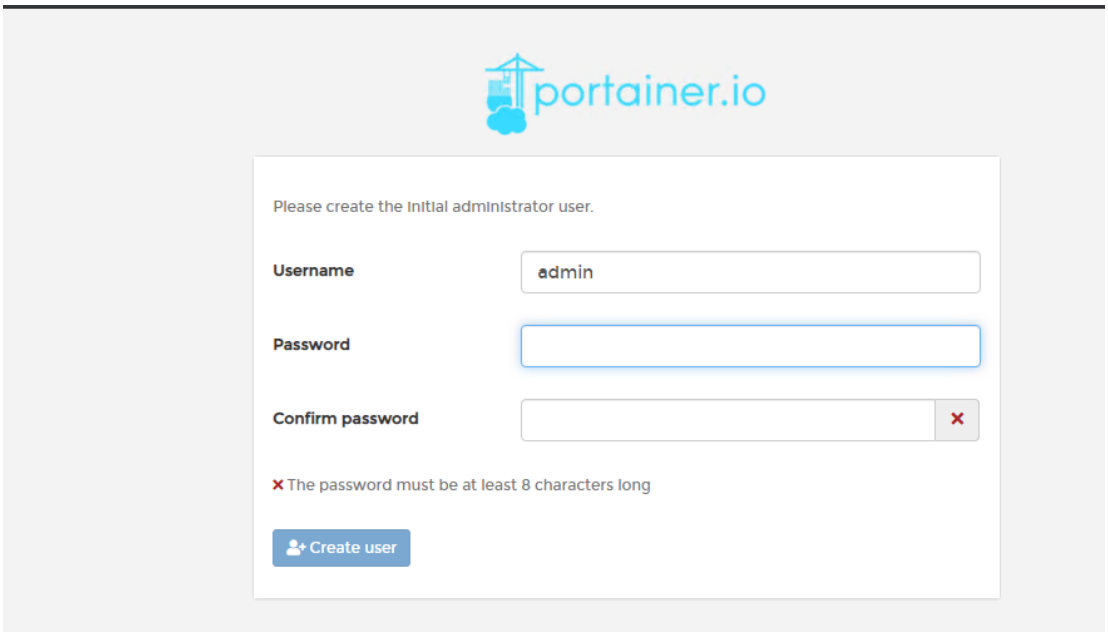
docker créate portainer_data

- Desplegando contenedor

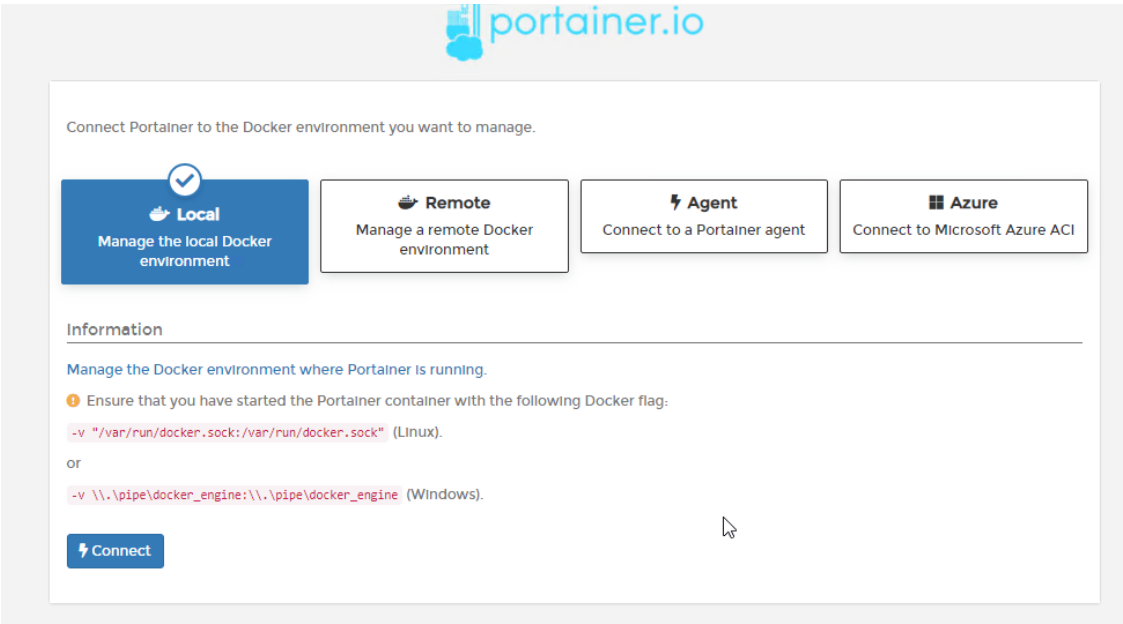
docker run -d -p 9000:9000 --name=portainer -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer



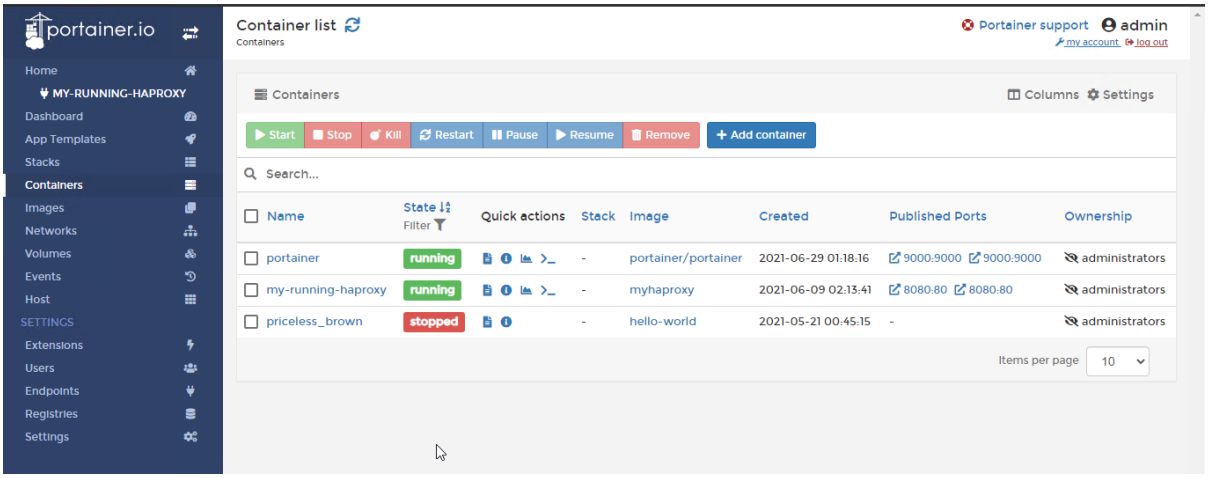
- Prueba de acceso



- Se crea la contraseña
- Para iniciar trabajamos el acceso a los contenedores locales, teniendo en cuenta que utilizamos el mismo servidor donde se tiene el haproxy



Nota: se puede colocar también remoto (aunque es local), dejando la ip del servidor linux, puerto 2375 y en nombre se coloca una del docker ps (ej: my-running-haproxy)



Accediendo a contenedor en Windows desde Portainer

Como es necesario llegar desde el Portainer hasta los contenedores desplegados en Windows, es necesario habilitar el acceso por el api daemon al docker de Windows. Para lo anterior realizamos lo siguientes

- En la ruta del servidor Windows C:\ProgramData\docker\config , se crea el archivo daemon.json con lo siguiente

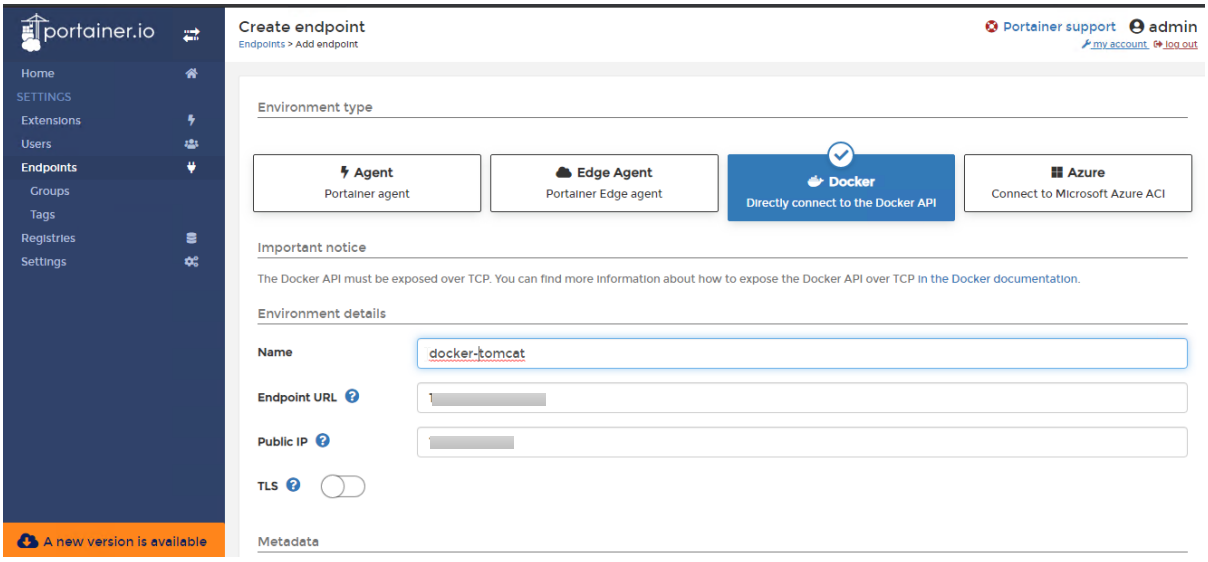
  {"hosts": ["tcp://0.0.0.0:2375","npipe://"]}

- Posteriormente se debe reiniciar el servicio de Docker (previamente detener los contenedores desplegados por buena práctica, reiniciar servicio, subir de nuevo los contenedores)

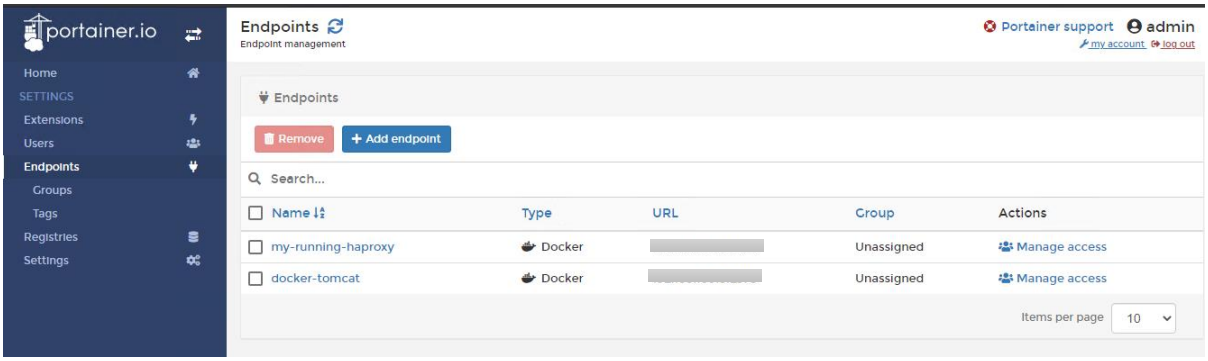
```
36 docker ps
37 docker stop tomcat1
38 docker stop tomcat2
39 docker ps
40 Restart-Service docker
41 docker ps
42 docker start tomcat1
43 docker ps
44 docker start tomcat2
45 docker ps
46 docker ps
```

- Se confirma que el servidor Windows server ya escucha por el puerto 2375

- Ahora se debe configurar el endpoint en gui del Portainer, con el fin de administrar los contenedores de este



- Se confirma correcta conexión y se pueden visualizar ambos servidores



Script de reinicio

Se desarrolla script en perl para identificar caída de servicios de los contenedores. Los scripts son dejados en el servidor Linux provistos por la Universidad, para el demo entregado.

- Se configura cpan
- Se instala LWP
  install Bundle::LWP
- Se instala html tree
  install HTML::Tree

  nota: Otro forma de instalar evitando los pasos anteriores, es con el comando:
  apt-get install libwww-perl

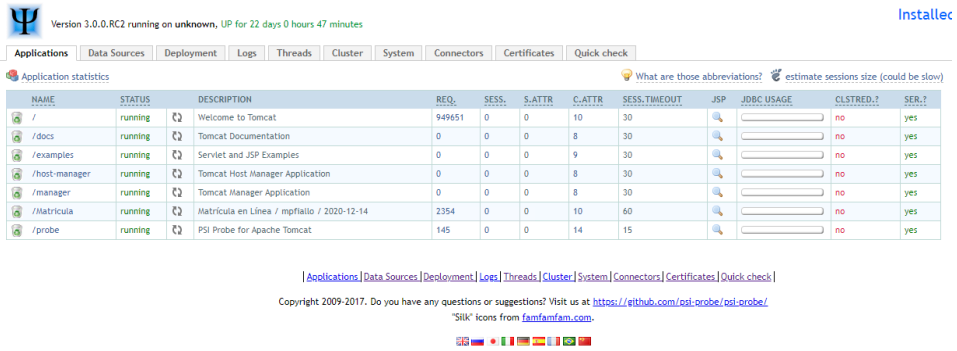
Anexo B. Evidencias de casos de pruebas

El presente anexo muestra las evidencias de los casos de pruebas revisados de la arquitectura planteada

Login

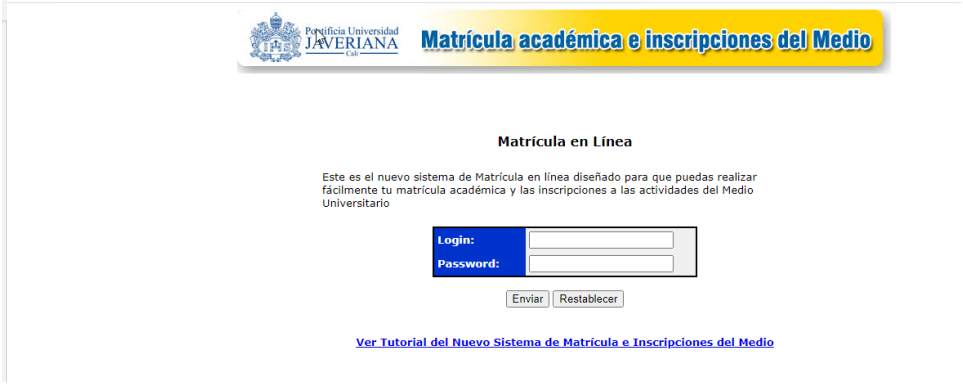
CP_1

Correcto despliegue de servicio y war probe para administrar aplicaciones del tomcat



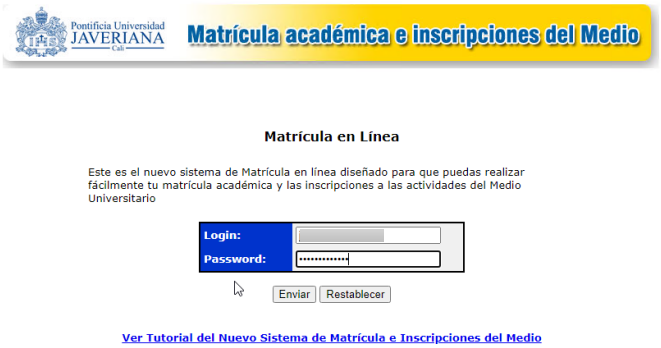
CP_2

Carga de aplicación War demo



CP_3

Acceso con aplicación demo



CP_4

Visualización de asignaturas disponibles de aplicación demo posterior al login

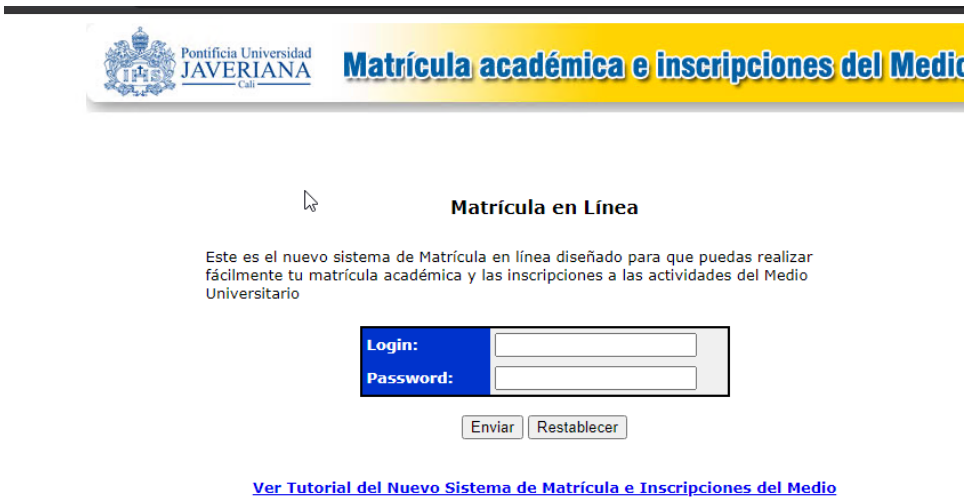
CP_5

Revisión de balanceo del HAPROXY

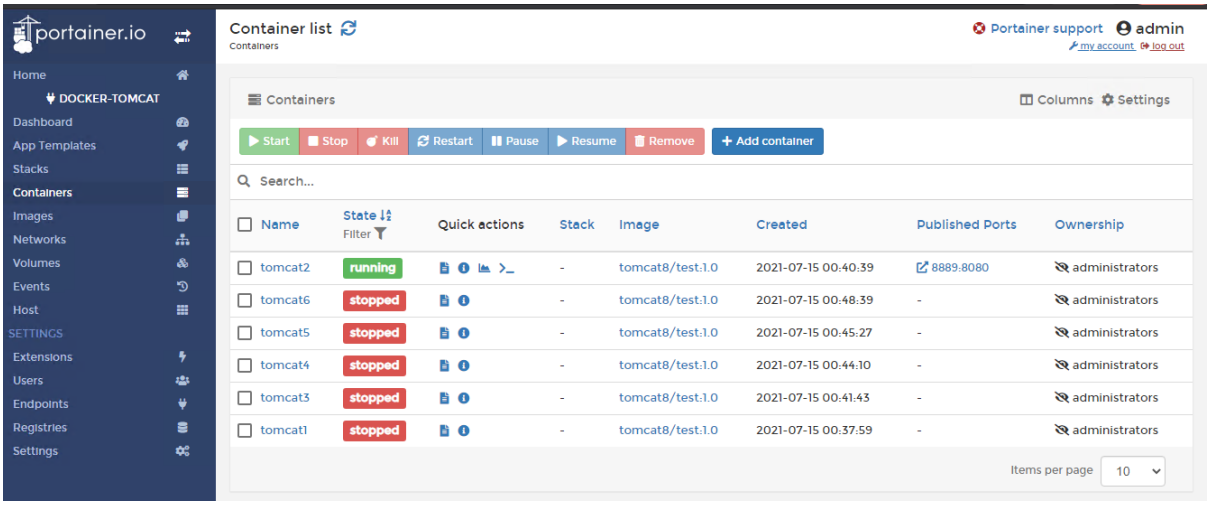
- Configuración de servidores para balancear

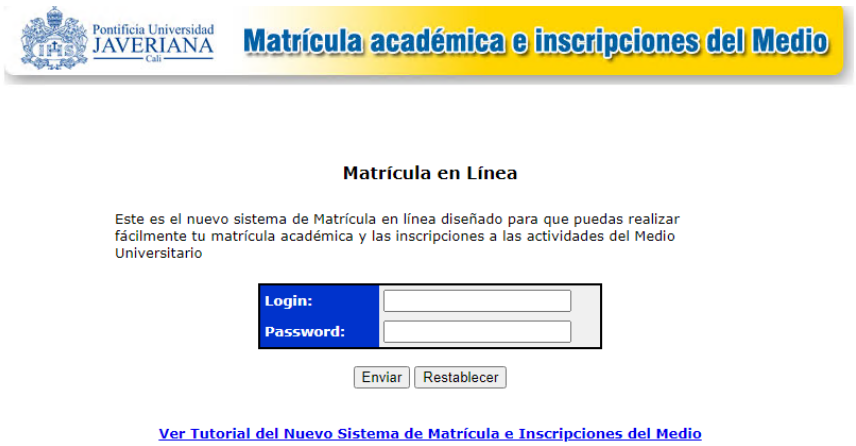
```
#---------------------------
backend static
    balance     roundrobin
    server      static1 192
    server      static2 192
    server      static3 192
    server      static4 192
    #server      static2 12

#---------------------------
# round robin balancing bet
#---------------------------
backend app
    balance     roundrobin

    option httpchk GET / HT
```

- Acceso por medio del haproxy



- Se bajan los contenedores de manera intencional por medio del portainer (para simular caída), dejando uno solo activo, confirmando que hace un correcto check del status de cada nodo backend para no enviar peticiones a esos contenedores que se encuentran detenidos

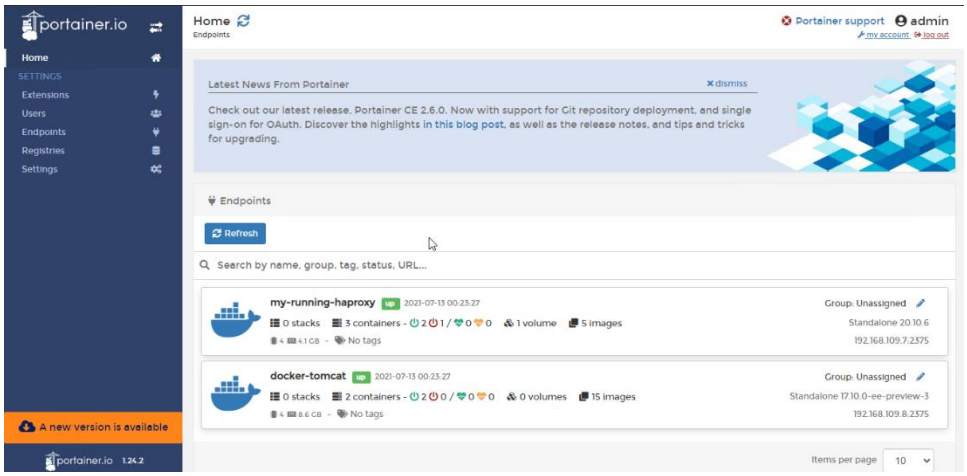




- Se carga varias veces en diferentes navegadores (para temas de caché), confirmando disponibilidad del servicio mediante el único contenedor que se deja activo

CP_6

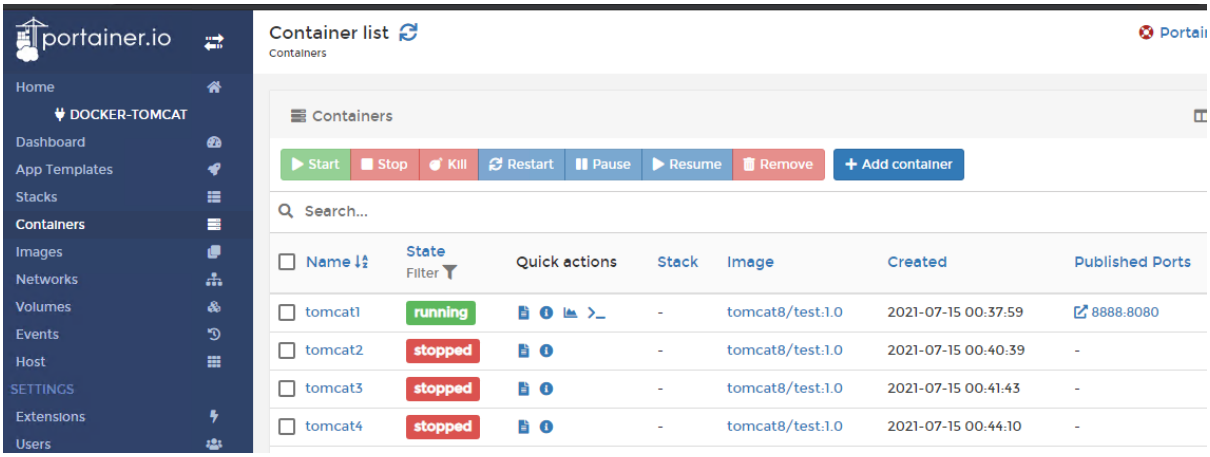
Despliegue y funcionamiento del administrador de los contenedores (Portainer)



CP_7

Prueba de script de restart de contenedores con servicio Tomcat

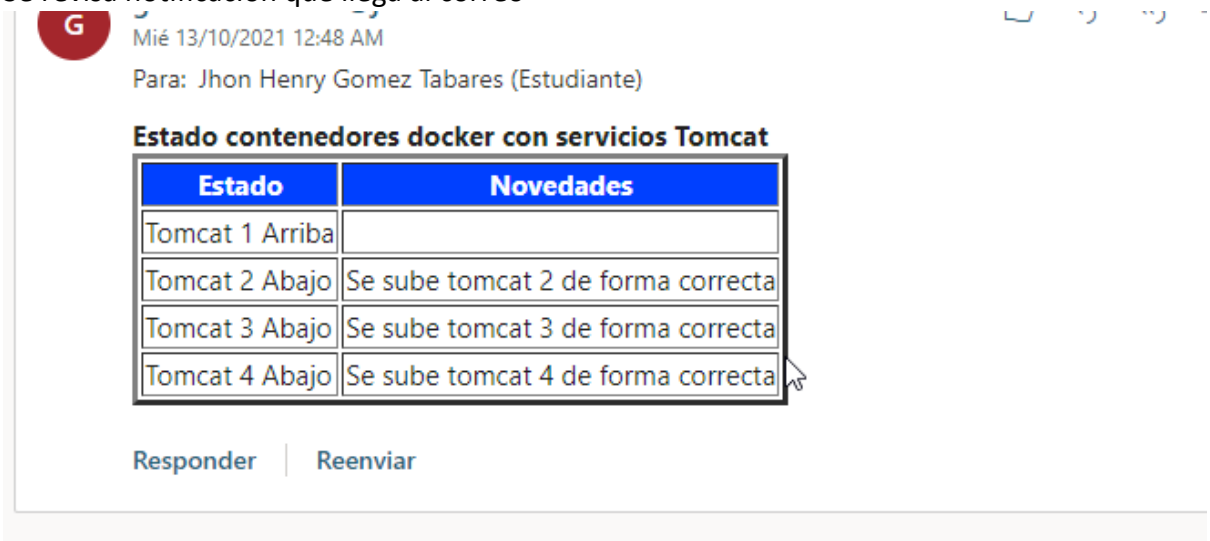
- Se bajan 3 de los cuatro contenedores configurados, para simular caída de los mismos



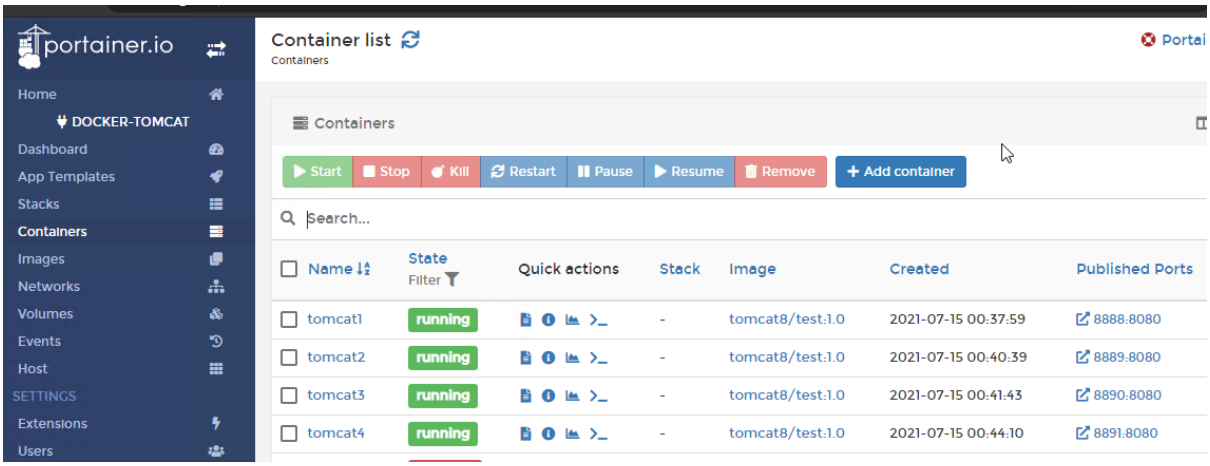
- Se ejecuta script de forma manual (queda documentado de que esto es una tarea de crontab que ejecutará cada x tiempo)
  perl checktomcat.pl

```
tomcat2
tomcat3
tomcat4
Tomcat 1 Arriba
Tomcat 2 Abajo
Se sube tomcat 2 de forma correcta
Tomcat 3 Abajo
Se sube tomcat 3 de forma correcta
Tomcat 4 Abajo
Se sube tomcat 4 de forma correcta
```

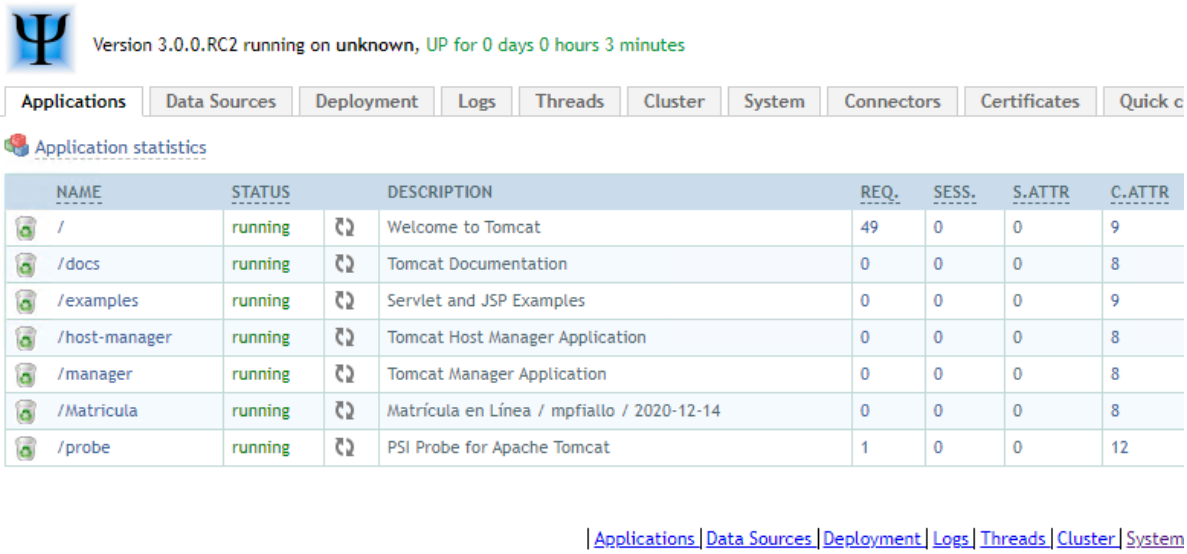
- Se revisa notificación que llega al correo



- Se confirma en el portainer que los contenedores se encuentran iniciados de forma correcta



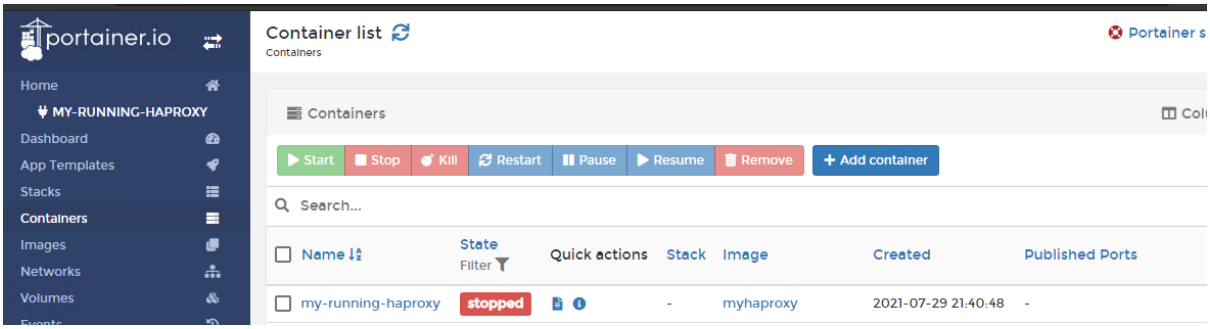
- Se confirma de forma aleatoria que el servicio de uno de los contenedores funcione sin novedades



CP_8

Prueba de script para reinicio de contenedores con el servicio HAProxy

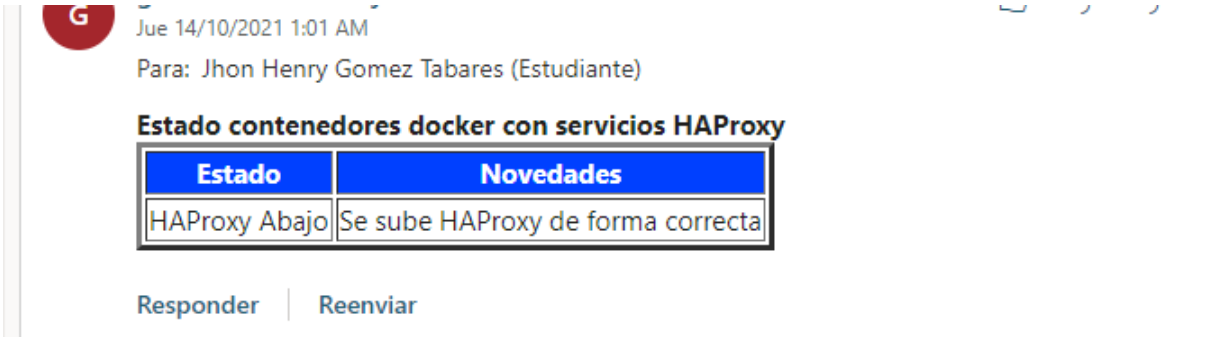
- Se baja contenedor que contiene servicio HAProxy; proceso realizado de forma intencional para simular caída



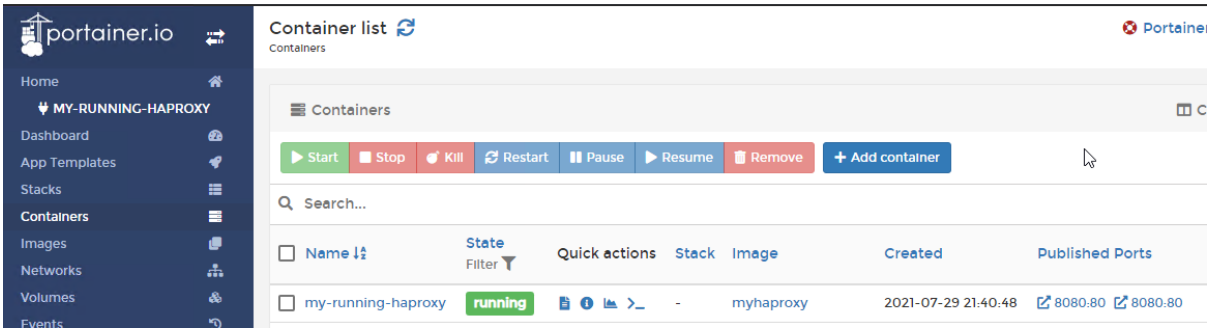
- Se ejecuta script de forma manual (queda documentado de que esto es una tarea de crontab que debe ejecutar cada x tiempo)
  perl checkhaproxy.pl

```
my-running-haproxy
HAProxy Abajo
Se sube HAProxy de forma correcta
```

- Se revisa notificación que llega al correo



- Se confirma en el portainer que los contenedores se encuentran iniciados de forma correcta



Con lo anterior, se confirma contenedor HAProxy arriba, el cual fue reiniciado por el script

Anexo C. Pruebas de carga Apache JMeter

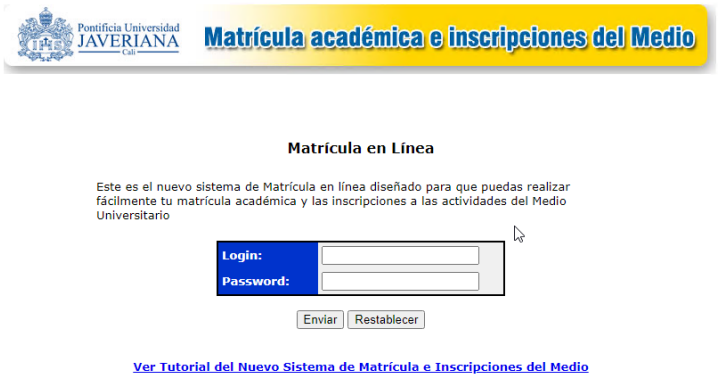
El presente anexo muestra los casos de las pruebas de carga, la configuración e instalación de los componentes necesarios para ejecutar todo el proceso de pruebas determinado de estos aspectos.

Pruebas de carga Login

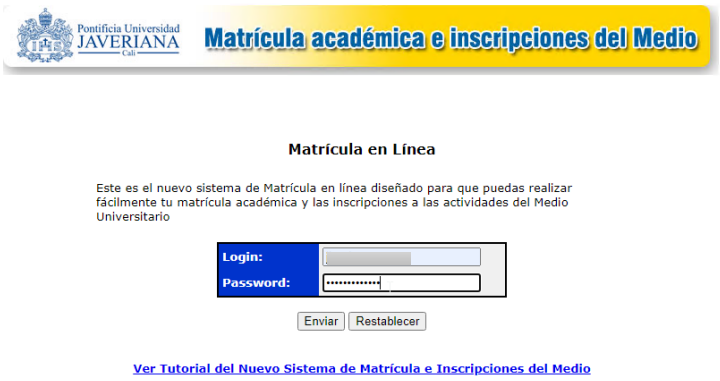
CP_1 Login

Pruebas realizadas ejecutando el login

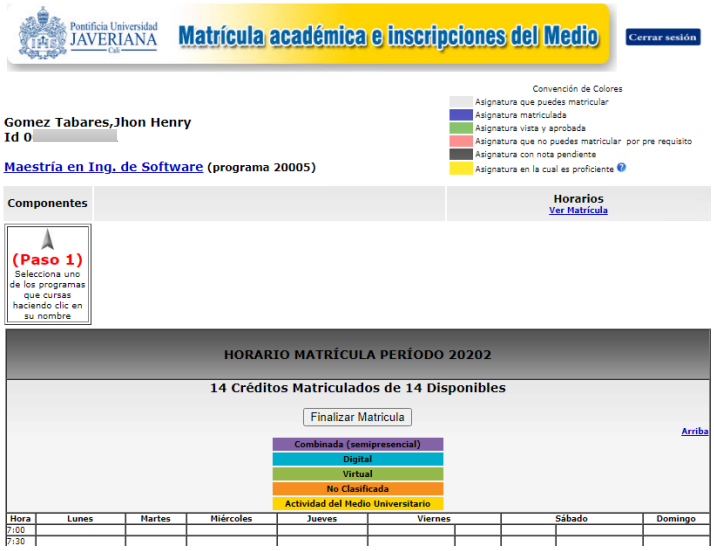
- Pantalla login



- Datos

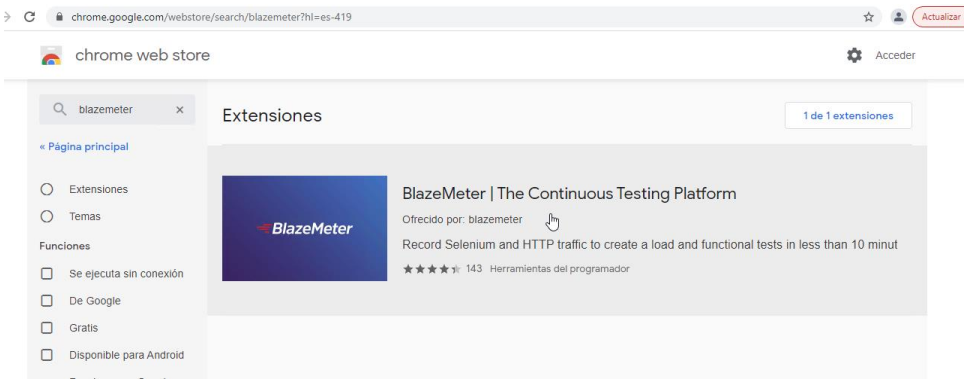


- Acceso

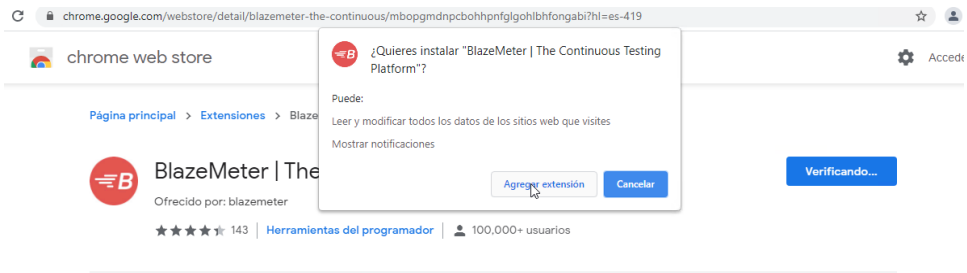


Uso del BlazeMeter

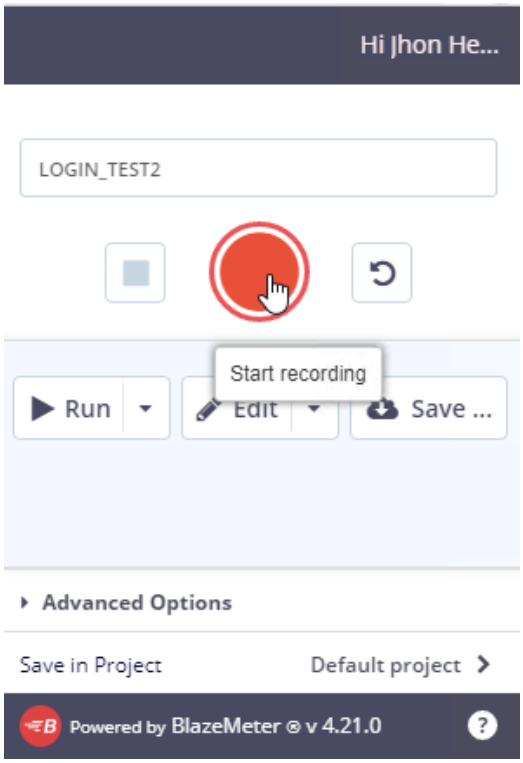
- Instalación de extensión en servidor Windows entregado:



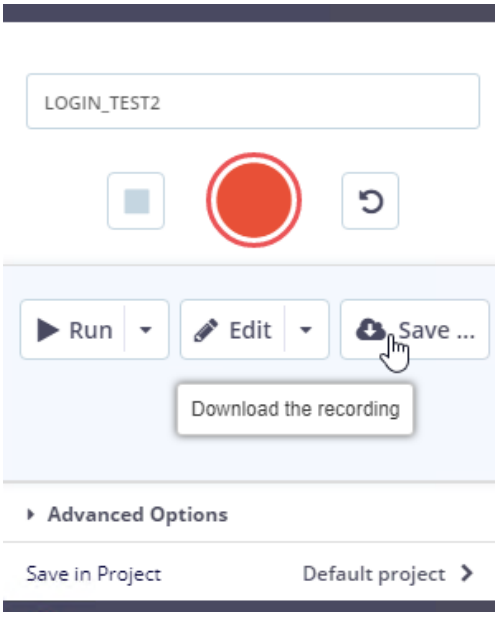
- Agregar extensión



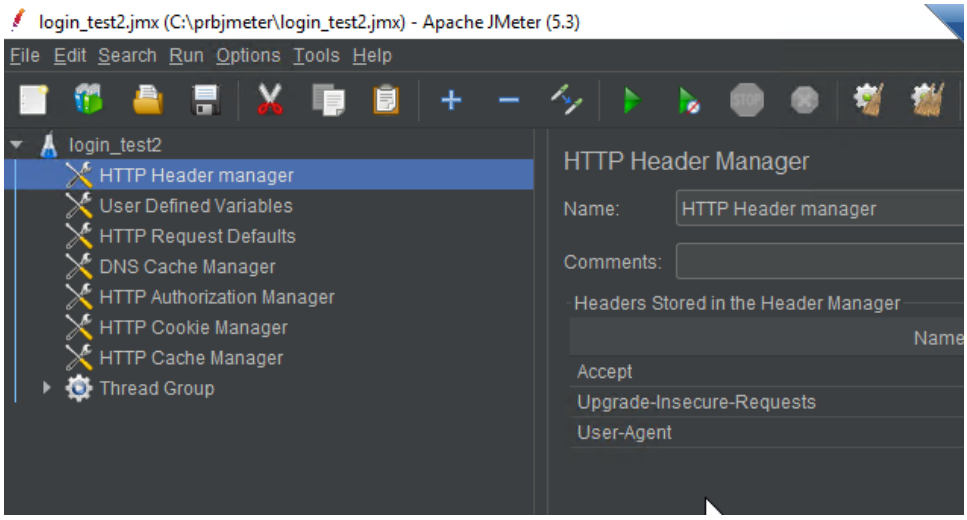
- Iniciar sesión en el Blazemeter (puedes usar la cuenta de Google)
- Configurar sesión de acceso



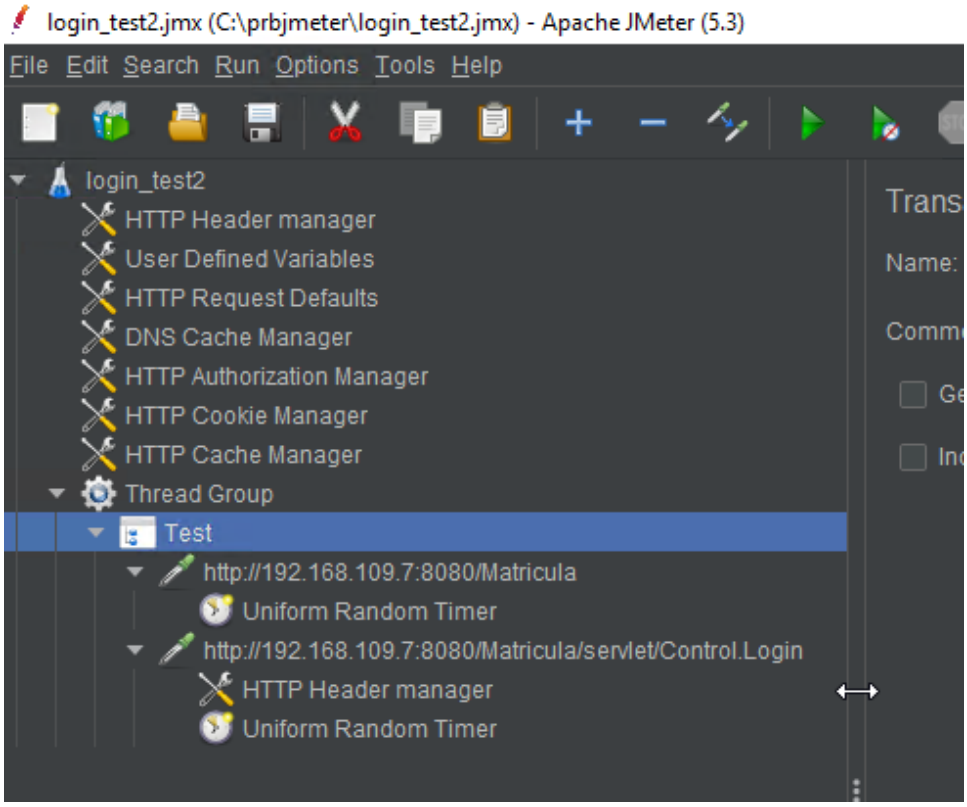
- Apenas se inicie la grabación, se debe realizar el acceso al HA y ejecutar las pruebas; para este caso se prueba un inicio de sesión; finalizado esto, se procede a parar el blazemeter y descargar la tarea



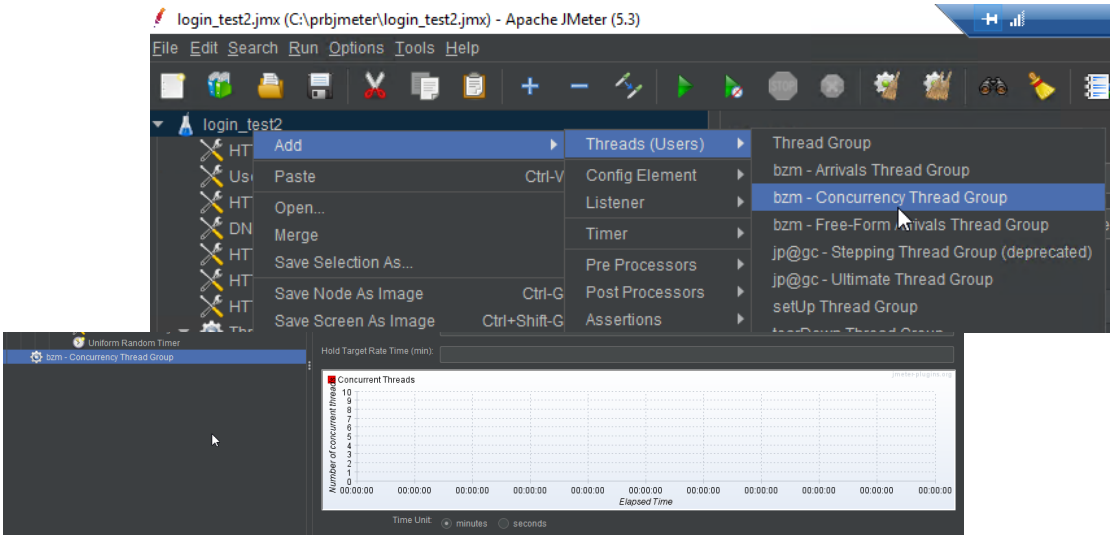
- Posterior a descargar el archivo .jmx, se debe importar en el jmeter



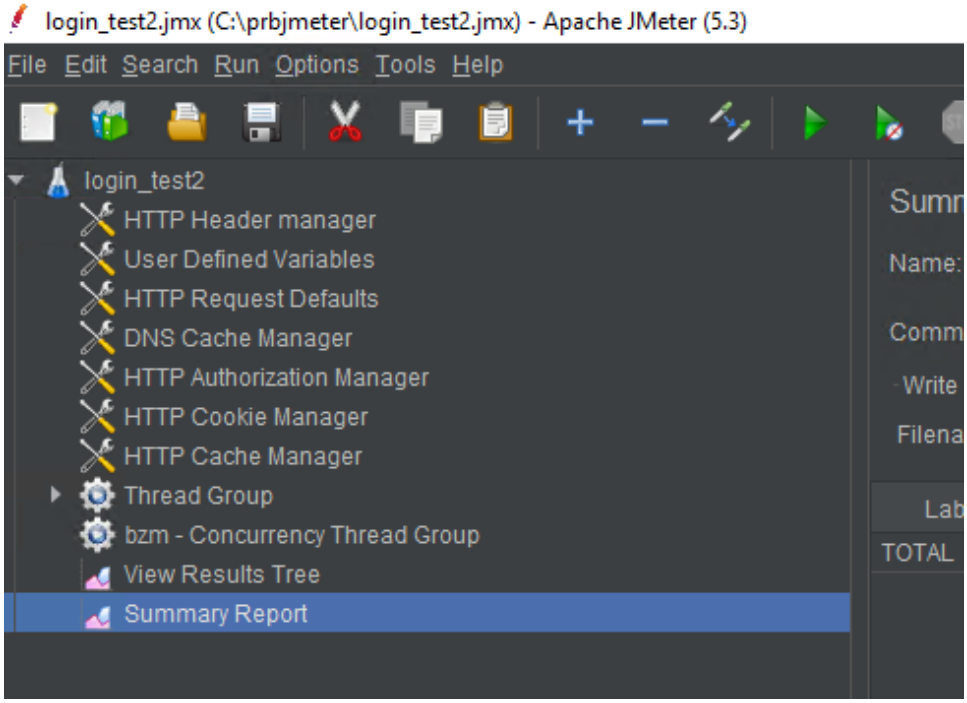
- Se confirma que en el import se observan las url consumidas



- Se activa plugin de concurrencia

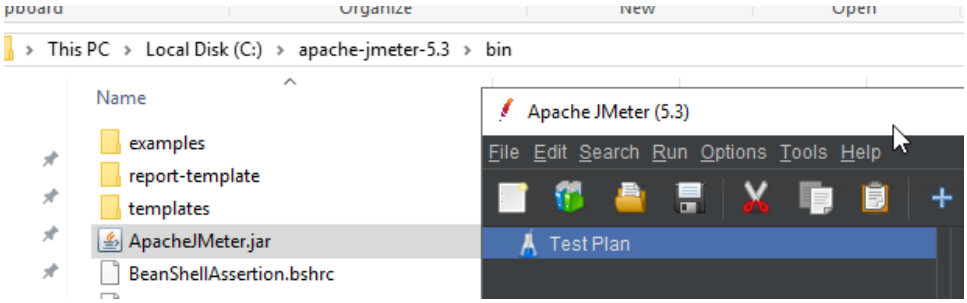


- Se añade también el plugin árbol de resultados y Summary Report

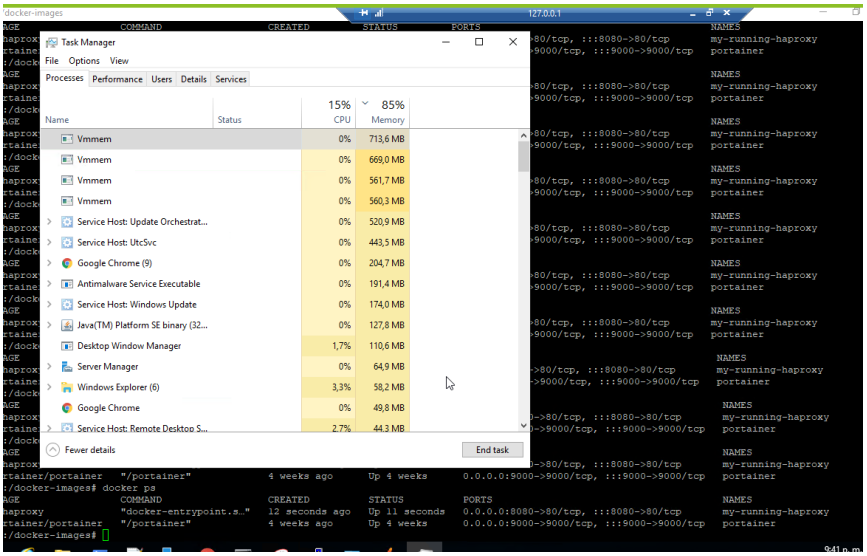


Uso de Jmeter

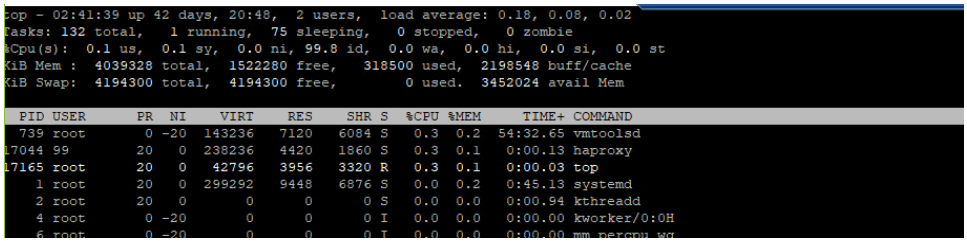
- Se instala versión de java
- Se descarga y se utiliza versión 5.3 de la herramienta JMeter



Estado servidor Windows



Estado Linux

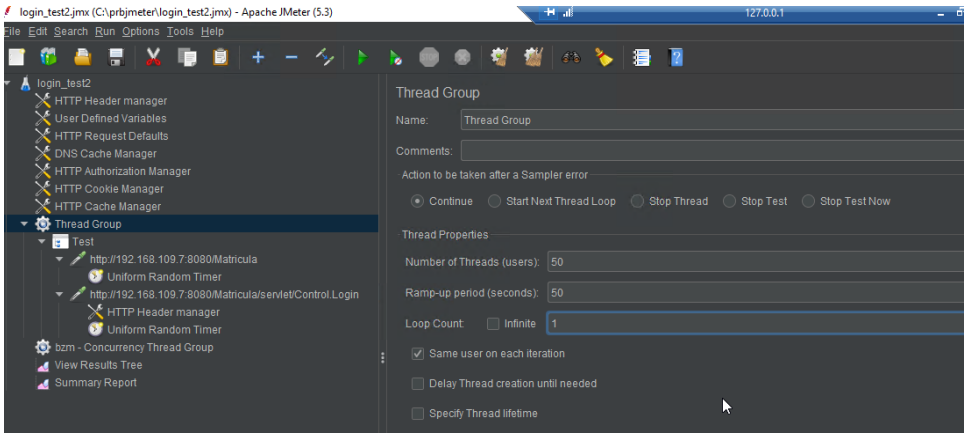


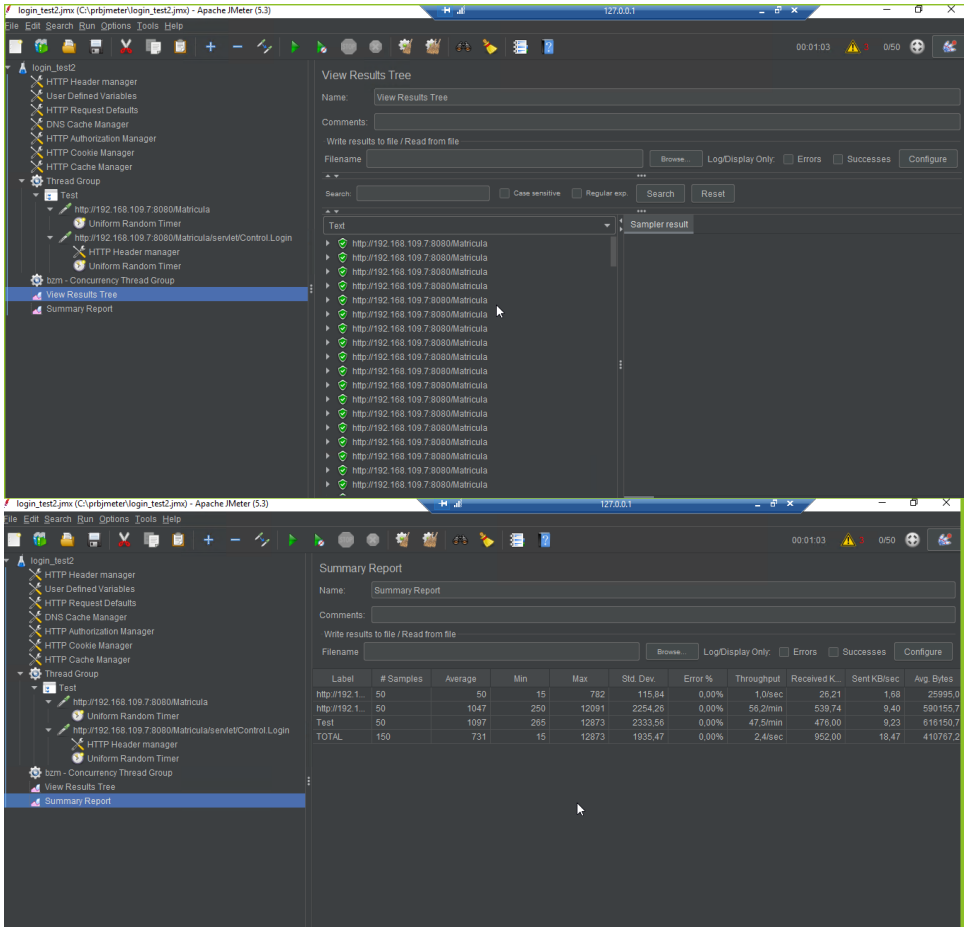
Número de Hilos: Representa el número total de usuarios virtuales que realizan la ejecución del script de prueba.

Período de Subida (en segundos): Representa el tiempo que se tardará en ejecutar el número completo de hilos. Por ejemplo, si tiene 100 usuarios con un período de subida de 50 segundos, JMeter demorará 50 segundos en ejecutar los 100 hilos, agregando 2 hilos por segundo.

Contador del bucle: Representa el número de veces que se ejecutará el script. Por ejemplo, si el contador del bucle es 2 y el número de hilos es de 100, entonces el script se ejecutará 200 veces.

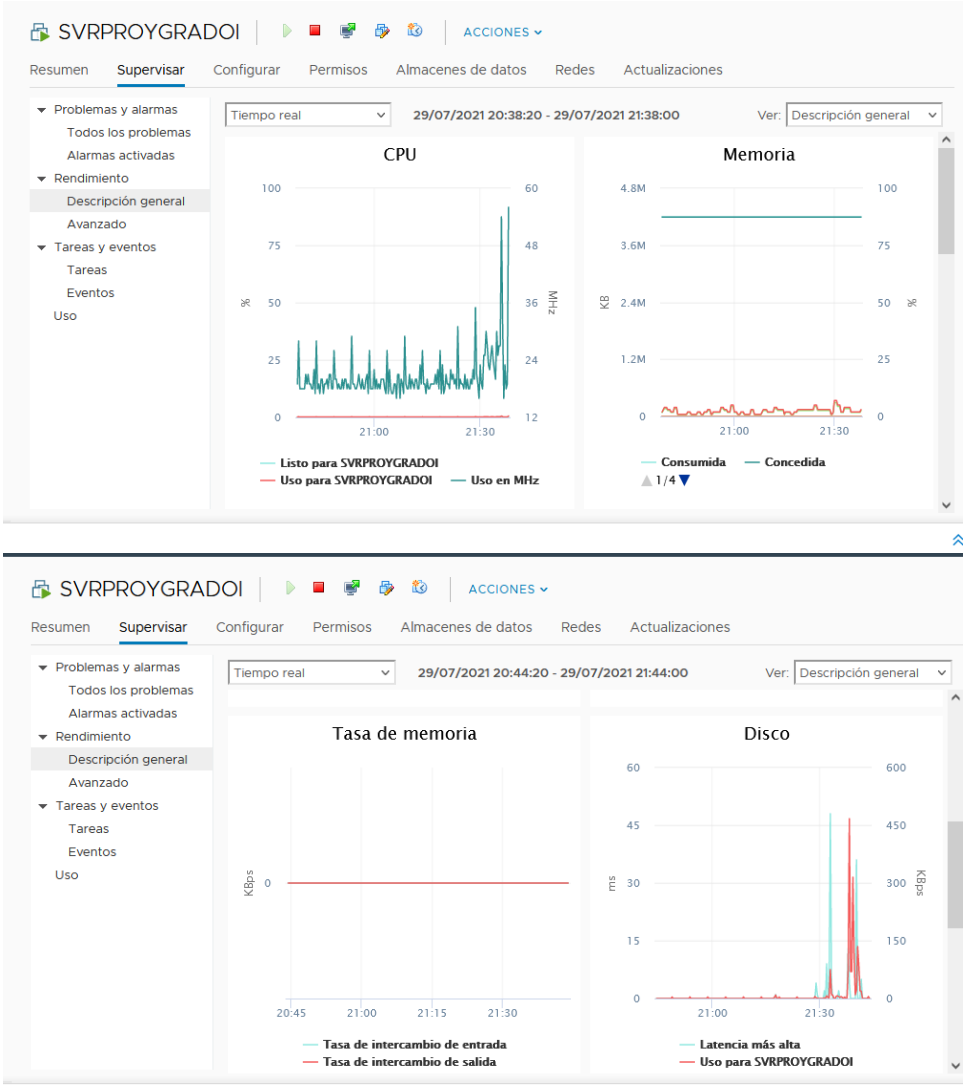
Prueba1



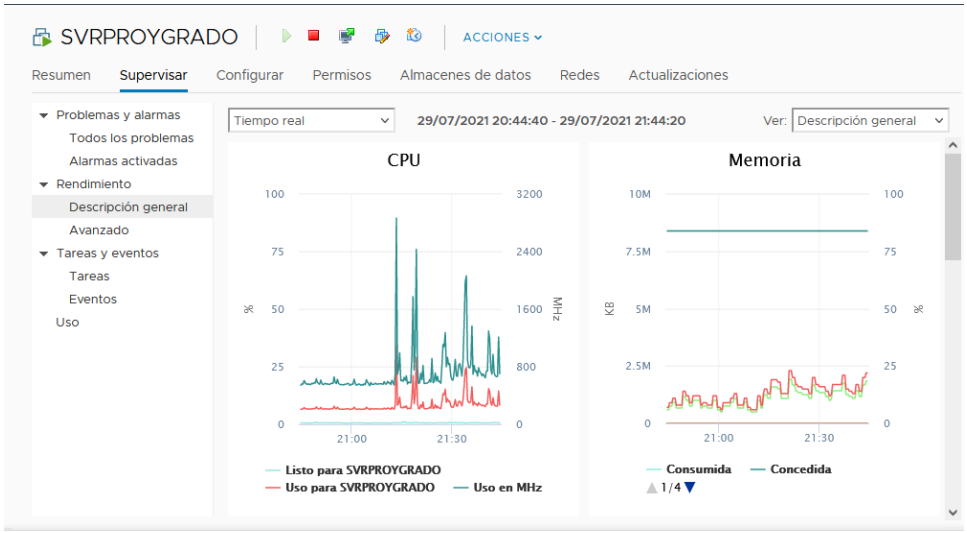


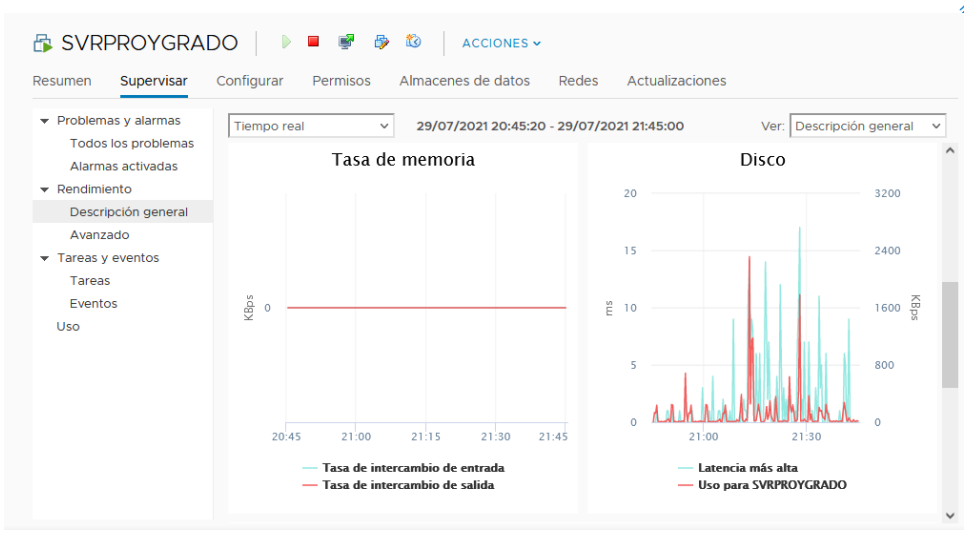
Estado de servidores durante la prueba

Servidor Linux:

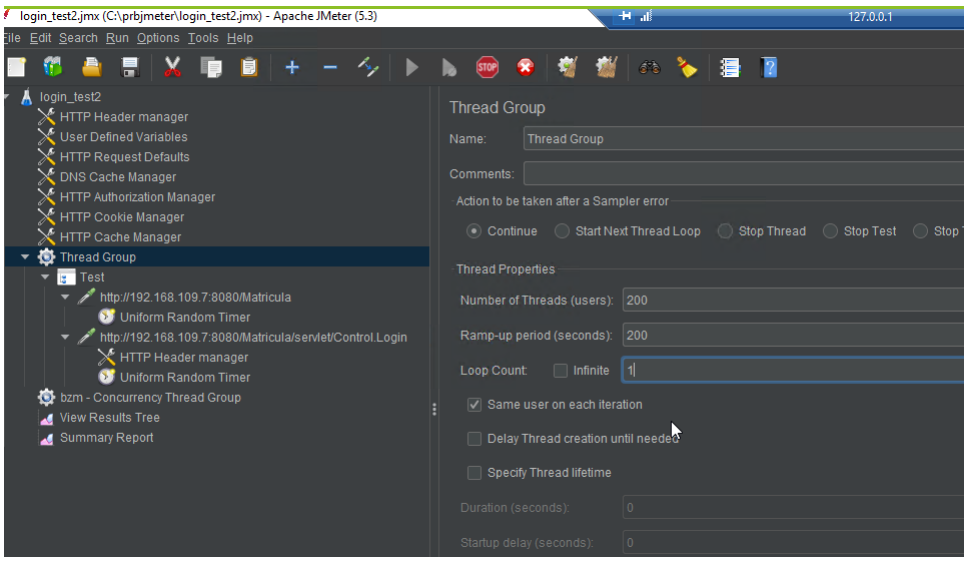


Servidor Windows Server


SVRPROYGRADO
ACCIONES
Resumen
Supervisar
Configurar
Permisos
Almacenes de datos
Redes
Actualizaciones
Problemas y alarmas
Todos los problemas
Alarmas activadas
Rendimiento
Descripción general
Avanzado
Tareas y eventos
Tareas
Eventos
Uso
Tiempo real
29/07/2021 20:44:40 - 29/07/2021 21:44:20
Ver:
Descripción general
CPU
Memoria
Listo para SVRPROYGRADO
Uso para SVRPROYGRADO
Uso en MHz
Consumida
Concedida
1 / 4

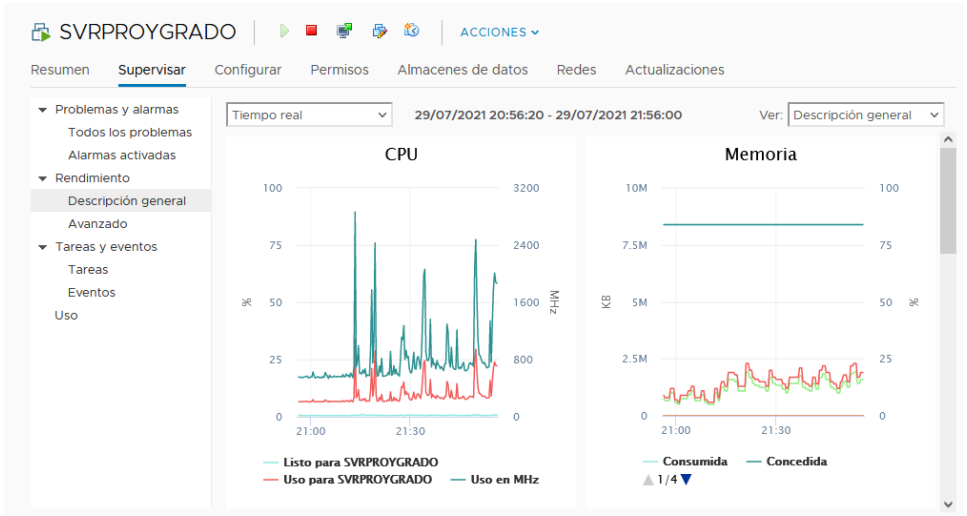

SVRPROYGRADO
ACCIONES
Resumen
Supervisar
Configurar
Permisos
Almacenes de datos
Redes
Actualizaciones
Problemas y alarmas
Todos los problemas
Alarmas activadas
Rendimiento
Descripción general
Avanzado
Tareas y eventos
Tareas
Eventos
Uso
Tiempo real
29/07/2021 20:45:20 - 29/07/2021 21:45:00
Ver:
Descripción general
Tasa de memoria
Disco
Tasa de intercambio de entrada
Tasa de intercambio de salida
Latencia más alta
Uso para SVRPROYGRADO

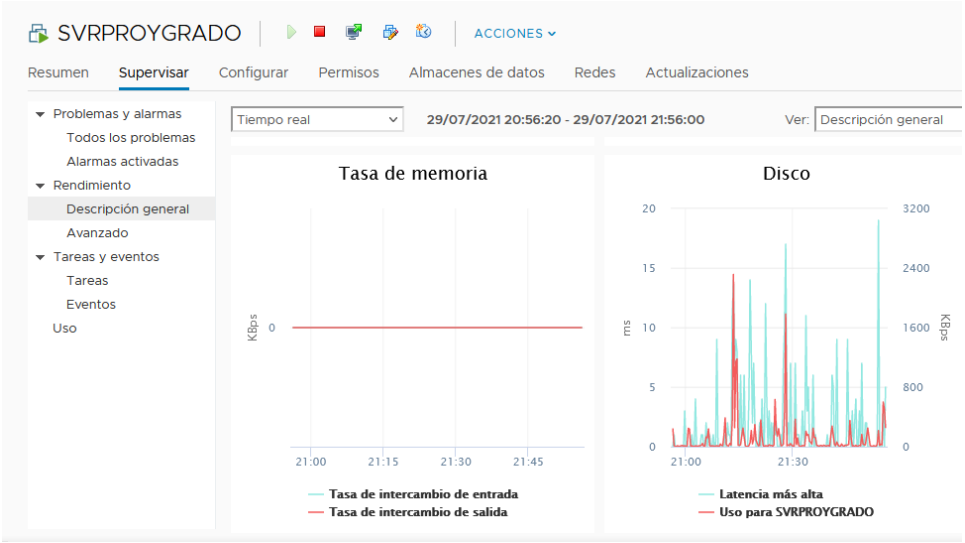
Prueba 2

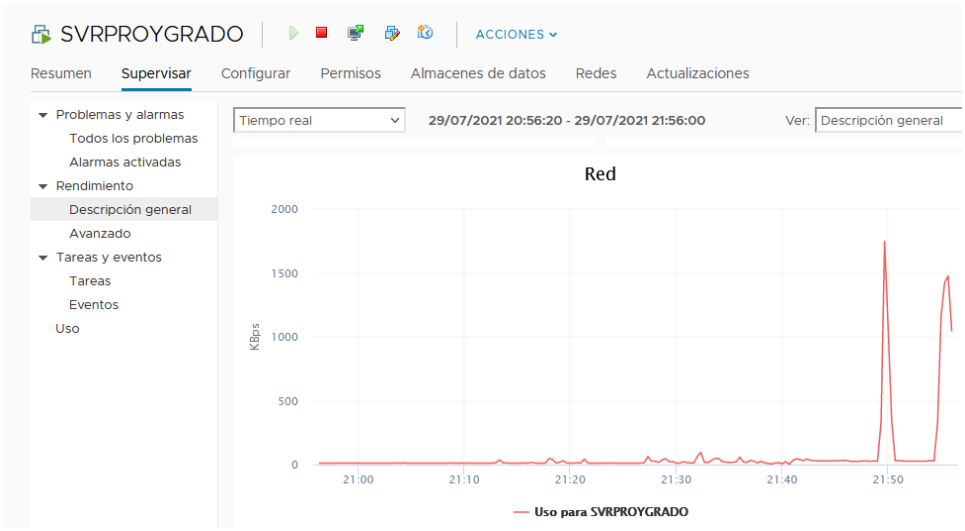


Estado de servidores durante la prueba

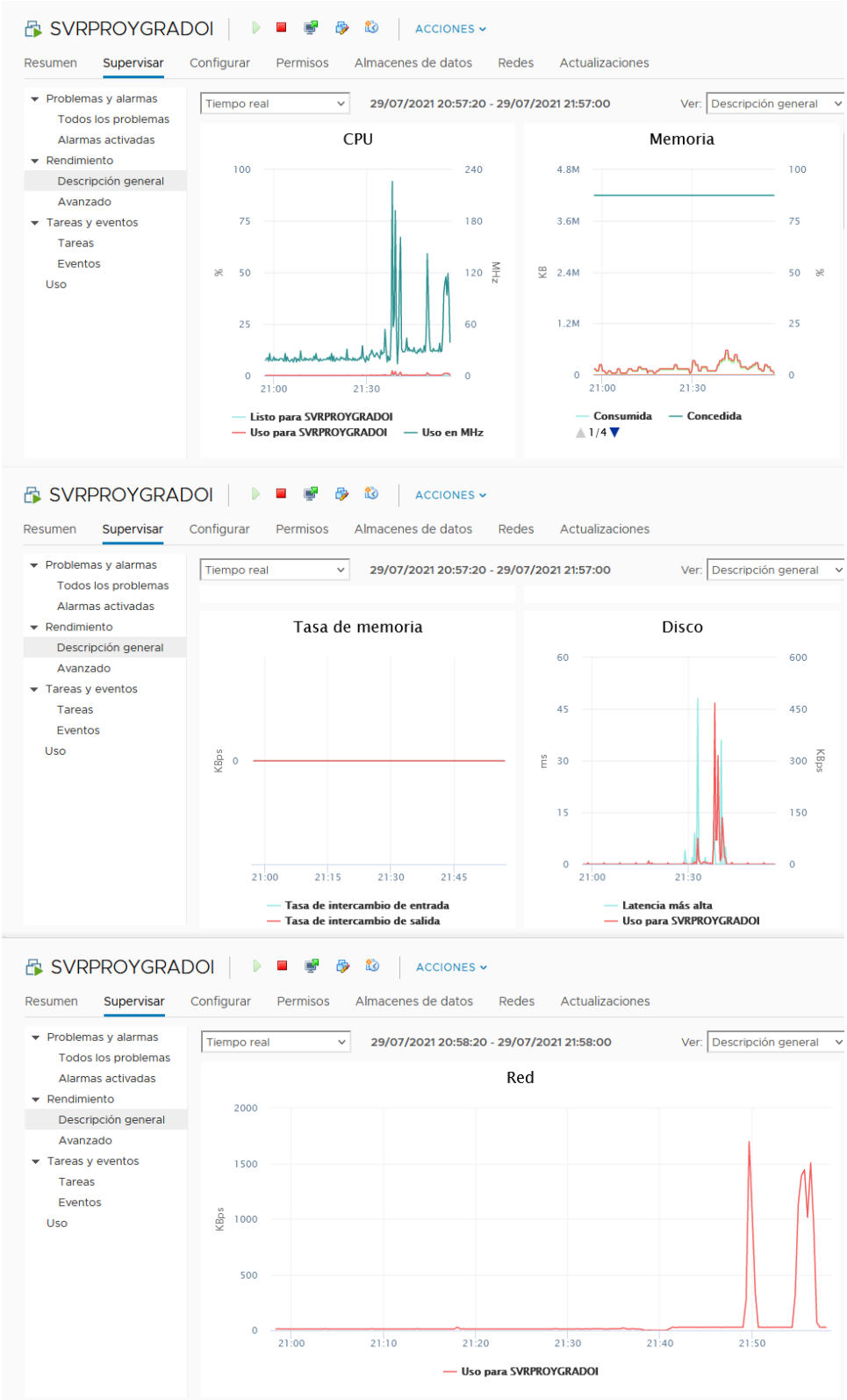
Servidor Windows Server





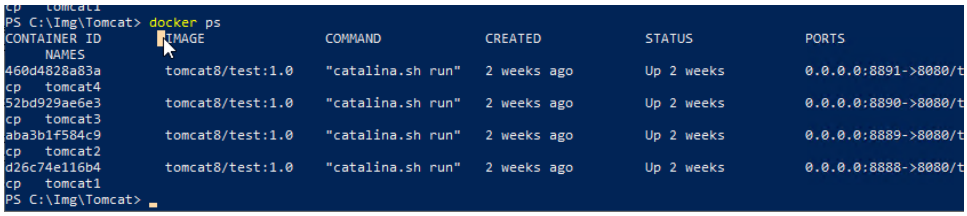


Servidor Linux:


SVRPROYGRADOI
ACCIONES
Resumen
Supervisar
Configurar
Permisos
Almacenes de datos
Redes
Actualizaciones
Problemas y alarmas
Todos los problemas
Alarmas activadas
Rendimiento
Descripción general
Avanzado
Tareas y eventos
Tareas
Eventos
Uso
Tiempo real
29/07/2021 20:57:20 - 29/07/2021 21:57:00
Ver:
Descripción general
CPU
%
MHz
Listo para SVRPROYGRADOI
Uso para SVRPROYGRADOI
Uso en MHz
Memoria
KB
%
Consumida
Concedida
1 / 4
SVRPROYGRADOI
ACCIONES
Resumen
Supervisar
Configurar
Permisos
Almacenes de datos
Redes
Actualizaciones
Problemas y alarmas
Todos los problemas
Alarmas activadas
Rendimiento
Descripción general
Avanzado
Tareas y eventos
Tareas
Eventos
Uso
Tiempo real
29/07/2021 20:57:20 - 29/07/2021 21:57:00
Ver:
Descripción general
Tasa de memoria
KBps
Tasa de intercambio de entrada
Tasa de intercambio de salida
Disco
ms
KBps
Latencia más alta
Uso para SVRPROYGRADOI
SVRPROYGRADOI
ACCIONES
Resumen
Supervisar
Configurar
Permisos
Almacenes de datos
Redes
Actualizaciones
Problemas y alarmas
Todos los problemas
Alarmas activadas
Rendimiento
Descripción general
Avanzado
Tareas y eventos
Tareas
Eventos
Uso
Tiempo real
29/07/2021 20:58:20 - 29/07/2021 21:58:00
Ver:
Descripción general
Red
KBps
Uso para SVRPROYGRADOI

Tener en cuenta las pruebas anteriores

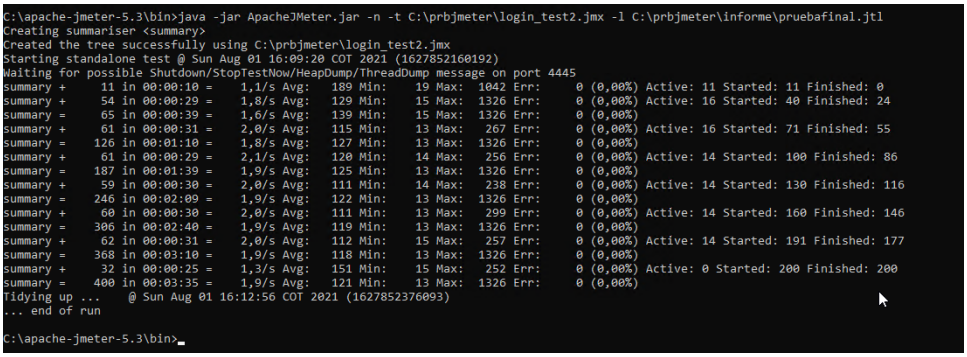
- Se utiliza el mismo servidor Windows Server para realizar las pruebas de carga con JMeter
- Solo estaban operando 4 contenedores Docker por recursos del servidor de pruebas



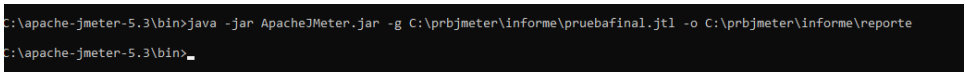
- Esta misma cantidad fue la configurada para balancear en el HAPROXY

Generación de informe

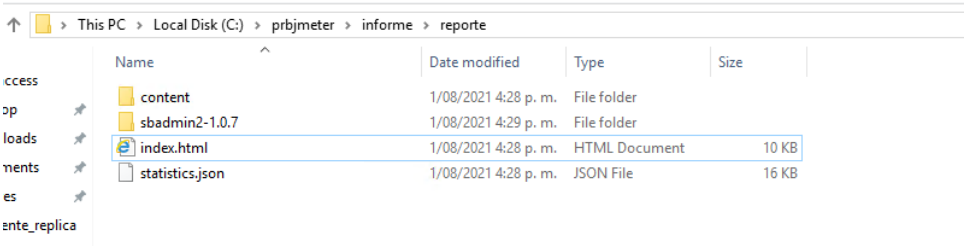
- Se genera el .jtl



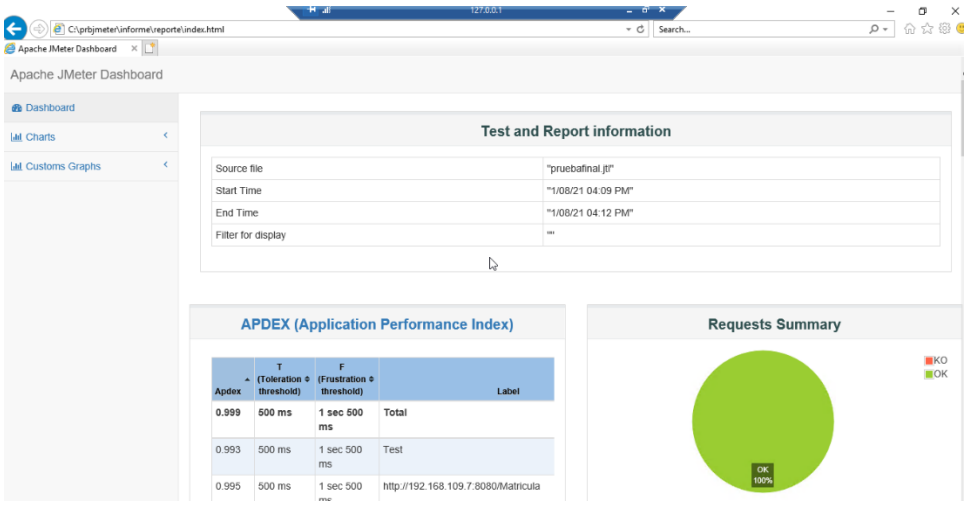
- Se genera reporte



- Ubicación en servidor Windows server (donde se han trabajado las pruebas)



- Reporte


C:\prbjmeter\informe\reporte\index.html
Search...
Apache JMeter Dashboard
Apache JMeter Dashboard
Dashboard
Charts
Customs Graphs
Test and Report information
Source file
"pruebafinal.jtl"
Start Time
"1/08/21 04:09 PM"
End Time
"1/08/21 04:12 PM"
Filter for display
""
APDEX (Application Performance Index)
Apdex
T (Toleration threshold)
F (Frustration threshold)
Label
0.999
500 ms
1 sec 500 ms
Total
0.993
500 ms
1 sec 500 ms
Test
0.995
500 ms
1 sec 500
http://192.168.109.7:8080/Matricula
Requests Summary
KO
OK
OK
100%