Facultad 4

Estrategia para la configuración del despliegue para el comportamiento multi-instancia del XAVIA PACSServer 4.0

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Jorge Yohan Fuentes Alfonso

Tutores: Ing. Damián Socarrás Lima

M.Sc Mailin Carballosa Infante

La Habana, diciembre del 2023

Declaración Jurada de Autoría

Declaro ser autor de la presente tesis que tiene por título: Estrategia de despliegue para el comportamiento multi-instancia del XAVIA PACSServer 4.0 para optar por el título de Ingeniero en Ciencias Informáticas, concediendo a la Universidad de las Ciencias Informáticas y en especial al Centro de Informática Médica la autorización a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los 8 días del mes de diciembre del año 2023.

Autor: Jorge Yohan Fuentes Alfonso

Tutor: Ing. Damián Socarras Lima

Tutor: M.Sc Mailin Carballosa Infante

Datos de contacto

Damián Socarrás Lima, graduado de Ingeniería en Ciencias Informáticas en el año 2020.Se desempeña como desarrollador del sistema XAVIA PACS en el departamento del desarrollo de aplicaciones del Centro de Informática Médica (CESIM).

Correo electrónico: dslima@uci.cu.

Mailin Carballosa Infante graduada de Ingeniería en Ciencias Informáticas en el año 2008. Master en ciencias de Gestión de información en 2012 y profesora de la Facultad 4.

Correo electrónico: minfante@uci.cu.

Agradecimientos Jorge Yohan Fuentes Alfonso

Después de estos años por fin ha llegado el momento de poner fin a esta etapa como estudiante de Ingeniería en Ciencias Informáticas. La cual no puedo finalizar sin antes mostrar mi agradecimiento a todas las personas que han formado parte de ella.

En segundo lugar, agradecer el apoyo incondicional de mi familia, y en especial a mis padres y abuelos por ayudarme con su esfuerzo y ejemplo a conseguir todo lo que me propongo.

En segundo lugar, agradecer a todos los profesores que han estado en las distintas etapas del camino por su labor docente y profesional que han marcado mi aprendizaje en estos 5 años.

Un agradecimiento especial Adolfo por toda su ayuda incondicional y por guiarme por el camino adecuado a la hora de realizar este trabajo.

Y por último y EL MAS IMPORTANTE para Dariannis mi compañera de equipo y tesis. No puedo expresar con palabras lo agradecido que estoy por haberte tenido como compañera y apoyo en cada uno de los trabajos que hicimos como equipo a lo largo del curso. Gracias por estar a mi lado en cada paso del camino hasta el último instante y aunque mi nombre sea el único que aparece como autor tú y yo sabemos que te mereces este trabajo tanto como yo. Gracias por todo.

Resumen

El servidor XAVIA PACSServer 4.0 es un componente esencial de un tipo de sistema de información clínica llamado PACS. El mismo almacena y gestiona la información de los estudios que se generan en las diferentes modalidades diagnósticas, soporta asociaciones simultáneas, así como garantiza el archivo de cada uno de estos estudios de forma ordenada. Además, se instala localmente en una red intrahospitalaria, razón por la cual solo permite configurar una única instancia del sistema, y cuando se instala en la nube solo se puede configurar para un hospital. Esto afecta la flexibilidad, disponibilidad, escalabilidad, de un proceso fácil de actualización y optimización de los recursos. En el presente trabajo se aborda la creación de una estrategia para la configuración de despliegue del sistema XAVIA PACSServer 4.0 con un comportamiento multi-instancia. Para la ejecución del mismo se siguió una estrategia explicativa y se emplearon los métodos: análisis y síntesis, histórico-lógico y modelación. Como ambiente de desarrollo se utilizaron la tecnología *Docker* para la creación y administración de contenedores, *Docker Swarm* para el escalado, la orquestación, la administración del clúster y el balanceo de carga, *Docker Compose* para definir y ejecutar aplicaciones de *Docker* de varios contenedores. Como herramientas Visual *Paradigm* para la creación de diagramas y *Visual Studio Code* como editor de código para la configuración. La estrategia de configuración de despliegue descrita permite desde un servidor central crear múltiples instancias del sistema donde cada centro de salud que requiere el servicio tendrá su propia instancia manteniendo la información aislada.

Palabras clave: contenedores, multi-instancia, servidor.

ABSTRACT

The XAVIA PACSServer 4.0 server is an essential component of a type of clinical information system called PACS. It stores and manages the information of the studies that are generated in the different diagnostic modalities, supports simultaneous associations, as well as guarantees the archiving of each of these studies in an orderly manner. In addition, it is installed locally in an intra-hospital network, which is why it only allows a single instance of the system to be configured, and when installed in the cloud it can only be configured for one hospital. Therefore, it lacks flexibility, availability, scalability, an easy process for updating and optimizing resources. Therefore, this work addresses the creation of a deployment strategy for the XAVIA PACSServer 4.0 system with multi-instance behavior. For its execution, an explanatory strategy was followed and the methods were used: analysis and synthesis, historical-logical and modeling. As a development environment, Docker technology was used for creating and managing containers, Docker Swarm for scaling, orchestration, cluster management and load balancing, Docker Compose for defining and running multi-container Docker applications. Such as Visual Paradigm tools for creating diagrams and Visual Studio Code as a code editor for configuration. The deployment strategy described allows multiple instances of the system to be created from a central server where each health center that requires the service will have its own instance, keeping the information isolated.

***Keywords**: containers, multi-instance, server.*

Índice

Índice de Tablas

Índice de Figuras

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC), han alcanzado un desarrollo incuestionable, ejemplo de esto es la Computación en la Nube (*"Cloud Computing*" en inglés). La misma es considerada como un nuevo paradigma de servicios de las TIC y un reto para el uso de estas al presentar una forma diferente de acceder a las tecnologías, las infraestructuras y los servicios (Alfredo Rodríguez Díaz, 2018). La informática en la nube es la distribución de recursos de las Tecnologías de la Información (TI) bajo demanda a través de Internet mediante un esquema de pago por uso, donde en lugar de comprar, poseer y mantener servidores y centros de datos físicos, puede acceder a servicios tecnológicos, como capacidad informática, almacenamiento y bases de datos, en función de sus necesidades a través de un proveedor de la nube (2023).

Los servidores basados en la nube son infraestructuras de computación que permiten almacenar, procesar y acceder a datos y aplicaciones a través de Internet. En lugar de tener que administrar y mantener su propio hardware, software y recursos de red, los servidores en la nube ofrecen flexibilidad, escalabilidad y acceso remoto. Estos servidores se encuentran alojados en centros de datos aislados y son administrados por proveedores de servicios en la nube. Los usuarios pueden acceder a estos servidores a través de una conexión a Internet y utilizar recursos informáticos, como capacidad de almacenamiento, potencia de procesamiento y servicios de aplicaciones, según sea necesario (2023).

Un servidor en la nube a diferencia de un servidor tradicional se puede compartir con muchos usuarios a través de una plataforma accesible como Internet. Solo una empresa o entidad determinada tiene acceso a un servidor tradicional (dedicado). Aunque los servidores en la nube realizan las mismas funciones que los servidores físicos, estos se hospedan y entregan a través de una red en lugar de configurarse y administrarse en el sitio. Ofrecen capacidad ilimitada de procesos, mientras que los servidores físicos se limitan a su infraestructura o capacidad informática existentes (2023).

El uso de estos servidores basados en la nube se ha hecho evidente en varios sectores de la sociedad como la salud, debido a que es una solución económica conveniente para los usuarios y rentable para los proveedores. Además, aportan una conexión segura y efectiva con una infraestructura de servidores capaz de acrecentarse de forma inmediata en función de las necesidades de servicios que requieran las organizaciones. Uno de los sistemas en este ámbito que han sido basados en la nube son los *Picture Archivan and Communication*

System (PACS), que son sistemas de software que permiten la gestión integral de las imágenes médicas digitales adquiridas mediante diferentes modalidades.

En el caso de los servidores PACS que gestionan grandes volúmenes de datos, estos necesitan un amplio espacio de almacenamiento para guardar varios escaneos de pacientes. En algunos casos los archivos requieren ser guardados durante un período de al menos cinco años, esto provoca que con el tiempo carezcan de espacio en el caso de los PACS tradicionales. Razón por la cual los sistemas PACS en el mundo están optando por una implementación de servidores basados en la nube. En estos no es necesario plantear la reducción del espacio de almacenamiento, dado que es fácil ampliar el almacenamiento basado en la nube de acuerdo con las necesidades del centro de salud. Entre los sistemas de salud que implementan este servicio a nivel internacional se encuentran *PostDICOM*, *Syngo.plaza e IntelliSpace* PACS, entre otros.

En el caso de Cuba, el Estado ha trabajado constantemente para mantener buenos indicadores de salud en la población, desarrollando sus propios sistemas PACS, evitando así la necesidad de importar soluciones extranjeras. El Centro de Informática Médica (CESIM), de la Universidad de las Ciencias Informáticas (UCI), con el propósito de llevar a cabo el proceso de informatización de la sociedad, también desarrolló una solución PACS. Este sistema ha tenido varios nombres como Cassandra PACS, ALAS PACS y XAVIA PACS debido a la estrategia comercial y de comunicación de la UCI (Arturo Orellana García a , Leodan Vega Izaguirre , Gerardo Ceruto Marrero , Arianne Méndez Mederos , Marien Díaz Ruíz, Filiberto López Cossio , Lissette Soto Pelegrín .).

El sistema XAVIA PACS, compuesto por diversas aplicaciones independientes e integrables, incluye el sistema XAVIA PACSServer. Este último es un servidor de imágenes médicas que facilita la gestión del archivo imagenológico de un hospital y se encuentra desplegado en varias instituciones como una aplicación de escritorio. Sin embargo, este sistema presenta ciertas desventajas. Por un lado, el escalado del servidor puede ser difícil y consumir mucho tiempo y recursos si las necesidades de la empresa cambian. Por otro lado, los costos de almacenamiento pueden ser elevados. Además, compartir información entre diferentes clientes en el mismo servidor puede ser un desafío. Para abordar estos problemas, se está desarrollando una nueva versión web del sistema basada en la nube (Centro de Informática Médica, 2017).

El XAVIA PACSServer 4.0 web una vez llevado a la nube solamente permitirá tener una instancia de este y se configurará para un único hospital, instalándose localmente en la red intrahospitalaria. Razón por la cual la actualización, distribución e instalación de este en los

diferentes hospitales resultará un proceso engorroso. Esto requiere que un especialista se desplace a cada uno de los centros médicos donde es solicitado este servicio y realice la instalación. Además de tratar con personal no especializado en el tema, a los cuales se les dificulta identificar o proporcionar al ingeniero o administrador encargado de las posibles reparaciones, la información necesaria sobre el estado del sistema. Razón por la cual un especialista debe ir en varias ocasiones al centro donde radica el problema para dar solución a este. En caso de que el especialista por razones ajenas a él no puede estar de forma presencial en esa institución debe indicar a la persona relacionada con el proceso informático del lugar, los pasos mediante los cuales puede dar solución al problema demorando este proceso varias horas en comparación con el tiempo que consumiría un especialista. Además, existen centros de salud que carecen de la infraestructura y tecnologías necesarias para hacer uso de este sistema.

Teniendo en cuenta lo antes analizado, se define como **problema de investigación** ¿Cómo mejorar el proceso de despliegue del sistema Xavia PACSServer?

Se propone como **objeto de estudio:** Proceso de despliegue en sistemas distribuidos.

El **campo de acción:** Proceso de despliegue del sistema XAVIA PACServer 4.0.

Se define como **objetivo general:** Diseñar una estrategia de configuración del despliegue para el sistema Xavia PACSServer 4.0 que permita el comportamiento multi-instancia.

Para dar cumplimiento al objetivo planteado se trazan las siguientes tareas de investigación:

- Análisis de las tendencias actuales de sistemas distribuidos en el ámbito internacional.
- Análisis de estrategias de despliegue de aplicaciones. Realización de los artefactos ingenieriles asociados al comportamiento multi-instancia de un servidor web.
- Diseño de la estrategia de configuración del despliegue para el sistema Xavia PACSServer 4.0 para un comportamiento multi-instancia.
- Aplicación de una estrategia de pruebas para la propuesta de solución.

Los métodos científicos utilizados en la investigación fueron:

Teóricos:

- Análisis y Síntesis: se utilizó para la investigación sobre los métodos de comunicación de servidores web existentes basados en la nube, determinando así los elementos que lo componen, las funciones que cumplen y su importancia, para seleccionar los parámetros que se deben tener en cuenta para la obtención de los métodos a utilizar.

- Histórico-Lógico: se utilizó para la investigación de los métodos de comunicación de los servidores con prestación de servicios para imágenes de tipo web. Analizar las formas de solución en estudios anteriores a problemas similares al planteado, así como los sistemas existentes que prestan este servicio.
- Modelación: para la modelación de los diagramas y modelos correspondientes al proceso de desarrollo de la arquitectura multi-instancia.

Empíricos:

- Pruebas: al someter los resultados obtenidos a pruebas de estrés y carga para determinar si el comportamiento del sistema una vez aplicada la multi-instancia cumple con los parámetros establecidos.

El presente trabajo de diploma está compuesto por tres capítulos:

Capítulo 1: Fundamentación teórica. En este capítulo se hace referencia a los fundamentos teóricos en los cuales se basa la investigación, incluye un análisis del estado del arte de la arquitectura multi-instancia, así como los lenguajes, metodología, herramientas y tecnologías seleccionadas para el desarrollo de la solución.

Capítulo 2: Propuesta de solución. En este capítulo se describe la solución propuesta para llevar a cabo la creación de una estrategia para la configuración del despliegue para el comportamiento multi-instancia del XAVIA PACSServer 4.0. Se especifican los requisitos no funcionales, además, de una descripción y configuración de la arquitectura multi-instancia así como la propuesta de despliegue para el sistema.

Capítulo 3: Implementación y Prueba. Se aplica la estrategia de prueba a la propuesta de solución. Se realizan pruebas de rendimiento como carga y estrés que permiten comprobar el correcto desempeño. Además, se detallan las pruebas realizadas a la aplicación luego de finalizadas las mismas.

Capítulo 1: Fundamentación teórica

En el presente capítulo se caracteriza la computación en la nube, así como la arquitectura multi-instancia, resaltando los beneficios que trae la aplicación de dicha arquitectura al sistema PACSServer 4.0. Se identificarán las tendencias, técnicas y tecnologías usadas en la actualidad que sirvieron de apoyo para la solución del problema. Además, se describen las diferentes herramientas y la metodología de desarrollo de software empleada.

1.1 Modelo multi-instancia

La multi-instancia es una propiedad de la nube que se implementa en un modelo que se basa en el concepto de instancia. Una instancia se un conjunto de usuarios que comparten una serie de requisitos y necesidades (Yépez, 2017), como por ejemplo una organización.

Las aplicaciones orientadas a servicios tradicionales, normalmente, dedican una aplicación por instancia. Es decir se crea una instancia personalizada en base a sus necesidades. Sin embargo los proveedores de Software como servicio están optando por una arquitectura multi-instancia debido a que estas instancias comparten o podrían compartir una misma estructura, desde recursos de hardware, hasta sistema operativo, bases de datos o lógica de negocio.

La arquitectura multi-instancia se refiere a un principio de arquitectura de software, donde un sistema comparte varios clientes, es decir que trabaja sobre una misma instancia de software, la cual se ejecuta desde un servidor (2023) (2021). Este concepto se implementa debido a la computación en la nube, dado que los proveedores de software se enfrentan a decisiones sobre cómo modificar la arquitectura de su software para competir en este mercado emergente.

Con una arquitectura multi-instancia, una aplicación de software está diseñada para particionar sus datos y la configuración, de manera que cada cliente trabaje con una instancia de la aplicación virtual personalizada (Fernandez, 2022). Cada cliente comparte los mismos recursos, sin embargo, los datos de cada cliente están separados y seguros lo que significa que un cliente no puede acceder a los datos de otro (Vega, 2018) (Sharma, 2022).

Algunos de los beneficios de la multi-instancia son:

- Disponibilidad: si hay un problema con una instancia, las demás instancias continuarán operando. Esto reduce el riesgo de tener una aplicación fuera de servicio.
- Escalabilidad horizontal: se puede incrementar la capacidad simplemente agregando más instancias. Cada instancia maneja una parte de la carga, permitiendo escalar la aplicación por separado. Este tipo de escalabilidad se basa en la modularidad de su funcionalidad, potenciando el rendimiento del sistema desde un aspecto de mejora global, a diferencia de aumentar la potencia de una única parte del mismo.
- Seguridad: si hay una vulnerabilidad o ataque en una instancia, las demás instancias no se ven comprometidas. Se pueden aislar instancias comprometidas rápidamente.
- Zonificación: las instancias se pueden replicar por zonas geográficas reduciendo latencia.
- Mantenibilidad: puede actualizarse, parchearse o repararse una instancia a la vez, sin afectar a las demás disminuyendo el impacto en el servicio (2021). A continuación se presenta una imagen que representa la arquitectura multi-instancia.

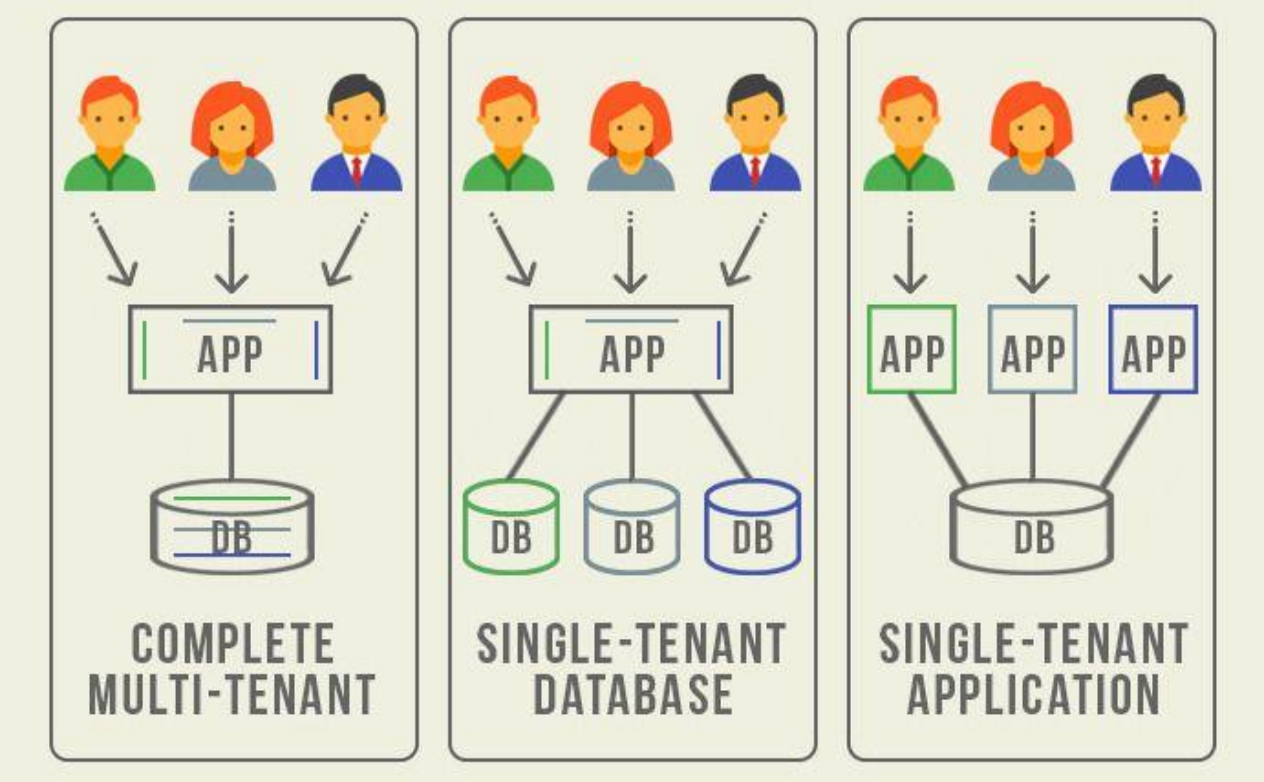


Figura 1: Tipos de arquitectura Multi-Tenant (Fuente: Tomada de Arquitectura de Multi-Tenant (Leifer Mendez, 2022)).

A continuación se presenta una tabla comparativa donde se caracterizan los tipos de arquitectura multo-instancia.

Tabla 1: Caracterización de los tipos de multi-instancia.

Tipo de inquilino	Descripción	Ventajas	Desventajas
Inquilino completo (*Complete multi-tenant*)	Una sola instancia de la aplicación y una sola base de datos se utilizan para servir a múltiples clientes o usuarios.	Mayor eficiencia y menor costo de mantenimiento.	Menor personalización y flexibilidad.
Base de datos de inquilino único (*Single-tenant database*)	Cada cliente o usuario tiene su propia instancia de la base de datos, pero comparte la misma instancia de software.	Mayor privacidad y seguridad.	Mayor complejidad y costo de mantenimiento.
Aplicación de inquilino único (*Single-tenant application*)	Cada cliente o usuario tiene su propia instancia de la aplicación y la base de datos.	Mayor personalización y flexibilidad.	Mayor complejidad y costo de mantenimiento.

Luego de la caracterización realizada en la Tabla 1 se puede apreciar los beneficios que implica el uso de la arquitectura multi-instancia, donde se identifica que el modelo que mejor se ajusta a las necesidades del cliente es *Complete multi-tenant*.

El modelo *multi-tenant* o multi-instancia es un enfoque para prestar servicios de software en la nube. Su principal ventaja es que permite a varias organizaciones (*tenant*) compartir una única instancia de software, lo que optimiza el uso de recursos. Es importante distinguir entre los usuarios y los *tenant*. Los usuarios son las personas que acceden a la aplicación, mientras que los *tenant* son las organizaciones que contratan el servicio. La Figura 2 muestra esta diferencia (Yépez, 2017).

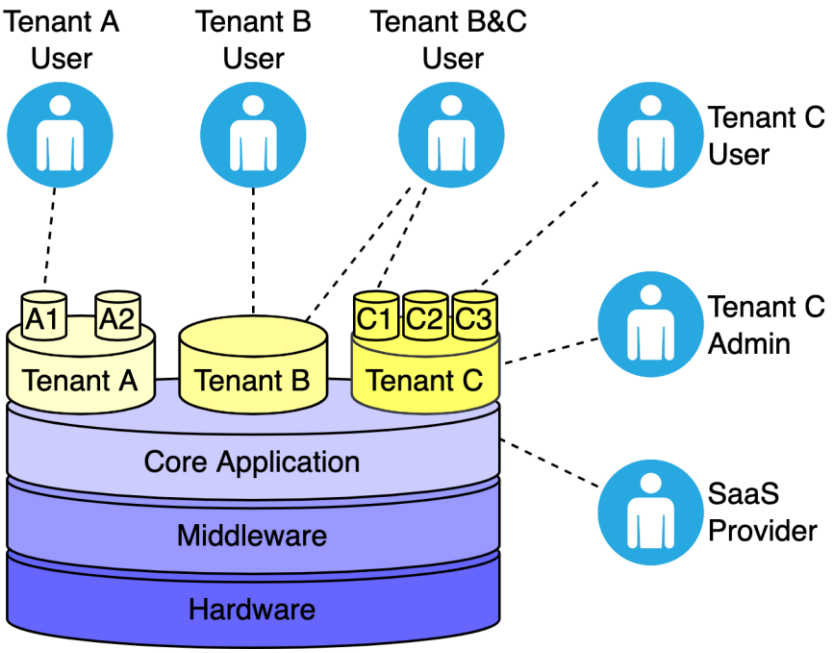


Figura 2: Arquitectura por capas del modelo multi-instancia (Fuente Elaboración propia).

En general, el modelo *multi-tenant* es una opción popular para las empresas que buscan reducir costos y mejorar la eficiencia operativa. Y a pesar de que este modelo no es adecuado para todas las aplicaciones y situaciones, por sus características es la opción adecuada para empresas que prestan servicios informáticos.

1.2 Computación en la nube (*Cloud Computing*)

La *cloud computing* es un paradigma que permite ofrecer servicios de computación a través de la red (Internet). Se trata de un modelo de prestación de servicios de negocios y tecnología que permite a los usuarios el acceso a un conjunto de recursos compartidos y modificables (redes, servidores, bases de datos, aplicaciones, entre otros). Estos recursos pueden ser asignados y liberados de manera casi autónoma sin depender del proveedor del servicio (Yépez, 2017).

Al optar por una solución de *cloud computing*, los datos, aplicaciones y cargas de trabajo se mueven de un centro de datos que se encuentra físicamente en las instalaciones de una organización (*on-premise en inglés)* a un centro de datos remoto gestionado por un proveedor de servicios. Considerado un modelo innovador, permite el abastecimiento de servicios TI que generan valor para las empresas. La adopción de la computación en la nube está creciendo de manera exponencial debido a la flexibilidad, agilidad, escalabilidad y simplicidad que ofrece a las empresas (Yépez, 2017) (Grupo Aire, 2023).

En el caso del servidor *cloud* es un servidor virtual que, al igual que los servidores tradicionales, proporciona mayor procesamiento y almacenamiento de datos, disponibles a través de una red pública o privada, con IPv4 o IPv6, y admite el acceso remoto. Estos servidores están virtualizados con un hipervisor que actúa como una capa de virtualización de software que permite crear y ejecutar múltiples máquinas virtuales dentro de un único servidor, así como diferentes sistemas operativos. Separa los recursos de la máquina virtual del sistema de hardware y permite distribuirlos (Grupo Aires, 2023).

Modelos de servicios para la computación en la nube:

Los servicios ofrecidos mediante el enfoque de *cloud computing* pueden ser diversificados en tres clases, conformando un modelo de tres capas, en el que cada una puede ser implementada utilizando los servicios de la capa inferior. El servicio más simple que puede ser ofrecido es la infraestructura como servicio, en segundo nivel se ofrece la plataforma de desarrollo como servicio y en un tercer nivel, las aplicaciones pueden ser ofrecidas como servicio (Bach. Rosales Torres, 2022).

- **Infraestructura como servicio (IaaS):** proporciona infraestructura para procesamiento, almacenamiento, redes y otros elementos sobre los cuales los clientes ejecutan sus sistemas operativos o aplicaciones. IaaS es la base del servicio y contiene los bloques fundamentales para la TI en la nube, pues proporciona los servicios y estructuras necesarios sobre los cuales se construyen los otros servicios. También ofrece servicios de virtualización como máquinas virtuales, cortafuegos, sistemas de backups o balanceadores de carga (2022) (2020).
- **Plataforma como servicio (PaaS):** es un entorno de desarrollo e implementación completo en la nube, con recursos que permiten entregar desde aplicaciones sencillas hasta aplicaciones empresariales sofisticadas basadas en la nube. En este nivel los proveedores de *cloud computing* ofrecen un entorno de desarrollo para que el usuario pueda crear y alojar sus propias aplicaciones y distribuirlas como servicio sin tener que preocuparse de la infraestructura que necesita. Al proporcionar infraestructura como servicio, PaaS aporta las mismas ventajas que IaaS, pero las características adicionales, como herramientas de desarrollo y otras herramientas empresariales, ofrecen mayores ventajas (Bach. Rosales Torres, 2022) (2023).
- **Software como servicio (SaaS):** en este nivel las aplicaciones son distribuidas como servicios y accedidos por demanda. Los usuarios no necesitan mantener infraestructura propia ni instalar software, porque la aplicación y sus datos asociados son accedidos por medio de Internet, mediante un navegador que puede ejecutar en un cliente ligero. Esta infraestructura, aloja el sistema de la empresa, así como sus datos, en servidores

externos a la misma, y se paga una cuota por su utilización (Bach. Rosales Torres, 2022) (2020).

Los servicios facilitan el flujo de datos de los usuarios a través de Internet, desde los clientes *frontend* hasta los sistemas de los proveedores, y viceversa. La diferencia entre estos tres modelos de servicios de *cloud computing* radica en las responsabilidades que tengan el proveedor de servicios y el usuario.

En cuanto a la configuración de la multi-instancia, es posible implementarla en los tres modelos de servicios para la computación en la nube. Sin embargo, la configuración de la multi-instancia en el modelo de SaaS es más común debido a su arquitectura de *multi-tenent*. Además, el servicio es accedido por los clientes a través de internet. Este servicio es prestado por una empresa y el cliente paga una cuota por su utilización. Por esta razón se empleará este modelo en la solución propuesta.

Modelos de despliegue en la nube:

Una infraestructura *cloud* puede estar desplegada siguiendo cuatro modelos distintos: (Moreno, 2016)

- **Nube privada** u *on-premise* en inglés es utilizada exclusivamente por una única organización, con múltiples consumidores. Aunque la organización ejecuta sus cargas de trabajo como un modelo privado, puede haber un tercero que administre, y el servidor puede alojarse tanto externamente como en las instalaciones de la empresa usuaria. En cuanto al acceso a la información, solo un conjunto de personas claramente definido tiene permiso para acceder a la información guardada en una nube privada, lo cual impide que cualquier persona pueda hacer uso de ella.
- **Nube *community*** esta infraestructura es utilizada únicamente por una comunidad de consumidores con objetivos en comunes. En el caso de una comunidad, varias organizaciones con entornos similares comparten la infraestructura y los recursos relacionados. Como las organizaciones tienen requisitos de seguridad, privacidad y rendimiento uniformes, esta arquitectura de centro de datos de múltiples inquilinos ayuda a las empresas a lograr sus objetivos específicos. Es por ello que un modelo comunitario es particularmente adecuado para organizaciones que trabajan en proyectos conjuntos. En ese caso, una nube centralizada facilita el desarrollo, la gestión y la implementación del proyecto.
- **Nube pública** se despliega para ser usada libremente por el público en general. Suele estar gestionada por una empresa o un organismo académico o gubernamental.

- **Nube híbrida** es una combinación de dos o más de los modelos anteriores, debidamente interconectados por un proveedor que habilita el paso de datos y aplicaciones entre ellas.

El sistema diseñado maneja información delicada que debe ser correctamente protegida, además de que solo será usada por instituciones de salud y dentro de esta solo un conjunto de personas claramente definido tiene permiso para acceder a la información guardada. Por esta razón, de los diferentes tipos de despliegues existentes en la nube se selecciona un modelo de nube privada.

1.3 Estrategias de despliegue

Las estrategias de despliegue se definen como la opción de modificar o actualizar una determinada aplicación. Idealmente, se busca llevar a cabo las transformaciones en el sistema sin tener periodos de inactividad, para que el cliente no se vea afectado por este. Además, estas estrategias de despliegue destacan por contribuir a que los despliegues se realicen de manera fiable y ágil, mediante procesos sencillos, continuos y gestionados. Es importante destacar también que existen múltiples tipos de estrategias de despliegue que se usan de acuerdo con las necesidades de la aplicación o el usuario. El método de IBM y el de Métrica III son de los mejor especificados (Panizzi, 2022). Por su nivel de detalle son analizados a continuación.

IBM

El método de despliegue: El método de IBM consta de 3 fases y 11 etapas que, aunque se analicen de forma lineal, forman parte de un ciclo en el que estos pasos pueden alternarse o repetirse.

Fase 0: Preparación del despliegue: El objetivo principal de esta fase es preparar el escenario requerido para un despliegue exitoso. Los pasos a seguir en esta fase son:
- Conformación del equipo de trabajo.
- Revisión de la documentación del despliegue.
- Desarrollo de un plan de despliegue de alto nivel.
- Establecer la dirección del despliegue (Equipo de Dirección).

Fase 1: Refinamiento y divulgación: Esta fase comienza después de haberse comprobado los cambios que pueden haberse realizado durante la última etapa de las negociaciones. Además, durante esta fase se refina el plan desarrollado en la fase anterior y se realizan las

reuniones de coordinación correspondientes a la preparación para el comienzo del despliegue. Los pasos a realizar son:
- Refinar el plan de despliegue.
- Concluir este plan.
- Realización de reuniones de coordinación del despliegue.

Fase 2: Despliegue del software: El propósito es comenzar a ejecutar el despliegue a partir del plan establecido. Comienza con la selección cuidadosa de las tareas de más rápido cumplimiento, moviéndose hacia las demás. Durante esta fase la gestión de proyectos es crítica. Incluye los siguientes pasos:
- Alcanzar las metas urgentes del despliegue.
- Ejecutar el plan de despliegue.
- Identificar nuevas necesidades del negocio.
- Actualizar el plan de despliegue.

Según IBM, para lograr el éxito en el despliegue de un software deben trabajar conjuntamente individuos de la compañía cliente y asociados (Cáceres, 2011) (Hernández, 2012).

Métrica III

Define el proceso de Implantación y Aceptación con el objetivo de asegurar la entrega y aceptación de un sistema, y la realización de todas las actividades para la puesta en producción de este. También se apoya en la experiencia de los usuarios para resolver problemas detectados en versiones anteriores del sistema. La implantación puede ser un proceso iterativo que se realiza de acuerdo a un plan previamente establecido cuya elaboración tiene en cuenta:

- El cumplimiento de los requisitos de implantación definidos en la actividad Establecimiento de Requisitos y especificados en la actividad Establecimiento de Requisitos de Implantación.

- La estrategia de transición del sistema antiguo al nuevo.

- Las principales actividades definidas para esta fase de Métrica III son:

- Establecimiento del Plan de implantación, formación necesaria para la implantación.

- Incorporación del sistema al entorno de operación, carga de datos al entorno de operación.
- Pruebas de implantación del sistema, pruebas de aceptación del sistema.

- Preparación del mantenimiento, establecimiento del acuerdo de nivel de servicio.

- Presentación y aprobación del sistema, paso a la producción.

Cada una de estas actividades contiene un grupo de tareas bien definidas para asegurar el cumplimiento de los objetivos de este proceso, además de establecer los roles y equipos que participan, así como sus responsabilidades (Cáceres, 2011).

En las diferentes estrategias para el despliegue de soluciones informáticas estudiadas, se plantean una serie de fases o etapas para llevar a cabo el despliegue del sistema en una organización determinada. En sentido general se establecen acciones como el diagnóstico de la organización, el desarrollo de despliegue piloto, configuración de la infraestructura automatización y configuración del entorno.

Debido a que el objetivo de la presente investigación está enfocado a que la aplicación XAVIA PACSServer 4.0 se pueda desplegar en varias instancias pero haciendo uso de una infraestructura común, que no depende de las características de la organización en sí, se determina que la fase en la que incide este trabajo en el despliegue, a partir de un proceso de configuración con la utilización de herramientas de contenerización, como *Docker* o *PodMan*.

1.4 Sistemas homólogos

El avance de la era digital ha llevado a un proceso de modernización en todo el mundo, lo que ha resultado en la integración de las TIC en prácticamente todas las áreas de la sociedad. Esto trae consigo un gran número de sistemas existentes basados en la nube, y su capacidad para ofrecer soluciones versátiles, escalables, con acceso remoto y seguridad avanzada. En este contexto, las empresas han tenido que adaptar sus procesos internos a las nuevas tecnologías. Para enfrentar la constante evolución y actualización en este sector, se implementaron sistemas con arquitecturas como la multi-instancia.
Entre los ejemplos más populares de aplicaciones basadas en el modelo multi-instancia se tiene *Slack, HubSpot, Salesforce, Shopify, Gmail, Drupal, Oracle Database y Dropbox* (Sharma, 2022). Tomándose como homólogos *Salesforce, Drupal, Oracle Database y Dropbox,* por ser los que más información aportan a la investigación.

Dropbox

Es una plataforma de almacenamiento en la nube que permite a los usuarios guardar y compartir archivos y carpetas con otros usuarios. Los archivos se pueden sincronizar automáticamente entre dispositivos y se pueden acceder a ellos desde cualquier lugar con

conexión a Internet. Es compatible con una amplia variedad de sistemas operativos, incluyendo *Windows, macOS, Linux, iOS y Android*. Además, *Dropbox* es multi-instancia, lo que significa que se puede tener varias cuentas de *Dropbox* en un solo dispositivo. Esto puede ser útil si se desea separar los archivos personales de los archivos de trabajo, por ejemplo. También se pueden vincular varias cuentas de *Dropbox* para acceder a ellas desde una sola cuenta (Delgado, 2020).

Dropbox utiliza *Docker* como herramienta para la creación de contenedores. De hecho, hay varios repositorios de *Docker* disponibles en línea que permiten a los usuarios ejecutar Dropbox en un contenedor *Docker*. EL repositorio de *Docker* de *otherguy* proporciona una imagen de *Docker* para *Dropbox* que se puede ejecutar en un contenedor *Docker*. Para la orquestación y balanceo de carga de los contenedores, *Dropbox* se apoya en *Swarm,* herramienta que permite agrupar una serie de hosts *Docker* en un clúster y gestionar de forma centralizada.

Drupal

Drupal es un sistema de gestión de contenido (CMS) de código abierto que permite a los usuarios crear y administrar sitios y aplicaciones web de manera flexible y escalable (Cuervo, 2020).

Una instalación multisitio (multi-instancia) permite que una sola instancia de *Drupal* (código base idéntico) sirva a varios sitios con diferentes nombres de dominio. Cada uno con sus propios módulos y temas activos (2022). Al utilizar este tipo de configuración se ahorra tiempo al administrar más de un sitio (2021). Una ventaja de este enfoque es que no requiere que todos los sitios ejecuten la misma versión de *Drupal.* Si una organización está estandarizando *Drupal,* pero no una implementación específica, con el tiempo puede crear una colección de sitios administrados por diferentes departamentos en *Drupal* 6, *Drupal* 7 y, eventualmente, *Drupal* 8 (Larry Garfield, 2012).

Con este enfoque, cada sitio podría residir en un dominio separado, pero también podría instalarse en un subdirectorio de un único dominio. Dando la perspectiva del creador de los sitios, es casi lo mismo que tener instalaciones separadas. Constituye una ventaja para los administradores del servidor pues cada vez que se lanza una nueva actualización principal de *Drupal*, solo tendrá que realizar esa actualización en un conjunto de código base en lugar de hacerlo para cada sitio.

Oracle Database

Es un sistema de gestión de bases de datos relacionales que ofrece múltiples características y funcionalidades para almacenar, procesar y analizar grandes volúmenes

de datos. *Oracle Database* permite crear múltiples instancias de bases de datos en un solo servidor o en varios servidores. Esto permite optimizar el uso de los recursos, mejorar la disponibilidad y el balanceo de carga, y facilitar la administración y el mantenimiento. *Oracle Database* emplea una arquitectura cliente-servidor, con el servidor de la base de datos y las conexiones del cliente ejecutándose en procesos separados. El proceso del servidor gestiona datos y recursos, y las conexiones del cliente se comunican con el servidor para acceder y manipular los datos.

La multi-instancia de *Oracle* es una característica que permite tener varias instancias de bases de datos en un solo servidor o en varios servidores. Cada vez que se inicia una instancia de base de datos, se asigna algo de memoria y esa memoria se denomina SGA (Área Global del Sistema). Junto con la asignación de memoria, lo harán uno o más procesos en segundo plano. SGA se utiliza para almacenar datos y controlar información sobre una instancia de base de datos a través de sus diversos subcomponentes, donde cada componente está dedicado a un propósito específico (2023).

La multi-instancia de *Oracle* se puede implementar de dos formas:

- *Oracle Real Application Clusters* (RAC): Posee un comportamiento que permite que varias instancias en diferentes servidores accedan a una misma base de datos almacenada en un sistema de archivos compartido o en un dispositivo de almacenamiento externo. Esto permite escalar horizontalmente la capacidad de procesamiento y garantizar la continuidad del servicio en caso de fallo de un servidor (2023) (Wheeler, 2019).
- *Oracle* Multi-instancia: Es una arquitectura que permite que varias bases de datos, llamadas contenedores *pluggable* (PDB), compartan una misma instancia y un mismo conjunto de archivos, llamado contenedor raíz (CDB). Esto permite reducir el consumo de recursos, simplificar las operaciones y facilitar la migración y consolidación de bases de datos (2023) (Wheeler, 2020).

Salesforce

Es una plataforma de gestión de relaciones con los clientes (conocida como CRM por sus siglas en inglés, *Customer Relationship Management*) basada en la nube que permite a las empresas gestionar sus contactos, oportunidades, campañas y más. Además, proporciona a todos los departamentos de su organización, incluidos los de marketing, ventas, servicio al cliente y comercio electrónico, una visión unificada de sus clientes en una plataforma integrada (2022).

Salesforce es uno de los CRM más innovador del mercado dado a su apuesta por la multi-instancia, debido a que el proveedor del servicio en cuestión (*Salesforce* en el caso del CRM) centra todos sus esfuerzos en el mantenimiento de una sola versión de su software, en lugar de trabajar en el desarrollo, implementación y mantenimiento de múltiples versiones personalizadas instaladas en las máquinas de cada uno de los clientes (2023).

Esta plataforma utiliza la arquitectura multi-instancia basada en el concepto de organizaciones, que son entidades lógicas. Cada organización es un inquilino que tiene su propio esquema, metadatos, lógica del negocio y datos. Todas las organizaciones comparten una misma instancia de la aplicación que se ejecuta en un conjunto de servidores físicos llamados *Pod*.

Tabla 2: Comparación de los sistemas homólogos.

No	Parámetro	Oracle Database	Drupal	*Dropbox*	*PACSServer*	*Salesforce*
1	multi-instancia	Este sistema permite que varias instancias en diferentes servidores accedan a una misma base de datos almacenada en un sistema de archivos compartido o en un dispositivo de almacenamiento externo.	Cada instancia de *Drupal* puede ejecutar de forma independiente con su propia base de datos, configuración y recursos, mientras hace uso en ambos casos del mismo servidor.	Permite tener varias cuentas de Dropbox en un solo dispositivo.	Permite crear una instancia del sistema para cada una de las entidades de manera independiente donde los recursos no son compartidos.	Este sistema permite que cada cliente tenga su propia instancia separada y segura en la plataforma, pero todos utilizando la misma infraestructura subyacente proporcionada por Salesforce.

2	Personalización de las instancias.	Oracle Database permite la personalización de cada una de sus instancias. Las propiedades de una instancia de Oracle se especifican utilizando parámetros de inicialización. Cuando se inicia la instancia, se lee un archivo de parámetros de inicialización y la instancia se configura en consecuencia.	En *Drupal* cada una de las instancias se encuentra personalizada.	El menú de preferencias permite personalizar la aplicación de Dropbox a gusto. Aunque este varía en función del sistema operativo.	Este sistema permite personalizar cada una de sus instancias atendiendo el servicio que ofrece o necesidades del cliente.	Salesforce permite a cada inquilino personalizar fácil y rápidamente sus aplicaciones mediante metadatos, datos que describen elementos como la interfaz de usuario (UI) y la lógica empresarial.
3	Escalabilidad y flexibilidad	Es un sistema flexible que permite un escalado horizontal de la capacidad de procesamiento y garantiza la continuidad del servicio en caso de fallo de un servidor.	*Drupal* es escalable y se puede utilizar para sitios web grandes y complejos con una gran cantidad de contenido. Además, es flexible y modular lo cual	Este modelo de servicio en la nube ofrece flexibilidad y escalabilidad.	El PACSServer 4.0 al ser un sistema basado en la nube con una arquitectura multi-instancia es un sistema flexible y que escaló en poder	*Salesforce* gracias a su implantación *cloud*, permite a los negocios un crecimiento y cambio constante. Además de un servicio robusto,

			es uno de sus principios fundamentales.		computacional, así como en rendimiento mediante la escalabilidad horizontal.	adaptable y confiable.
4	Tecnologías	*Docker* para empaquetar la aplicación en contenedores y *Kubernetes* como orquestador para balancear la carga.	*Docker* para empaquetar la aplicación en contenedores. *Kubernetes* y *OpenStack* para balancear la carga.	*Docker* para crear los contenedores. *Docker Swarm* para orquestar los contenedores y balancear la carga.	*Docker* para empaquetar la aplicación en contenedores. *Docker Swarm* para orquestar los contenedores y balancear la carga.	*Docker* para el trabajo con contenedores y *Kubernetes* como orquestador y balancear la carga.

En la Tabla 2 se establece una comparación entre los sistemas homólogos seleccionados. En esta se tienen en cuenta parámetros como la multi-instancia, personalización de las instancias, escalabilidad, flexibilidad y tecnologías usadas. Estos son aspectos indispensables para establecer una comparación con el sistema XAVIA PACSServer 4.0. Otro aspecto relevante que se valoró es que su implementación esté basada en la nube.

A pesar de que cada uno de estos sistemas desarrollan la multi-instancia, no se tuvieron en cuenta su configuración para el posterior desarrollo del sistema XAVIA PACSServer 4.0. Sin embargo se tienen en cuenta las tecnologías utilizadas.

En el caso de las tecnologías en su mayoría son sistemas que optan por excelentes propuestas como es el caso de *Docker, Docker Swarm y Kubernetes*. Debido a que todos estos sistemas usan *Docker* en el trabajo con contenedores se decide seleccionar dicha herramienta para implementar la *multi-tenant* para el XAVIA PACSServer 4.0 se hará uso en este caso de *Docker* para automatizar el despliegue de aplicaciones dentro de contenedores. Este software permitirá gestionar contenedores sobre distintos sistemas operativos, debido a que funciona en varias plataformas. Y como balanceador de carga

Docker Swarm que fue creado por los mismos desarrolladores de *Docker* para poder agrupar una serie de *hosts* de *Docker* en un *clúster* y gestionar así los *clústeres* de forma centralizada, además de orquestar los contenedores. Por lo que se tomó como referente Dropbox de las soluciones analizadas.

Existen varias razones por las que no se hace uso de *Kubernetes* y se opta por *Docker Swarm*, aunque ha resultado mediante los sistemas analizados más que evidente la excelencia y ventajas que implica usar esta tecnología. En el caso de *Docker Swarm,* es el que mejor se ajusta a los requisitos no funcionales del proyecto que se está realizando, dado que el XAVIA PACSServer 4.0 no es una aplicación de alta demanda con una configuración compleja y son limitados los servicios que presta. Además, *Docker Swarm* es más rápido en cuanto al despliegue y ampliación de contenedores, con balanceo de carga automático.

1.5 Herramientas CASE

A medida que pasa el tiempo se logra entender que el empleo del software es una buena opción para agilizar y sistematizar las tareas en el desarrollo de procesos. El desarrollo de software no es la excepción; en este caso dichas herramientas se han denominado CASE (*Computer Aided Software Engineering por sus siglas en inglés* o Ingeniería de Software Asistida por Computadora). Estas incluyen un conjunto de programas que facilitan la optimización de un producto ofreciendo apoyo permanente a los analistas, ingenieros de software y desarrolladores. CASE es la aplicación de métodos y técnicas que dan utilidades a los programas, por medio de otros, procedimientos y su respectiva documentación (Herramientas CASE para ingieneria de requisitos, 2008).

Como herramienta CASE se utiliza *Visual Paradigm*: por ser una herramienta que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Además, de que utiliza UML (por sus siglas en inglés, *Unified Modeling Language*) como lenguaje unificado de modelado, es colaborativa, soporta múltiples usuarios trabajando sobre el mismo proyecto. Permite control de versiones y realizar ingeniería tanto directa como inversa en diferentes lenguajes. Se pueden representar todos los tipos de diagramas UML para las distintas fases del ciclo de vida de un proyecto como el diagrama de despliegue en este caso.

1.6 Herramientas y tecnologías utilizadas

La virtualización basada en contenedores se ha vuelto una práctica muy habitual, especialmente en los servicios en la nube para poder sacar mayor partido de los servidores de los centros de datos. Por lo que se han desarrollado herramientas para implementar estas soluciones de forma más eficiente como: *PodMan*, Singularity, *Docker* entre otras para la creación y administración contenedores. Soluciones que han sido complementadas con otras herramientas como *Kubernetes*, *Docker Swarm*, *OpenShift* entre otras, para la orquestación y balancear la carga de estos contenedores.

PodMan

Es una herramienta de código abierto para desarrollar, gestionar y ejecutar contenedores en sistemas Linux. Fue desarrollada por ingenieros de *Red Hat* y la comunidad *open source* para gestionar el ecosistema completo de contenedores usando la biblioteca *libpod*. Las funciones y herramientas que la acompañan, como *Buildah* y *Skopeo*, permiten a los desarrolladores personalizar los entornos de contenedores según sus necesidades. Los *pods* son grupos de contenedores que se ejecutan juntos y comparten los mismos recursos, de manera similar a los *pods* de *Kubernetes*. *Podman* los gestiona a través de una interfaz de línea de comandos (CLI) sencilla y de la biblioteca *libpod*, la cual ofrece las API para administrar los contenedores, sus imágenes, los *pods* y los volúmenes (Mario Germán Castillo Ramírez, 2021).

Singularity

Es una plataforma de contenedores que permite crear y ejecutar contenedores de software de manera portátil y reproducible. Fue desarrollada por *Sylabs* y se enfoca en la verificabilidad y la reproducibilidad de los contenedores. Los contenedores de *Singularity* se pueden usar para empaquetar flujos de trabajo científicos completos, software y bibliotecas, e incluso datos. Esto significa que no se tiene que pedirle al administrador del clúster que instale; puede ponerlo en un contenedor de *Singularity* y ejecutarlo (Gómez, 2019)**.**

Docker

Esta es una tecnología de virtualización que permite a los desarrolladores empaquetar, enviar y ejecutar aplicaciones en contenedores. Estos son una forma ligera y portátil de virtualización que permiten a las aplicaciones y servicios ejecutarse de manera aislada en diferentes entornos, sin necesidad de preocuparse por las diferencias de infraestructura o de sistema operativo. Además, *Docker* se integra con herramientas para la gestión y orquestación de contenedores, lo que permite escalar y gestionar aplicaciones de manera fácil y eficiente (). A continuación se presenta una tabla comparativa de los gestores de contenedores.

Tabla 3: Comparación entre gestor de contenedores

Parámetro	***Docker***	***PodMan***	***Singularity***
Facilidad de uso	Facilidad de uso y amplio ecosistema	No requiere un *daemon* central y tiene una interfaz de línea de comandos sencilla	Facilidad de uso
Flexibilidad	Muy flexible y ejecuta cualquier aplicación en su propio contenedor	Muy flexible y puede ejecutar cualquier aplicación en su propio contenedor	Es muy flexible
Gestión de imágenes	Gestión de imágenes muy sólida, lo que facilita la creación, gestión y eliminación de contenedores	Gestión de imágenes muy sólida, lo que facilita la creación, gestión y eliminación de contenedores	Gestión de imágenes sólida, lo que facilita la creación, gestión y eliminación de contenedores
Integración con otras herramientas	Amplia gama de integraciones con herramientas de *DevOps*.	Se integra bien con *Kubernetes* y otras herramientas de *DevOps*	Se integra bien con algunas herramienta de *DevOps*

Luego de la comparación entre herramientas de orquestación de contenedores, se selecciona *Docker* como tecnología de contenerización debido a que en la investigación de los sistemas homólogos realizado en el epígrafe 1.4 se determina que es el utilizado por estos sistemas.

1.6.1 *Docker*

En la Figura 3 se presenta la arquitectura cliente servidor de *Docker*, donde se puede apreciar la capa cliente, la cual puede ejecutar instrucciones directamente sobre el anfitrión de *Docker* para descargar imágenes y desplegar contenedores. En el *Docker Host* existe un *Docker Daemon*; que se encarga de controlar las imágenes y los contenedores que un cliente puede generar. Toda la comunicación entre cliente y servidor se realiza a través de *sockets,* utilizando un API de tipo *Restfull.*

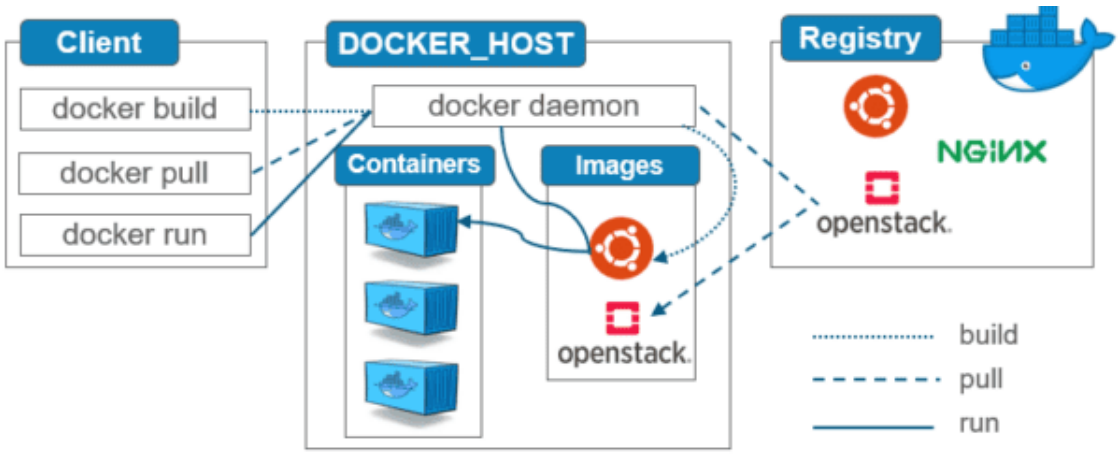


Figura 3: Arquitectura de Docker (Fuente: tomada de Geekflare. Arquitectura Docker y sus componentes) (Avi, 2023).

El *Docker Daemon* gestiona las imágenes a través del *Docker registry*, el cual se encarga de organizar y publicar las imágenes que un usuario puede utilizar. Cada contenedor se crea a partir de una imagen, es un entorno seguro y aislado en el que se ejecuta la aplicación. *Docker* administra los recursos del sistema operativo, incluyendo espacios de nombres, dispositivos de red y cortafuegos, a través de *libcontainer*. La administración de los contenedores se realiza siguiendo el estándar *Open Container Initiative-OCI* (Luz Elena Gutiérrez López, 2020).

Componentes en el Motor *Docker*:

Servidor: Es el *daemon Docker* llamado *dockerd.* Puede crear y gestionar imágenes *Docker*, contenedores, redes, entre otras.

API *Rest*: Se utiliza para indicar al *daemon Docker* lo que debe hacer.

Interfaz de línea de comandos (CLI): Este cliente es utilizado para introducir comandos *Docker*.

Cliente *Docker*

Los usuarios pueden interactuar con *Docker* a través de un cliente. Cuando se ejecuta cualquier comando *Docker*, el cliente lo envía al *daemon Docker*, que lo lleva a cabo. La API de *Docker* es utilizada por los comandos *Docker.* El cliente *Docker* puede comunicarse con más de un *daemon* (Avi, 2023).

Registros *Docker*

Es la ubicación donde se guardan las imágenes *Docker.* Puede ser un registro *Docker* público o privado. *Docker Hub* es el lugar por defecto de las imágenes *Docker,* el registro

público de sus almacenes. También puede crear y ejecutar su propio registro privado (Avi, 2023).

Imágenes

Una imagen en *Docker* es un archivo o file que se encuentra compuesto de diversas capas y que se utiliza con el objetivo de ejecutar un código dentro de un contenedor de *Docker*. Estas imágenes contienen todo el sistema de ficheros inicial en los que se va a basar el *container* para su funcionamiento, así como su punto de entrada o *entrypoint.* Este punto de entrada se refiere a la aplicación o comando que deberá ejecutarse una vez que el usuario lance un contenedor que esté asociado a esa imagen en *Docker* (Avi, 2023).

Contenedores

Los contenedores son paquetes que se basan en el aislamiento virtual para implementar y ejecutar aplicaciones que acceden a un *kernel* (núcleo) de sistema operativo compartido sin la necesidad de máquinas virtuales. Las tecnologías basadas en contenedores nacen del concepto de particionamiento o segmentación del hardware y del aislamiento del proceso *chroot*. Aunque el término contenedor es relativamente nuevo los sistemas operativos UNIX usaban “jaulas” como forma de aislar un proceso y sus subprocesos del resto del sistema para evitar acceder a recursos protegidos. El objetivo final de los contenedores ha crecido respecto a las jaulas, queriendo no prohibir el acceso a determinados recursos, sino aislar un proceso de todos los recursos excepto a los que se le hayan permitido explícitamente para su ejecución.

El uso de contenedores es una buena práctica para el desarrollo de software, pero la construcción manual de los mismos puede ser una tarea complicada que da lugar a errores. Con la intención de solucionar este problema surgió *Docker* es una plataforma de código abierto que permite crear, usar y ejecutar aplicaciones dentro de contenedores de software; así cualquier programa que se ejecute con *Docker* se está ejecutando en un contenedor consistente construido según los mejores métodos de desarrollo.

1.6.2 *Docker Compose*

Es una herramienta que permite definir y ejecutar aplicaciones de *Docker* que necesitan de más de un contenedor para funcionar correctamente. Con un archivo de configuración llamado *docker-compose.yml*, se puede configurar todos los servicios que la aplicación necesita. Luego, con un único comando, se pueden crear e iniciar todos los servicios que se han definido anteriormente en el archivo. Algunas de las características que *Docker Compose* aporta son la persistencia de datos en volúmenes cuando los contenedores son creados, lo que significa que todos aquellos datos que se ha guardado dentro de los

volúmenes durante la ejecución del contenedor no se perderán. Otra característica es que se almacena en caché la configuración de los contenedores, lo que significa que si se modifica la configuración, cuando se reinicia el servicio se reusarán los contenedores cuya configuración no ha cambiado (Juan, 2021) (2023).

1.6.3 *Docker Swarm*

Swarm es un software creado por los programadores de *Docker* que permite agrupar una serie de hosts de *Docker* en un clúster y gestionar los clústeres de forma centralizada, así como orquestar los contenedores. *Docker Swarm* se basa en una arquitectura maestro-esclavo. Cada *clúster* de *Docker* está formado al menos por un nodo maestro (también llamado administrador o *manager* en inglés) y tantos nodos esclavos (llamados de trabajo o *workers* en inglés) como sea necesario. Mientras que el maestro de *Swarm* es responsable de la gestión del clúster y la delegación de tareas, el esclavo se encarga de ejecutar las unidades de trabajo (tareas o *tasks* en inglés) que son asignadas por los nodos manager.

Docker Swarm soporta dos modos de definir servicios *swarm*: servicios globales o replicados.

- **Servicios replicados**: se trata de tareas que se ejecutan en un número de réplicas definido por el usuario. A su vez, cada réplica es una instancia del contenedor definido en el servicio. Los servicios replicados se pueden escalar, permitiendo a los usuarios crear réplicas adicionales. Si así se requiere, un servidor web se puede escalar en 2, 4 o 100 instancias con una sola línea de comandos.
- **Servicios globales**: si un servicio se ejecuta en modo global, cada nodo disponible en el clúster inicia una tarea para el servicio correspondiente. Si al *clúster* se le añade un nodo nuevo, el *manager* de *swarm* le atribuye una tarea para el servicio global de forma inmediata.

Otra de las características de *Docker Swarm* es el balanceo de cargas, pues con el modo enjambre *Docker* dispone de funciones integradas de balanceo de carga. *Docker* distribuye las consultas entrantes de forma inteligente entre las instancias del servidor web disponibles (2023).

Balanceo de carga: Los nodos manager en un *clúster* de *Docker Swarm* utilizan balanceo de carga para exponer los servicios que se desean hacer disponibles al exterior. Para ello, se asignan puertos *PublishedPorts* que son accesibles desde el exterior. Cualquier puerto que no esté en uso puede ser seleccionado. Los componentes externos, como los balanceadores de carga en la nube, pueden acceder a los servicios a través de los

PublishedPorts de cualquier nodo en el *clúster,* incluso si el nodo al que se accede está ejecutando actualmente la tarea para ese servicio. Además, el modo *Swarm* tiene un DNS interno que asigna automáticamente una entrada DNS a cada servicio. Los managers usan balanceo de carga interno para distribuir las peticiones entre los servicios dentro del *clúster* basados en los nombres de servicio atribuidos por el DNS.

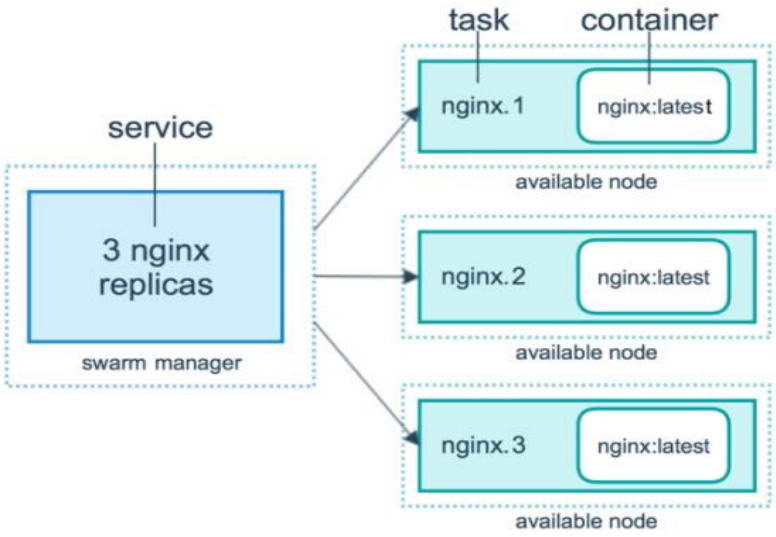


Figura 4: Arquitectura Docker Swarm (Fuente: Elaboración propia)

La arquitectura de *Docker Swarm* que se presenta en la Figura 4, tiene un nodo manager que contiene el servicio *nginx* al cual se le realizan 3 réplicas en 3 nodos disponibles, donde cada uno posee un *task* con un contenedor.

1.5.4 Editor de texto

Visual Studio *Code* (VS *Code*) es un editor de código fuente ligero, pero de gran potencia que se ejecuta en las computadoras con sistemas operativos Windows, MacOS o Linux. Viene con soporte integrado para *JavaScript, TypeScript y Node.js,* y con un buen ecosistema de extensiones para otros lenguajes (2023). Además de tener extensiones para el trabajo con *Docker*. Por esto es seleccionado para la configuración de los archivos.

Conclusiones parciales

La fundamentación teórica en el presente capítulo, permitió el análisis sobre los sistemas homólogos que aportó detalles significativos en cuanto a funcionalidades y características que requiere la configuración de la arquitectura para el comportamiento multi-instancia del sistema XAVIA PACSServer 4.0, así como el uso de conceptos como computación en la nube y arquitectura multi-instancia mediante los cuales se pone en contexto las bases sobre

las que se desarrollará el trabajo. Además, la descripción de los sistemas XAVIA PACS-RIS y XAVIA PACSServer aporta un conocimiento general del sistema con el que se va a trabajar. Luego del análisis realizado para determinar las herramientas y tecnologías necesarias que se ajustan al tipo de proyecto que se está realizando, se determinó usar *Docker* para crear y administrar contenedores. Se seleccionó *Docker Swarm* para la orquestación y balanceo de carga, *Docker Compose para* definir y ejecutar aplicación de *Docker.* Como entorno de desarrollo *Visual Estudio Code* y *Visual Paradigm* para el modelado de diagramas UML.

Capítulo 2: Propuesta de solución

En el presente capítulo se describe la propuesta de solución a la problemática planteada. Se plantea el proceso de automatización del comportamiento multi-instancia para el despliegue del sistema. Se describe y configura el proceso mediante los parámetros de configuración haciendo uso de la herramientas seleccionadas y se propone una nueva forma de despliegue basada en el uso de contenedores *Docker.*

2.1 Descripción de la propuesta de solución

La solución debe permitir administrar el sistema XAVIA PACSServer 4.0 desde un servidor central y que se ejecuten varias instancias del sistema. Este servidor prestará el servicio a diferentes instituciones de salud. Para lograr mayor seguridad de la información, cada cliente contará con su propia base de datos. Cuando se requiera configurar este servicio para un nuevo centro de salud se hará con una dirección IP mediante la cual el cliente podrá hacer uso del mismo. También se podrá eliminar la configuración del servicio en caso que la institución decida prescindir de él.

Cuando un servicio sea eliminado quedará la información cifrada debido a que se usarán los volúmenes de *Docker* para guardar la información y estos aunque se elimine el contenedor, la información persistirá Cosa que no ocurre cuando se hace uso de *bind mounts* que es otra forma de guardar la información con la que cuenta *Docker,* donde una vez eliminado el contenedor se elimina todo tipo de información.

Docker Swarm, se comporta como un enjambre que permitirá agregar al clúster más servidores a medida que aumenta la demanda del servicio. Este posibilitará que se puedan ejecutar varias réplicas del mismo sistema para que siempre exista al menos una que pueda dar respuesta al servicio que se está solicitando.

Haciendo uso de esta solución el centro aumenta la flexibilidad y escalabilidad en cuanto a poder computacional. Se disminuyen los costos, se optimiza el uso de recursos de los servidores y los tiempos. Además, permitirá que otros centros de salud que no hacen uso de este sistema, debido a que no cuentan con la infraestructura o los recursos necesarios para tener este tipo de tecnologías, puedan acceder a ellos.

Se aborda el proceso de configuración para el comportamiento multi-instancia en un sistema PACS web basado en la nube, desde la creación de las instancias hasta su despliegue y gestión.

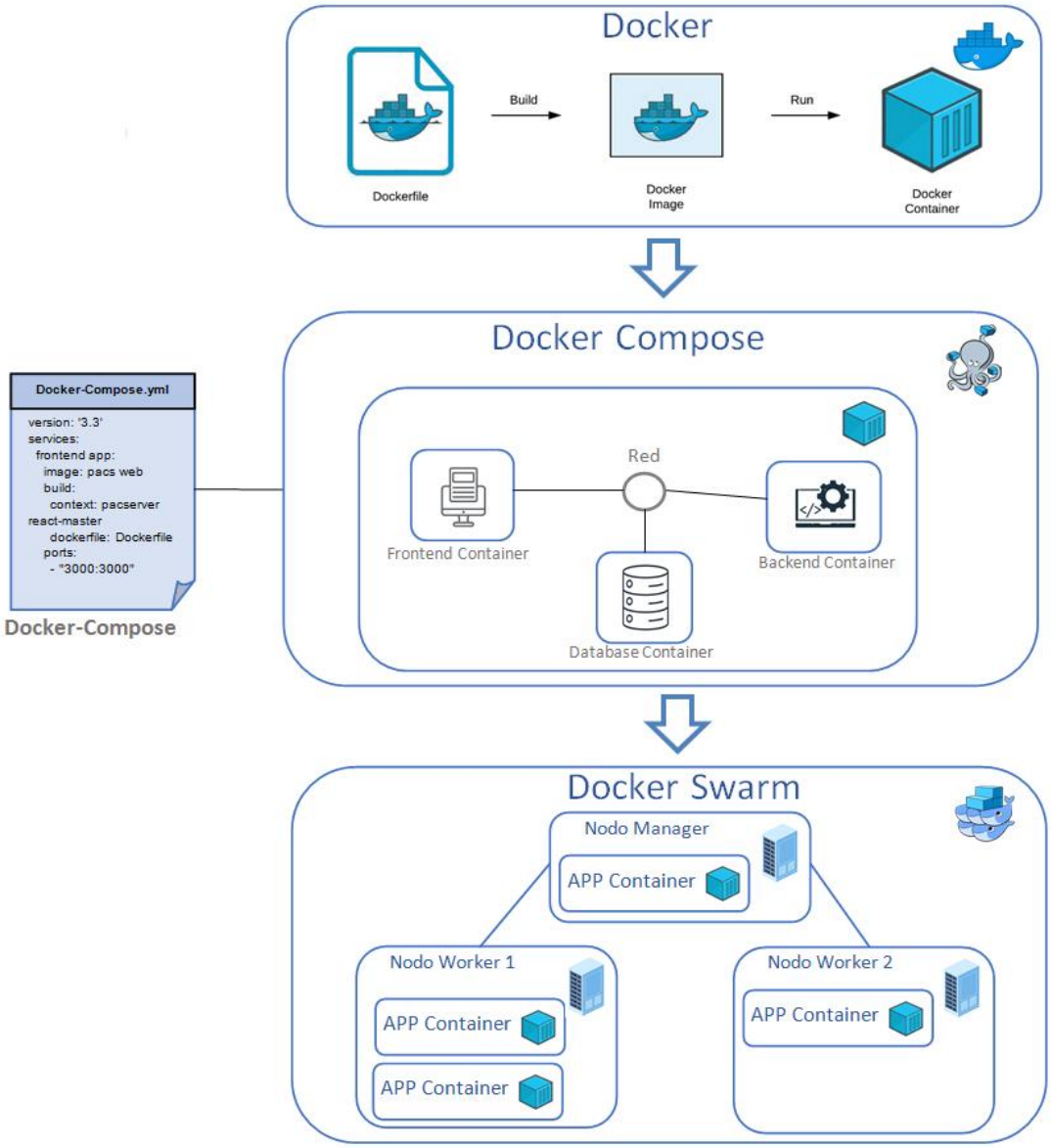


Figura 5: Estrategia de configuración para el despliegue del sistema XAVIA PACSServer 4.0 (Fuente Elaboración propia).

En la Figura 5 se puede apreciar el diseño de la estrategia de configuración para el despliegue del sistema XAVIA PACSServer 4.0. Esta arquitectura se caracteriza por varios elementos clave que se describen a continuación:

Imagen de *Docker*

Primero, se crea una imagen de *Docker*, proceso que se realiza a partir de un archivo *Dockerfile*, que contiene una serie de instrucciones que *Docker* sigue para construir la imagen. La imagen de *Docker* es esencialmente un modelo que *Docker* puede usar para crear contenedores.

Contenedor *Docker*

En segundo lugar, se crea el contenedor *Docker*. Este contenedor alberga la aplicación y se crea a partir de la imagen de *Docker* mencionada anteriormente. El proceso de creación del contenedor se realiza mediante un archivo *docker-compose*. Este archivo contiene todas las instrucciones necesarias para construir el contenedor, incluyendo cómo configurar el entorno del contenedor y qué servicios debe ejecutar.

Finalmente, el contenedor se despliega en varios nodos utilizando la tecnología *Docker Swarm*. *Esta* es una herramienta que permite a los desarrolladores gestionar un clúster de nodos *Docker* y desplegar servicios en ellos de manera eficiente.

Características del funcionamiento

- El sistema deberá estar disponible las 24 horas del día.
- La aplicación debe manejar un alto volumen de datos.
- Debe existir un nivel de aislamiento adecuado entre cada instancia para evitar el acceso no autorizado a la información.
- La información manejada por el sistema tendrá una cuidadosa protección para que no pueda ser modificada.

Ambiente de ejecución

- El ambiente en el que se ejecutará debe ser en un sistema operativo Linux.
- Puede ser ejecutado en un servidor con un procesador de 64 bits y 8 GB de memoria RAM. Pero se recomienda una RAM de 32GB.

2.2 Estrategia para la configuración del despliegue del sistema

Basándose en la investigación realizada. Se propone una Estrategia para la configuración del despliegue de aplicaciones web basadas en la nube, para que posean un comportamiento multi-instancia, usando herramientas de contenerización. Esta estrategia no será usada por si sola, sino que se integra a otras estrategias de despliegue que tienen en cuenta todas las faces globales de un despliegue. Se propone que dentro de la última fase de despliegue que propone IBM o Métrica III, uno de sus pasos consistirá en una estrategia en sí, la cual cuenta con 3 fases o etapas divididas en 8 pasos, que una vez terminados permitirán que un sistema web posea un comportamiento multi-instancia. Estas fases son:

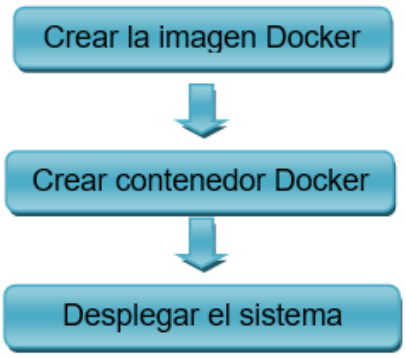


Figura 6: Etapas para la automatización del despliegue (Fuente Elaboración propia)

Fase 1: Crear la imagen *Docker*: El principal objetivo de esta fase es configurar las bases sobre el cual ejecutar el sistema. Los pasos a seguir en esta fase son.

- Crear *Dockerfile* para *frontend.*
- Crear imagen para *frontend* partiendo del *Dockerfile.*
- Crear *Dockerfile* para *backend*.
- Crear imagen para *backend* partiendo del *Dockerfile.*

Fase 2: Crear contenedor *Docker*: En esta fase se crea el contenedor que contendrá la aplicación. Los pasos son:

- Configurar el archivo *docker-compose.yml* a partir de las imágenes *Docker.*
- Crear contenedor.

Fase 3: Desplegar el sistema: En esta fase se realizan los pasos finales para desplegar el sistema. Cuenta con 2 paso que son:

- Activa el modo *Swarm* de *Docker.*
- Despliegue de la aplicación.

De forma general el procedimiento está estructurado de manera secuencial siguiendo una serie de fases. Las fases dependen unas de otras y no se puede pasar a la siguiente fase si no se ha cumplido con todos los subprocesos de la fase o etapa anterior.

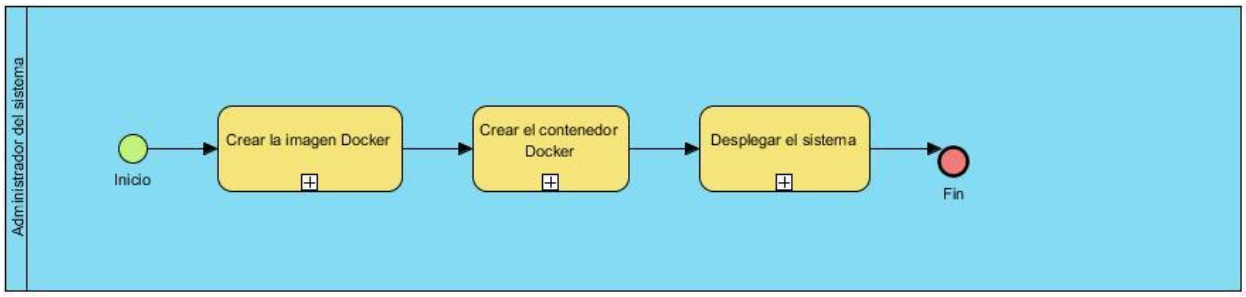


Figura 7: Estrategia para la configuración del despliegue del sistema XAVIA PACSServer 4.0 (Fuente Elaboración propia).

Fase 1: En esta primera fase es donde se configuran los archivos *Dockerfile* que interpreta *Docker* para la creación de las imágenes. En un primer paso se crea el archivo para el *frontend*, el cual tiene como entrada la imagen *node: 18-alpine*. En el segundo paso se ejecuta este *Dockerfile* y tiene como salida la imagen *frontend_app*. En el paso 3 se crea el archivo para el *backend* que tiene como entrada la imagen *aspnet*: 4.1 y se finaliza la primera fase con la ejecución de este archivo la cual tiene como salida la imagen *backend_app*. Todo este proceso queda representado en la Figura 7. Con estas dos imágenes se puede proceder a la ejecución de la fase 2.

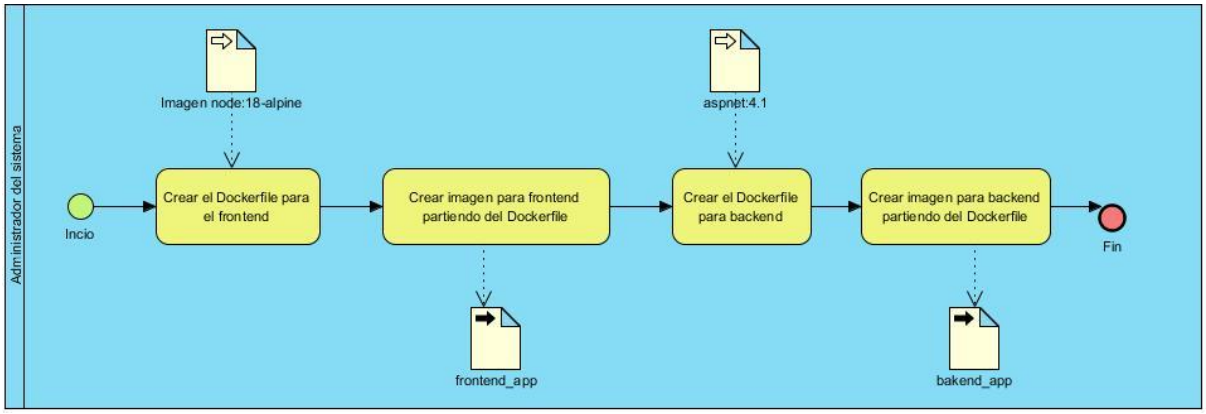


Figura 8: Creación de imágenes Docker (Fuente Elaboración propia).

Fase 2: En la segunda fase es donde se crea el contenedor a partir de las imágenes creadas y haciendo uso del *Docker Compose*. Para ello en un primer paso se crea el archivo *docker-compose.yml* este tiene como entrada las imágenes creadas en la fase 1 y una tercera imagen para crear el contenedor con las configuraciones de la base de datos llamada *postgres*: 9.4.5. Luego en el paso 2 se ejecuta este archivo y se obtiene como salida el contenedor del sistema XAVIA PACSServer 4.0. Este proceso puede observarse en la Figura 8. Luego se puede proceder con la fase final.

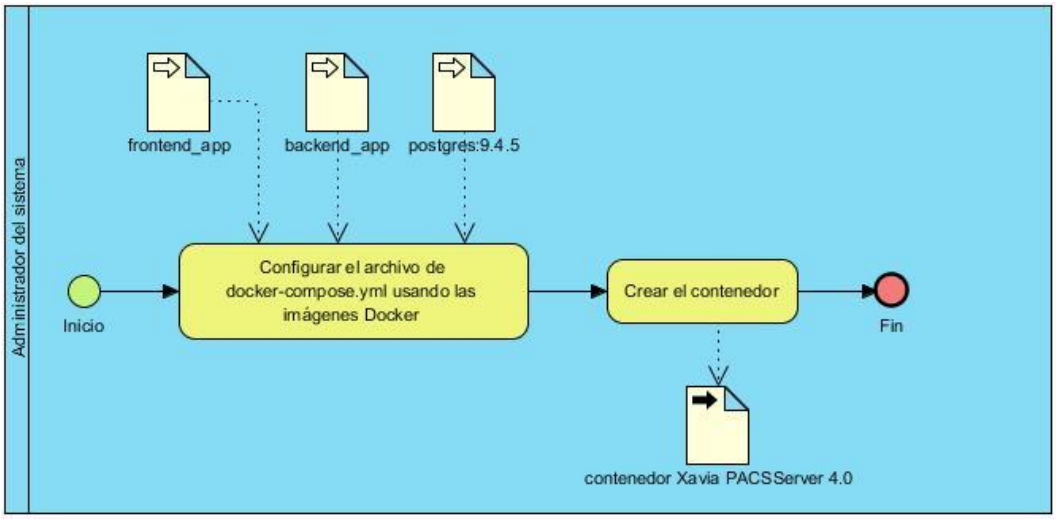


Figura 9: Creación del contenedor (Fuente Elaboración propia).

Fase 3: En esta última fase se comienza activando el modo *Swarm* de *Docker* para poder ejecutar los contenedores en una granja de nodos, esto permite tener al menos un nodo maestro que ejecute el contenedor y posibilita que en cualquier momento puedan ser agregados nodos trabajadores que ejecuten el servicio. Una vez activado el modo *Swarm* se procede al paso 2 que es desplegar el servicio, este tiene como entrada el contenedor creado en la fase 2 de la estrategia. Este proceso se observa en la Figura 9.

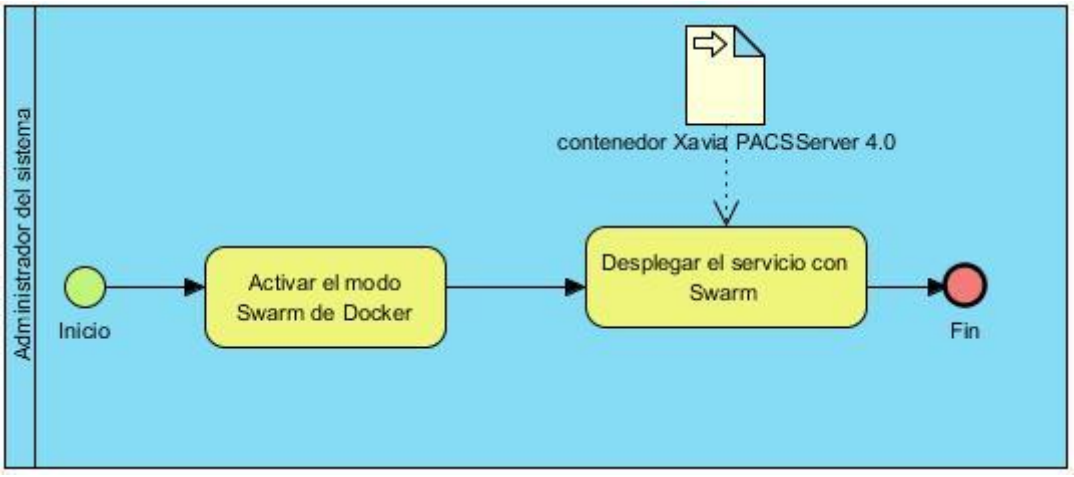


Figura 10: Despliegue del sistema (Fuente Elaboración propia).

Una vez finalizado este proceso quedaría desplegado el sistema XAVIA PACSServer con un comportamiento multi-instancia.

2.3 Configuración de los archivos para el despliegue

Para implementar la funcionalidad de multi-instancia, se sigue una serie de pasos para su configuración. Estos incluyen:

Etapa 1

Para configurar la multi-instancia mediante el uso de *Docker* primeramente: es necesario crear las imágenes que no son más que plantillas de solo lectura, es decir, una imagen puede contener el sistema de archivo de un sistema operativo, permitiendo crear los contenedores basados en esta configuración.

Para la creación de las imágenes se comienza con una imagen base que parte de un lenguaje de programación o de alguna distribución. Haciendo uso de los *Dockerfile* que son los archivos que contienen las instrucciones que crean las imágenes. Deben estar guardados dentro de un directorio. Este directorio es el que contiene todos los archivos necesarios para construir la imagen.

Dentro de este directorio se crea un archivo llamado *Dockerfile* con este contenido:
Comando de configuración para *Frontend*.

```
FROM node:18-alpine
    WORKDIR /frontend_app
    COPY . .
    EXPOSE 3000
    RUN npm config set registry=http://nexus.prod.uci.cu/repository/npm-all
    RUN npm install react-scripts@latest --force && npm install --force
    RUN npm run build
    CMD [ "npm","start" ]
```

Comando de configuración para *Backend*.

```
FROM aspnet:4.1
    WORKDIR /backend_app
    COPY --from=build-env /build/out .
    CMD ["dotnet", "DotNet.Docker.dll"]
```

Con el FROM se indica la imagen base sobre la que se basa esta imagen. El comando WORKDIR especifica un directorio de trabajo dentro del contexto de *build*. Con COPY se copia un archivo del *build context* y lo guarda en la imagen. El comando EXPOSE permite que un puerto sea vinculable a la máquina huésped a través del puerto especificado. RUN ejecuta el comando indicado durante el proceso de creación de la imagen. Y CMD es el comando por defecto que va a ejecutarse.

Para crear las imágenes se usa *Docker build*.

```
docker build -t frontend_app:v1 .
```

```
docker build -t backend_app:v1 .
```

El parámetro -t permite etiquetar la imagen con un nombre y una versión. El **.** indica que el *build context* es el directorio actual.

Etapa 2

Luego de creadas las imágenes es necesario construir los contenedores y conectarlos. Para esto se hace uso de *Docker Compose* que es una herramienta proporcionada por *Docker*, para automatizar la creación, inicio y parada de un contenedor o un conjunto de ellos. Se utiliza para definir y ejecutar aplicaciones multi-contenedor. Con un solo comando se puede crear e iniciar todos los servicios que necesita la aplicación. Para ello se crea un fichero llamado *docker-compose.yml* con el siguiente contenido.

Los pasos son:

1. Se crea el fichero de *Docker Compose.*

```
version: '3.3'
services:
    frontend_app:
        build:
            context: pacserver_react-master
            dockerfile: Dockerfile
        ports:
            - "3000:3000"

services:
    backend_app:
        build:
            context: Server XAVIA PACS
            dockerfile: Dockerfile

    persistence:
        image: postgres:9.4.5
        ports:
            - 5432:5432
        environment:
            -POSTGRES_USER= postgres
            -POSTGRES_PASSWORD=postgres
        volumes:
            - ./postgres.conf:/etc/postgresql/postgresql.conf

        comand: postgres -c config_file=/etc/postgresql/postgresql.conf
        restart: always
```

Con este fichero se crean todos los contenedores que forman el programa; *frontend_app*, *backend_app*, la base de datos *postgres* y crea una red a la cual conecta todos los contenedores.

Los ficheros de *Compose* están divididos en tres secciones: *services*, *volumes* y *networks*; e indican un número de versión. Esto permite realizar lo mismo que el cliente de *Docker*, pero de forma automática.

2. Se ejecuta el fichero.

```
docker-compose up
```

3. Se comprueba si se crearon correctamente los contenedores.

```
docker-compose ps
```

El cual le dará un resultado como este.

```
        Name                       Command                  State     Ports
-----------------------------------------------------------------------------
contenedor_frontend_app_1   docker-entrypoint.sh npm start   Exit 1
contenedor_backend_app_1    docker-entrypoint.sh npm start   Exit 1
contenedor_postgres         docker-entrypoint.sh npm start   Exit 1
```

Etapa 3

Una vez esté listo el archivo de *Docker Compose* se procede al despliegue con *Docker Swarm*.

Los pasos son:

1. Se activa el modo *Swarm*.

```
docker swarm init
```

2. Se despliega una pila de servicios.

```
docker service create --name service_savia --publish published=5000,target=5000 registry:1
```

A esta pila se le pone un nombre --name service_savia, se publica el puerto que usará --publish published=5000, target=5000 y el nombre de la imagen en la que se basa registry.

3. Se crea un *Docker stack.*

```
docker stack deploy --compose-file docker-compose.yml savia
```

A este stack se le especifica el *compose* que utilizará --compose-file docker-compose.yml y se le otorga un nombre stack_savia.

4. Se comprueba si se creó correctamente.

```
docker service savia
```

El cual arroja el siguiente resultado.

ID	NAME	MODE	REPLICAS	IMAGE	PORT
h29yypunt7v6	savia_frontend_app	replicated	1/1	pacs_web:latest	*:3000->3000/tcp
0ouy67329yyp	savia_backend_app	replicated	1/1	pacs_web:latest	
40pdhu8rns34	savia_postgres_app	replicated	1/1	pacs_web:latest	

De esta forma queda configurado el despliegue del sistema XAVIA PACSServer 4.0.

2.4 Tipo de arquitectura cliente/servidor

Es un tipo de desarrollo de software, en el que las diferentes labores se racionan entre los recursos, como son los servidores y los demandantes que son los clientes.

- Cliente: Programa ejecutable que participa activamente en el establecimiento de las conexiones. Envía una petición al servidor y se queda esperando por una respuesta. Su tiempo de vida es finito una vez que son servidas sus solicitudes, termina el trabajo.
- Servidor: Es un programa que ofrece un servicio que se puede obtener en una red. Acepta la petición desde la red, realiza el servicio y devuelve el resultado al solicitante. Al ser posible implantarlo como aplicaciones de programas, puede ejecutarse en cualquier sistema donde exista TCP/IP y junto con otros programas de aplicación. El servidor comienza su ejecución antes de comenzar la interacción con el cliente.

Ambos roles pueden actuar de manera independiente y realizar sus propias actividades. Permite la independencia geográfica, es decir el cliente y el servidor pueden estar en sitios distintos e incluso no es necesario el conocimiento de su ubicación para la comunicación. Una de las ventajas claves es la centralización del control. En este tipo de arquitectura, el servidor es responsable de gestionar los accesos, los recursos y garantizar la integridad de los datos. Y tanto los clientes como los servidores pueden aumentar su capacidad de forma independiente. Al distribuir las funciones y responsabilidades entre varios ordenadores independientes, se facilita la tarea de reemplazar, reparar, actualizar e incluso trasladar un servidor, sin afectar a los clientes de manera significativa o incluso sin afectarlos en absoluto (2023).

En cuanto a la seguridad, en las redes Cliente/Servidor, los clientes no tienen acceso directo a las direcciones IP de otros clientes, lo que dificulta el rastreo y el *hackeo* de los usuarios. Esto brinda una capa adicional de protección y privacidad en comparación con las redes P2P, donde la visibilidad de las direcciones IP es mayor.

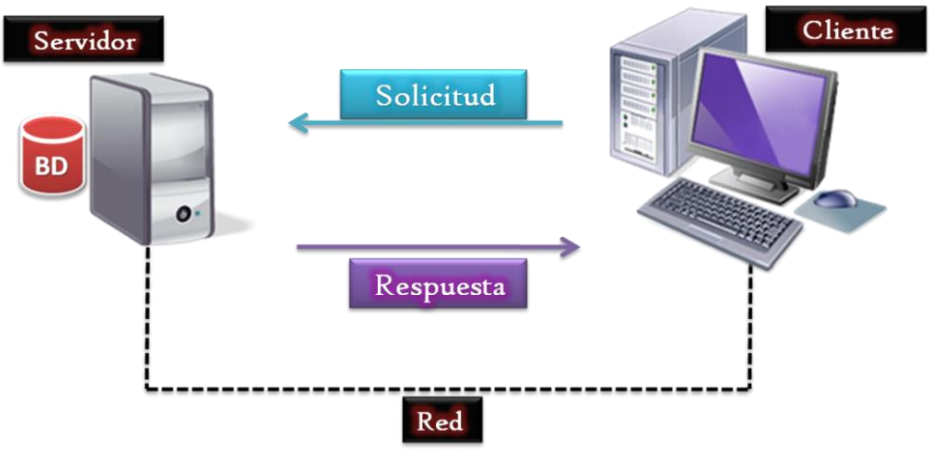


Figura 11: Arquitectura Cliente Servidor (Fuente: tomado de Historia de la web ()).

Una arquitectura cliente-servidor es una forma de organizar una red de computadoras en la que un servidor central proporciona servicios a los clientes que lo solicitan. Una arquitectura multi-instancia es una forma de implementar un servidor que puede manejar múltiples solicitudes simultáneas de diferentes clientes, cada una con su propia instancia del servicio. Algunas ventajas de usar una arquitectura cliente-servidor y multi-instancia para un servidor PACS web basado en la nube son: escalabilidad, interoperabilidad, mantenimiento, seguridad.

2.5 Diagrama despliegue

El modelo de despliegue se realiza como parte de la implementación para describir la distribución física del sistema. Establece la correspondencia entre la arquitectura lógica, los procesos y los nodos. Cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo de hardware similar. Los nodos poseen relaciones que representan medios de comunicación entre ellos. Se reflejan en este artefacto los protocolos de comunicación mediante los cuales se comunican los nodos respectivos.

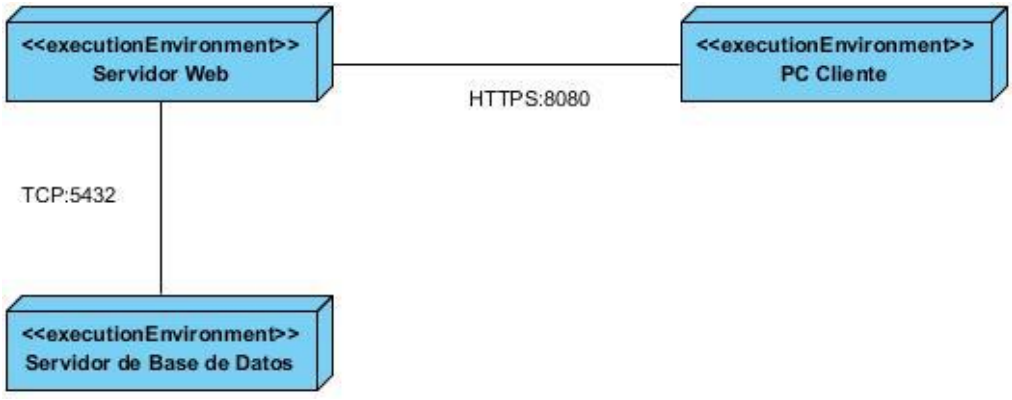


Figura 12: Diagrama de despliegue (Fuente: Elaboración propia).

Conclusiones parciales

En este capítulo se obtuvo una descripción detallada de la propuesta de solución en la cual se reflejaron las características del comportamiento multi-instancia aplicada al sistema XAVIA PACSServer 4.0. Se describe la propuesta de configuración para el despliegue del sistema XAVIA PACSServer 4.0, indicando las fases y actividades a desarrollar en cada una. Además, se reflejó la configuración de sistema y se modeló la arquitectura física del sistema.

Capítulo 3: Validación del sistema diagnóstico

En el presente capítulo se aborda sobre la seguridad que brindan las herramientas sobre las cuales se trabaja. Además, se evalúa el grado de calidad y fiabilidad de los resultados obtenidos luego de la configuración del comportamiento multi-instancia en el sistema XAVIA PACSServer 4.0.

3.1 Seguridad *Docker*

Los contenedores *Docker* son seguros por varias razones:

- **Aislamiento**: *Docker* utiliza varias características del *kernel* de Linux, como *cgroups* y espacios de nombres, para aislar los contenedores entre sí y del host. Esto significa que incluso si un contenedor se ve comprometido, el atacante no puede acceder fácilmente a otros contenedores o al host.
- **Imágenes inmutables**: Las imágenes *Docker* son inmutables, lo que significa que no cambian una vez creadas. Esto reduce la superficie de ataque, ya que un atacante no puede instalar software malicioso en una imagen.
- **Control de recursos**: *Docker* permite limitar los recursos que un contenedor puede utilizar, como la CPU, la memoria y el disco. Esto puede prevenir ataques de agotamiento de recursos.
- **Seguridad a nivel de plataforma**: *Docker* proporciona características de seguridad a nivel de plataforma, como *Docker Content Trust y Docker Bench for Security,* que pueden ayudar a detectar y prevenir amenazas.
- **Menor superficie de ataque**: Como los contenedores *Docker* suelen ser de un solo uso, tienen menos componentes y por lo tanto una menor superficie de ataque en comparación con las máquinas virtuales.
- **Actualizaciones rápidas**: Como las imágenes *Docker* son fáciles de construir y desplegar, es más fácil aplicar parches de seguridad y actualizaciones.

3.1.1 Seguridad *Swarm*

Seguridad ante fallas en un servidor:

Si uno de los servidores del enjambre de *Swarm* se apaga repentinamente, los contenedores que se ejecutan en ese servidor se detendrán. Sin embargo, *Docker Swarm* tiene un mecanismo de recuperación incorporado que garantiza que los contenedores se reinicien automáticamente en otro servidor del enjambre.

Docker Swarm también tiene un mecanismo de equilibrio de carga incorporado que garantiza que los contenedores se distribuyan uniformemente entre los servidores del enjambre. Si el servidor que falla es el líder del enjambre, *Docker Swarm* elegirá automáticamente un nuevo líder.

Comportamiento ante la falta de disponibilidad de servidores:

Si no hay ningún servidor disponible en el enjambre de *Swarm,* los servicios permanecerán en un estado pendiente hasta que haya un nodo disponible que pueda ejecutar sus tareas. Si no hay suficientes recursos disponibles para ejecutar una tarea, *Docker Swarm* no programará la tarea hasta que haya suficientes recursos disponibles.

3.1.2 Volumen

Si elimina un contenedor de *Docker* que tiene un volumen, este no se elimina automáticamente. Los volúmenes persisten hasta que no hay contenedores que los usen. Puede listar los volúmenes con el comando `docker volume ls`. Si desea eliminar un volumen, puede usar el comando `docker volume rm <volume_name>`. Si desea eliminar todos los volúmenes no utilizados, puede usar el comando `docker volume prune`.

3.2 Pruebas de software

Las pruebas demuestran que un programa hace lo que se necesita que haga, así como descubrir sus defectos antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa (Sommerville, 2011).

Las pruebas de software son una actividad fundamental en el desarrollo de cualquier producto o aplicación informática, que permiten asegurar su calidad, funcionalidad y rendimiento. Las pruebas de software consisten en someter al software a diferentes escenarios y condiciones para verificar que se comporta de acuerdo con los requisitos establecidos y que no presenta errores o defectos que afecten a su funcionamiento (Maidelyn Piñero González, 2021).

Pruebas de rendimiento

Las pruebas de rendimiento de software son un conjunto de pruebas que se realizan en el *testing* de software para medir la velocidad, fiabilidad y estabilidad de un sistema bajo una carga de trabajo determinada. Estas pruebas se realizan para localizar cuellos de botella, identificar y localizar problemas de rendimiento en la aplicación.

Existen varios tipos de pruebas de rendimiento, como son:

Prueba de carga: se establecen unos objetivos determinados y se mide el resultado.

Prueba de estrés: se aplica mucha carga, bastante más de la esperada, para ver cómo se comportaría la aplicación ante un pico de afluencia de usuarios.

Prueba de estabilidad: permite probar cómo se comporta la aplicación en una prueba de una duración larga con una carga moderada, para ver si el sistema se degrada o sigue funcionando correctamente.

Con el objetivo de hallar un recurso que permitiera implementar este tipo de pruebas, se llevó a cabo una indagación encontrando cuatro soluciones *QALoad*, *LoadRunner, Quality Center y JMeter*. Estos recursos reúnen las condiciones necesarias para realizar las pruebas de carga y estrés o apoyar el proceso de pruebas.

Se escogió como instrumento de prueba *JMeter*, la cual es una herramienta Java creada dentro del proyecto Jakarta, que permite efectuar pruebas de rendimiento y pruebas funcionales sobre aplicaciones web. Permite efectuar pruebas web tradicionales, pero también permite realizar test de FTP, JDBC, JNDI, LDAP, SOAP/XML-RPC, y *WebServices* (en Beta). Permite la ejecución de pruebas distribuidas entre distintos computadores, para realizar pruebas de rendimiento. Además, habilitar o inhabilitar una parte del test, lo que es muy útil cuando se está elaborando un test largo, y se desea desactivar ciertas partes iniciales que sean muy pesadas o largas. Tiene la forma de generar un caso de prueba a través de una navegación de usuario. Además de ser un software libre y gratis (SanchezAlmenares, 2019).

Tabla 4: Estrategia de pruebas. (Fuente: Elaboración propia).

Pruebas	**Método**	**Herramienta**	**Alcance**
Carga	Caja blanca	*Apache JMeter* v5.5	Se establecen objetivos determinados sobre la carga que se necesita que el sistema soporte. Para esto se simulan escenarios reales donde se pueden medir los límites y obtener información sobre las métricas definidas.
Estrés	Caja blanca	*Apache JMeter* v5.5	Estas pruebas se realizan para observar el comportamiento del sistema cuando es sometido a una carga superior a la establecida.

Para las pruebas de estrés y carga realizadas al modelo multi-instancia aplicado al sistema XAVIA PACSServer 4.0 se hizo uso de la herramienta *JMeter*. El objetivo de estas pruebas era verificar la capacidad del sistema para soportar diferentes niveles de demanda de los usuarios en el entorno y para ello, se realizaron tres iteraciones, incrementando el número de muestras en cada una.

El entorno seleccionado para las pruebas posee las siguientes características:

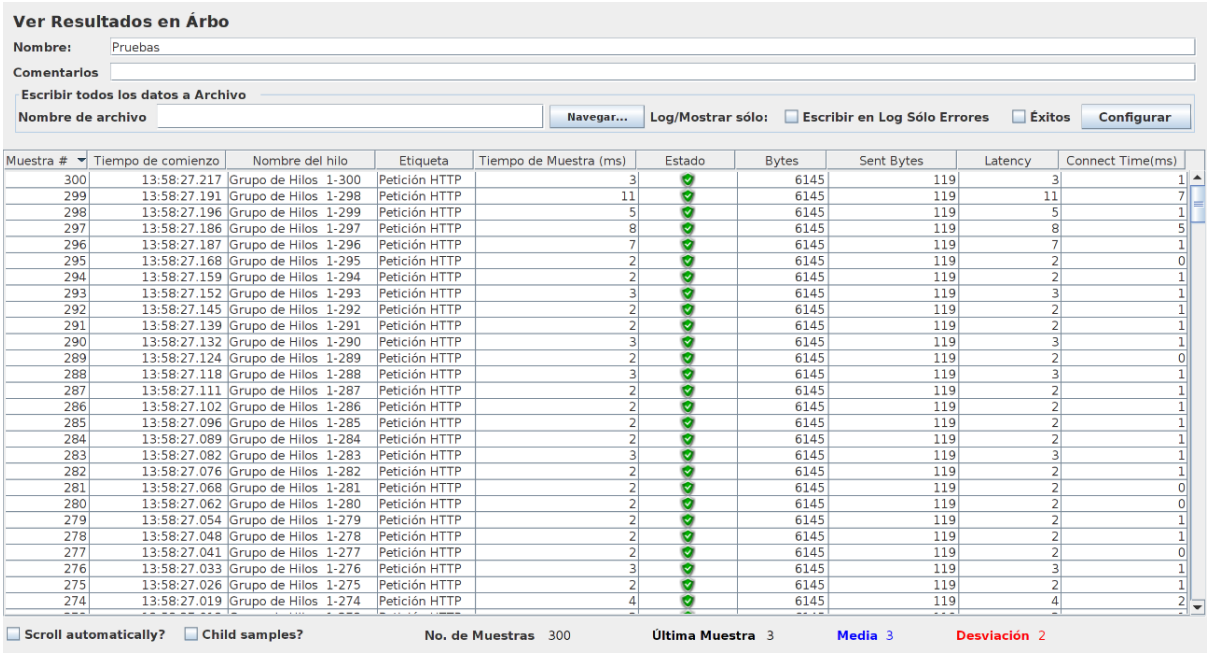
• **Microprocesador**: AMD A6 9225.

• **Memoria RAM**: 8 GB.

• **Tarjeta de red:** Ethernet: 10/100 Mbps.

• **Sistema operativo**: *Linux*, Arquitectura 64 bits.

Para una primera iteración se seleccionó una muestra 300 servicios para simular el comportamiento real del sistema.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestra (ms)	Estado	Bytes	Sent Bytes	Latency	Connect Time(ms)
300	13:58:27.217	Grupo de Hilos 1-300	Petición HTTP	3		6145	119	3	1
299	13:58:27.191	Grupo de Hilos 1-298	Petición HTTP	11		6145	119	11	7
298	13:58:27.196	Grupo de Hilos 1-299	Petición HTTP	5		6145	119	5	1
297	13:58:27.186	Grupo de Hilos 1-297	Petición HTTP	8		6145	119	8	5
296	13:58:27.187	Grupo de Hilos 1-296	Petición HTTP	7		6145	119	7	1
295	13:58:27.168	Grupo de Hilos 1-295	Petición HTTP	2		6145	119	2	0
294	13:58:27.159	Grupo de Hilos 1-294	Petición HTTP	2		6145	119	2	1
293	13:58:27.152	Grupo de Hilos 1-293	Petición HTTP	3		6145	119	3	1
292	13:58:27.145	Grupo de Hilos 1-292	Petición HTTP	2		6145	119	2	1
291	13:58:27.139	Grupo de Hilos 1-291	Petición HTTP	2		6145	119	2	1
290	13:58:27.132	Grupo de Hilos 1-290	Petición HTTP	3		6145	119	3	1
289	13:58:27.124	Grupo de Hilos 1-289	Petición HTTP	2		6145	119	2	0
288	13:58:27.118	Grupo de Hilos 1-288	Petición HTTP	3		6145	119	3	1
287	13:58:27.111	Grupo de Hilos 1-287	Petición HTTP	2		6145	119	2	1
286	13:58:27.102	Grupo de Hilos 1-286	Petición HTTP	2		6145	119	2	1
285	13:58:27.096	Grupo de Hilos 1-285	Petición HTTP	2		6145	119	2	1
284	13:58:27.089	Grupo de Hilos 1-284	Petición HTTP	2		6145	119	2	1
283	13:58:27.082	Grupo de Hilos 1-283	Petición HTTP	3		6145	119	3	1
282	13:58:27.076	Grupo de Hilos 1-282	Petición HTTP	2		6145	119	2	1
281	13:58:27.068	Grupo de Hilos 1-281	Petición HTTP	2		6145	119	2	0
280	13:58:27.062	Grupo de Hilos 1-280	Petición HTTP	2		6145	119	2	0
279	13:58:27.054	Grupo de Hilos 1-279	Petición HTTP	2		6145	119	2	1
278	13:58:27.048	Grupo de Hilos 1-278	Petición HTTP	2		6145	119	2	1
277	13:58:27.041	Grupo de Hilos 1-277	Petición HTTP	2		6145	119	2	0
276	13:58:27.033	Grupo de Hilos 1-276	Petición HTTP	3		6145	119	3	1
275	13:58:27.026	Grupo de Hilos 1-275	Petición HTTP	2		6145	119	2	1
274	13:58:27.019	Grupo de Hilos 1-274	Petición HTTP	4		6145	119	4	2

Figura 13: Resultados de las pruebas de carga primera iteración.

La respuesta del sistema ante la muestra de 300 servicios, arrojó resultados satisfactorios con una media de 3 ms y una desviación estándar 2 ms, dando bajos tiempos de respuesta por cada petición.

Lugo de obtener resultados satisfactorios en una primera iteración con una muestra de 300 servicios, se incrementa el número de servicios a 499 en una segunda iteración para identificar el comportamiento real del sistema ante mayor carga.

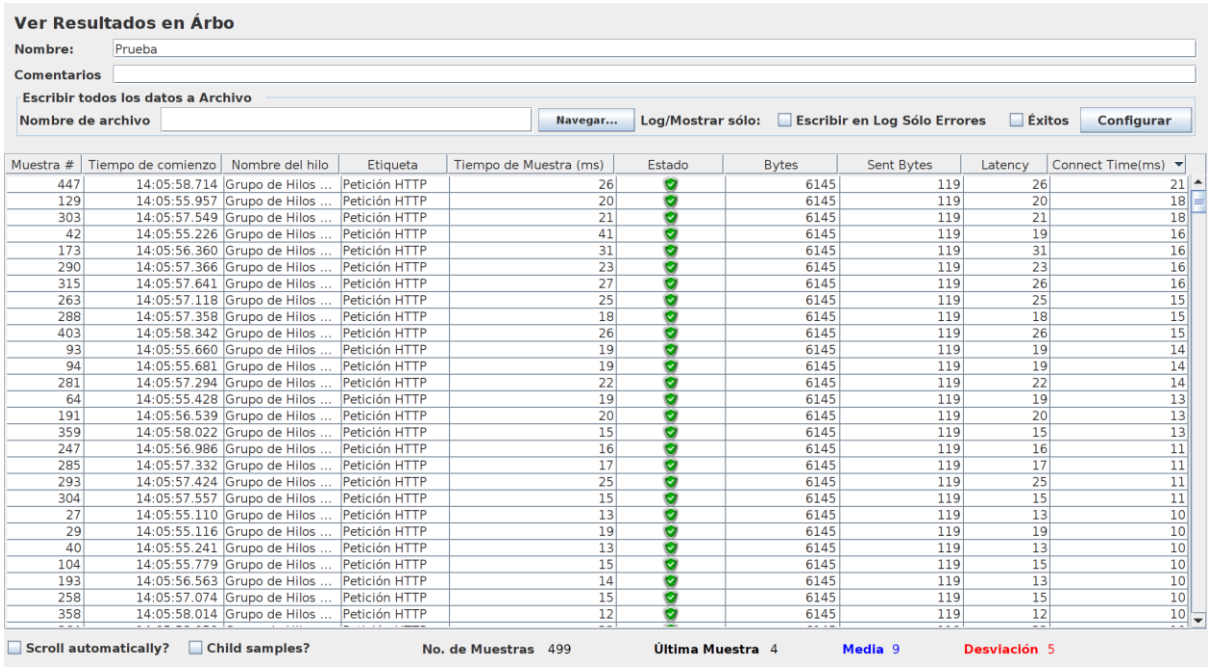
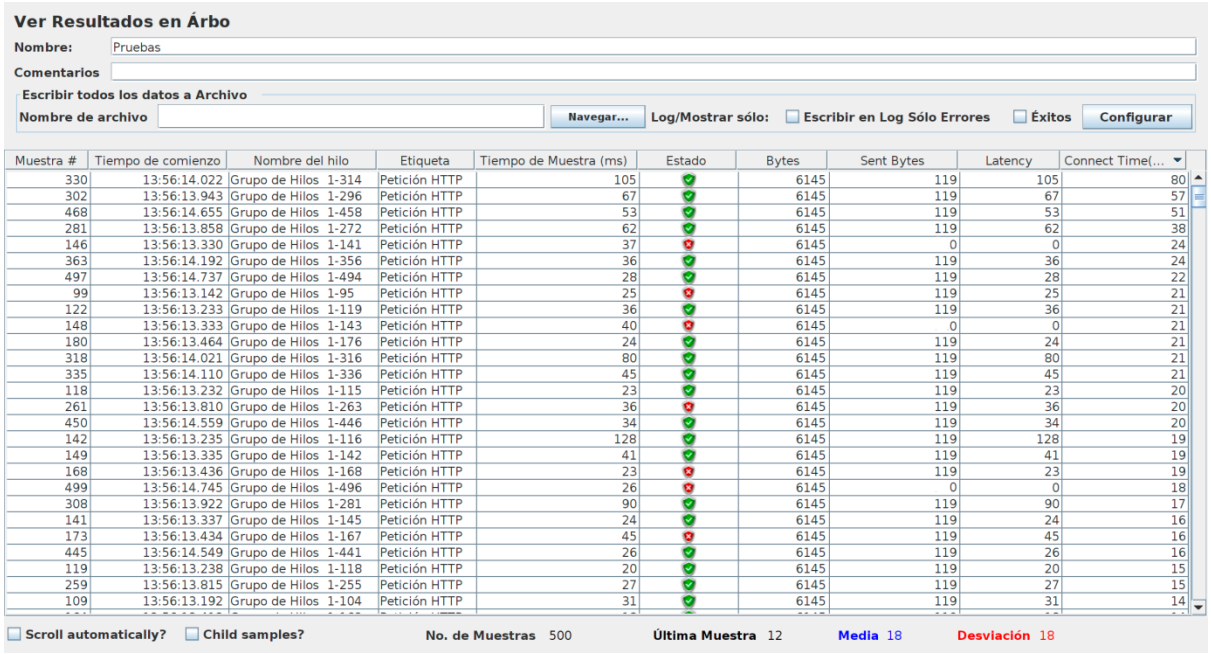


Figura 14: Resultados de las pruebas de carga segunda iteración.

La respuesta del sistema ante la muestra de 499 servicios, arrojó resultados satisfactorios con una media de 9 ms y una desviación estándar 5 ms. Dando bajos tiempos de respuesta por cada petición.

Una vez realizada dos iteraciones donde se comprobaron los servicios que es capaz de soportar el sistema en el ambiente seleccionado, se realiza una tercera iteración con una muestra superior a las anteriores de 500 servicios.



Muestra #	Tiempo de comienzo	Nombre del hilo	Etiqueta	Tiempo de Muestra (ms)	Estado	Bytes	Sent Bytes	Latency	Connect Time(...
330	13:56:14.022	Grupo de Hilos 1-314	Petición HTTP	105		6145	119	105	80
302	13:56:13.943	Grupo de Hilos 1-296	Petición HTTP	67		6145	119	67	57
468	13:56:14.655	Grupo de Hilos 1-458	Petición HTTP	53		6145	119	53	51
281	13:56:13.858	Grupo de Hilos 1-272	Petición HTTP	62		6145	119	62	38
146	13:56:13.330	Grupo de Hilos 1-141	Petición HTTP	37		6145	0	0	24
363	13:56:14.192	Grupo de Hilos 1-356	Petición HTTP	36		6145	119	36	24
497	13:56:14.737	Grupo de Hilos 1-494	Petición HTTP	28		6145	119	28	22
99	13:56:13.142	Grupo de Hilos 1-95	Petición HTTP	25		6145	119	25	21
122	13:56:13.233	Grupo de Hilos 1-119	Petición HTTP	36		6145	119	36	21
148	13:56:13.333	Grupo de Hilos 1-143	Petición HTTP	40		6145	0	0	21
180	13:56:13.464	Grupo de Hilos 1-176	Petición HTTP	24		6145	119	24	21
318	13:56:14.021	Grupo de Hilos 1-316	Petición HTTP	80		6145	119	80	21
335	13:56:14.110	Grupo de Hilos 1-336	Petición HTTP	45		6145	119	45	21
118	13:56:13.232	Grupo de Hilos 1-115	Petición HTTP	23		6145	119	23	20
261	13:56:13.810	Grupo de Hilos 1-263	Petición HTTP	36		6145	119	36	20
450	13:56:14.559	Grupo de Hilos 1-446	Petición HTTP	34		6145	119	34	20
142	13:56:13.235	Grupo de Hilos 1-116	Petición HTTP	128		6145	119	128	19
149	13:56:13.335	Grupo de Hilos 1-142	Petición HTTP	41		6145	119	41	19
168	13:56:13.436	Grupo de Hilos 1-168	Petición HTTP	23		6145	119	23	19
499	13:56:14.745	Grupo de Hilos 1-496	Petición HTTP	26		6145	0	0	18
308	13:56:13.922	Grupo de Hilos 1-281	Petición HTTP	90		6145	119	90	17
141	13:56:13.337	Grupo de Hilos 1-145	Petición HTTP	24		6145	119	24	16
173	13:56:13.434	Grupo de Hilos 1-167	Petición HTTP	45		6145	119	45	16
445	13:56:14.549	Grupo de Hilos 1-441	Petición HTTP	26		6145	119	26	16
119	13:56:13.238	Grupo de Hilos 1-118	Petición HTTP	20		6145	119	20	15
259	13:56:13.815	Grupo de Hilos 1-255	Petición HTTP	27		6145	119	27	15
109	13:56:13.192	Grupo de Hilos 1-104	Petición HTTP	31		6145	119	31	14

Figura 15: Resultados de las pruebas de estrés tercera iteración.

La respuesta del sistema ante la muestra de 500 servicios, arrojó resultados no satisfactorios al no responder algunas de las peticiones. Con una media de 18 ms y una desviación estándar 18 ms.

En el siguiente gráfico de barras se puede apreciar los resultados arrojados en las tres iteraciones anteriores donde se evaluaron el rendimiento a través de las pruebas de carga y estrés para corroborar el comportamiento del modelo multi-instancia aplicado al XAVIA PACSServer 4.0 durante la ejecución de varios servicios de manera simultánea generados por la herramienta *JMeter.*

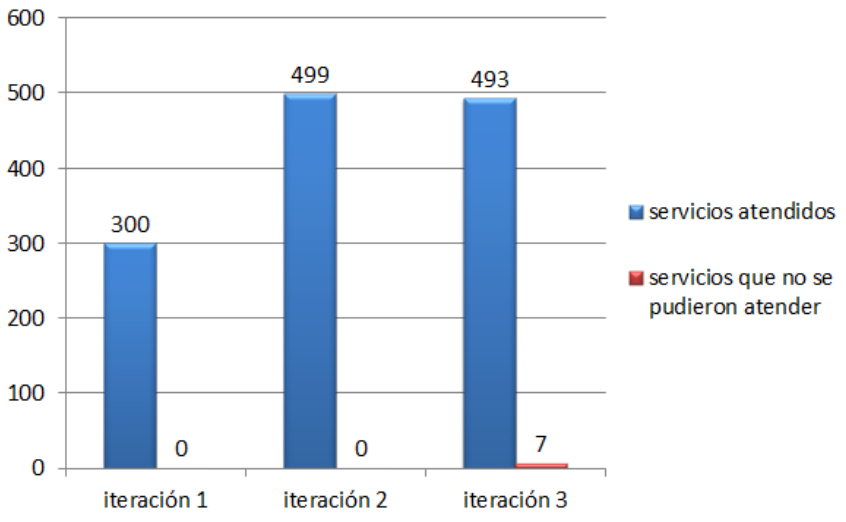


Figura 16: Resultado de las pruebas de estrés y carga (Fuente: Elaboración propia).

Como se puede observar, las respuestas fueron satisfactorias en las dos primeras iteraciones, donde fueron atendidos a todos los servicios solicitados. Sin embargo, en la tercera iteración el número de muestra superó los servicios que podían ser ejecutados debido a las limitaciones del entorno provocando que algunos servicios se perdieran o se demoraran más de lo esperado. Por lo tanto, se concluye que el modelo multi-instancia aplicado al XAVIA PACSServer 4.0 no es capaz de soportar una carga superior a 499 servicios en el entorno sobre el que se realizaron las pruebas como se refleja en la iteración número tres.

Como solución a las respuestas no satisfactorias que fueron obtenidas durante la tercera iteración de las pruebas, se propone un cambio de entorno, dicho entorno debe contar con 32 GB de memoria RAM. De esta manera, se podrá garantizar un mejor rendimiento y una mayor satisfacción de los usuarios.

Conclusiones Parciales

Como parte de la validación del sistema se realizó un estudio de los métodos y herramientas que contribuyen a la realización de las diferentes pruebas, obteniendo variedad de respuesta en ambos casos, aunque por las características y particularidades del servicio que se ofrece se selección *JMeter* como herramienta de prueba. Mediante la cual se realizaron pruebas de rendimiento como es el caso de las pruebas de estrés y carga. En este atendiendo el entorno que fue seleccionado y las propiedades se obtuvo una prueba de carga con resultados satisfactorios, no siendo así en el caso de la prueba de estrés donde la muestra tomada fue superior a la muestra de carga, provocando que no todas las solicitudes fueran atendidas. Además, se obtuvo el diseño de despliegue para el sistema XAVIA PACSServer 4.0.

Conclusiones generales

Una vez configurada la multi-instancia de la presente investigación, cumplidas las tareas de investigación y el objetivo general, se obtienen las siguientes conclusiones:

- La arquitectura multi-instancia es una de las de mayor contribuciones de la computación en la nube, aportando resiliencia, flexibilidad, escalabilidad y disminución de los gastos.
- La propuesta de despliegue basada en contenedores permitió trazar una nueva estrategia para el proceso de planificación de un despliegue de sistemas centralizados con arquitectura multi-instancia.
- Las pruebas de software permitieron comprobar la calidad del resultado obtenido y la validez de su funcionamiento en el contexto del rendimiento una vez que tuvo múltiples instancias del sistema XAVIA PACSServer 4.0 en ejecución al mismo tiempo.
- Se trazó una nueva estrategia para el despliegue del sistema XAVIA PACSServer 4.0 una vez aplicado a este el comportamiento multi-instancia.

Recomendaciones

El proceso de configuración de la multi-instancia se realiza a través del cliente de *Docker.* Este cliente es una interfaz difícil de usar para muchos, debido a que se realiza a través de comandos y llamadas al sistema, por esta razón se recomienza la realización de una aplicación web, la cual se encargue de la administración de las instancias del sistema, esto proporcionaría una forma más amena y sencilla para el usuario encargado de gestionar el sistema.

Bibliografía

Alfredo Rodríguez Díaz, María Vidal Ledo,Ariel Delgado Ramos,Bárbara D. Martínez González,Karel Barthelemy Aguilar. 2018. *La Computación en la nube en la salud en Cuba.* la habana : s.n., 2018.

2023. Amazon Web Services, Inc. *AWS | Informática en la nube. Ventajas y Beneficios.* [En línea] 2023. [Citado el: 6 de 10 de 2023.] https://aws.amazon.com/es/what-is-cloud-computing/.

2020. ambit. *ambit.* [En línea] 9 de 1 de 2020. [Citado el: 6 de 10 de 2023.] https://www.ambit-bst.com/blog/definición-de-iaas-paas-y-saas-en-qué-se-diferencian.

Arturo Orellana García a , Leodan Vega Izaguirre , Gerardo Ceruto Marrero , Arianne Méndez Mederos , Marien Díaz Ruíz, Filiberto López Cossio , Lissette Soto Pelegrín . *La plataforma XAVIA PACS RIS y su aporte al dignóstico por imágenes médicas digitales en Cuba.* La habana : s.n.

Avi. 2023. Geekflare. *Arquitectura Docker y sus componentes para principiantes.* [En línea] 25 de 9 de 2023. [Citado el: 6 de 10 de 2023.] https://geekflare.com/es/docker-architecture/.

Bach. Rosales Torres, Jimy Estivens. 2022. *Modelo de Aceptación Tecnológica Para Determinar el Nivel de uso de Computación en la Nube en Las Empresas del Sector Tecnológico.* HUARAZ - PERU : s.n., 2022.

Cáceres, Kariné Pupo. 2011. *Estrategia de despliegue para la Solución de Software.* 2011.

Centro de Informática Médica, CESIM. Departamento de Aplicaciones, CESIM. Grupo de Mercadotecnia, CESIM. 2017. *Manual de Usuario.* La Habana, Cuba. : Centro de Diseño. Departamento creativo. UCI., 2017.

2023. Containerize. *Containerize.* [En línea] 21 de 2 de 2023. [Citado el: 21 de 9 de 2023.] https://blog.containerize.com/es/what-is-multitenancy-why-a-multi-tenant-approach-2/.

Cuervo, Víctor. 2020. Arquitecto IT. *Arquitecto IT.* [En línea] 20 de 10 de 2020. [Citado el: 28 de 9 de 2023.] https://arquitectoit.com/drupal/que-es-drupal/.

Delgado, Antonio. 2020. geeknetic. *geeknetic.* [En línea] 21 de 11 de 2020. [Citado el: 20 de 10 de 2023.] https://www.geeknetic.es/Dropbox/que-es-y-para-que-sirve.

2022. Deploy Drupal CMS on Oracle Linux with MySQL Database Service. *Deploy Drupal CMS on Oracle Linux with MySQL Database Service.* [En línea] 12 de 4 de 2022. [Citado el: 28 de 9 de 2023.] https://docs.oracle.com/es/solutions/drupal-with-mds/index.html#GUID-69A4EBE8-85F4-45CB-A16C-D1D1285C792A.

2023. Docker docs. [En línea] 2023. [Citado el: 15 de 10 de 2023.] https://docs.docker.com/compose/.

Docker Docs. [En línea] https://docs.docker.com/engine/.

2023. docs.docker. *docs.docker.* [En línea] 2023. [Citado el: 20 de 10 de 2023.] https://docs.docker.com/engine/swarm/key-concepts/.

2021. Drupal.org. *Drupal.org.* [En línea] 23 de 9 de 2021. [Citado el: 28 de 9 de 2023.] https://www.drupal.org/docs/multisite-drupal/multisite-drupal-considerations.

2022. ExpandLatam. *ExpandLatam.* [En línea] 29 de 4 de 2022. [Citado el: 3 de 10 de 2023.] https://www.expandlatam.com/blog/crm/que-es-salesforce-crm/.

Fernandez, Ana Julia. 2022. E-Saurio. *E-Saurio.* [En línea] 23 de noviembre de 2022. [Citado el: 28 de 9 de 2023.] https://blog.e-saurio.com/multitenancy-que-es-y-por-que-debes-saber-al-respecto/.

2023. GeeksforGeeks. *GeeksforGeeks.* [En línea] 16 de 2 de 2023. [Citado el: 28 de 9 de 2023.] https://www.geeksforgeeks.org/oracle-architecture/.

Gómez, Enrique Serrano. 2019. *Checkpoint and restore of Singularity containers.* 2019.

Grupo Aire. 2023. stackscale. *stackscale.* [En línea] 23 de 3 de 2023. [Citado el: 28 de septiembre de 2023.] https://www.stackscale.com/es/blog/cloud-computing-guia-definitiva/#Capitulo_1_Que_es_el_cloud_computing.

Grupo Aires. 2023. stackscale. *stackscale.* [En línea] 31 de 8 de 2023. [Citado el: 27 de 9 de 2023.] https://www.stackscale.com/es/blog/servidor-cloud/.

2021. HadsonParedes. *HadsonParedes.* [En línea] 21 de 9 de 2021. [Citado el: 27 de 9 de 2023.] http://blog.hadsonpar.com/2021/09/arquitectura-multi-tenant.html.

Hernández, Yasmany Núñez. 2012. *Despliegue del SCADA-UX para Energía UCI.* 2012.

Herramientas CASE para ingieneria de requisitos. **Andrea Alarón, Eva Sandoval. 2008.** 2008.

Historia de la web . *Historia de la web .* [En línea] [Citado el: 21 de 10 de 2023.] http://charliedaw2236.blogspot.com/p/arquitectura-cliente-servidor.html.

Juan, Jordi Fornes de. 2021. *Arquitectura de Servicios con DockerSwarm.* Barcelona,Catalunya : s.n., 2021.

Larry Garfield. 2012. Palantir.net. *Palantir.net.* [En línea] 9 de 4 de 2012. [Citado el: 28 de 9 de 2023.] https://www.palantir.net/blog/multi-headed-drupal.

Leifer Mendez, Marianna Rolfo. 2022. Codigoencasa.com. [En línea] 5 de enero de 2022. [Citado el: 5 de diciembre de 2023.] https://codigoencasa.com/multi-tenancy/.

Luz Elena Gutiérrez López, Carlos Andrés Guerrero Alarcón. 2020. *Extensión de la arquitectura Docker para el despliegue automático de contenedores.* Barranquilla : s.n., 2020. 29.

Maidelyn Piñero González, Aymara Marin Diaz, Yaimí Trujillo Casañola, Denys Buedo Hidalgo. 2021. Revista Cubana de Ciencias Informáticas. *scielo.* [En línea] enero-marzo de 2021. [Citado el: 21 de 10 de 2023.] http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992021000100089.

Mario Germán Castillo Ramírez, Luisa Maria Muñoz Valencia. 2021. *Implementación de servicios con la tecnología de contenedores PODMAN.* Cali - Colombia : Servicio Nacional de Aprendizaje, SENA, 2021.

2023. microsoft . *¿Qué es PaaS? Plataforma como servicio | Microsoft Azure.* [En línea] 2023. [Citado el: 6 de 10 de 2023.] https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-paas.

2023. Microsoft Azure. [En línea] 2023. [Citado el: 6 de 10 de 2023.] https://azure.microsoft.com/es-mx/resources/cloud-computing-dictionary/what-is-a-cloud-server/.

2023. microsoft Ignite. *microsoft Ignite.* [En línea] 14 de 10 de 2023. [Citado el: 20 de 10 de 2023.] https://learn.microsoft.com/es-es/visualstudio/docker/tutorials/docker-tutorial.

Moreno, Federico Alonzo Fernández. 2016. *Diseño e implementación de herramientas para la instalación y la mejora de la administración y la seguridad de un gestor de identidad basado en openstack.* Madrid : Universidad politécnica de Madrid , 2016.

2023. Oracle Help Center. *Oracle Help Center.* [En línea] 28 de 6 de 2023. [Citado el: 28 de 9 de 2023.] https://oracle-base.com/articles/12c/multitenant-overview-container-database-cdb-12cr1.

Panizzi, Marisa Daniela. 2022. *Modelo de Proceso de Despliegue de Sistemas de Software.* 2022.

2022. RedHat. *RedHat.* [En línea] 25 de 3 de 2022. [Citado el: 20 de 9 de 2023.] https://www.redhat.com/es/topics/cloud-computing/what-is-saas.

2023. Run a Single Database across Multiple Servers. *OCI.* [En línea] 2023. [Citado el: 26 de 9 de 2023.] https://www.oracle.com/database/real-application-clusters/.

SanchezAlmenares, Liudmila. 2019. Prueba Automática de Carga y Estrés . *Serie Científica De La Universidad De Las Ciencias Informáticas.* [En línea] 2019. [Citado el: 22 de 10 de 2023.] https://publicaciones.uci.cu/index.php/serie/article/view/339/232.

2023. Seidor. *Seidor.* [En línea] 15 de 3 de 2023. [Citado el: 3 de 10 de 2023.] https://www.seidor.com/blog/arquitectura-salesforce-multitenant.

Sharma, Rajnish Kumar. 2022. Net solutions. [En línea] 4 de 4 de 2022. [Citado el: 28 de 9 de 2023.] https://www.netsolutions.com/insights/5-reasons-why-you-should-choose-multi-tenant-architecture-for-your-saas-application/.

Sommerville, Ian. 2011. *Ingeniería de de Software.* México : Printed in Mexico, 2011.

2023. Universidad de Valladolid Repositorio Documental. *Universidad de Valladolid Repositorio Documental.* [En línea] 2023. [Citado el: 21 de 10 de 2023.] https://uvadoc.uva.es/handle/10324/60421.

Vega, Adrian Alonso. 2018. Mi blog. *Mi blog.* [En línea] 23 de 12 de 2018. [Citado el: 28 de 9 de 2023.] https://adrianalonso.es/arquitectura-del-software/enfoques-de-arquitecturas-multitenant-para-aplicaciones-saas/.

Wheeler, Patrick. 2019. *Oracle Multitenant with Oracle Database 19c.* EE. UU : s.n., 2019.

—. 2020. Oracle Real Application Clusters 19c Técnico Arquitectura. *ORACLE WHITE PAPER .* EE.UU : s.n., 2020.

Yépez, Adán Gomezcoello. 2017. *Multitenencia Cloud: Una Revición Sistemátoca de la literatura.* 2017.

—. 2017. *Multitenencia cloud: Una revisioón sistematica de la literatura .* madrid : s.n., 2017.