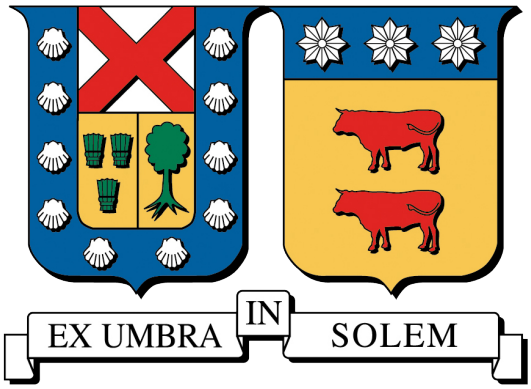
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



Automatización de la Administración de Infraestructura para el Sistema de Soporte de Decisiones SIPAT

EDUARDO ARTURO TOLEDO DONOSO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

PROFESOR GUÍA	:	Sr. Javier Cañas Robles
PROFESOR CORREFERENTE	:	Sra. Claudia López Moncada

Julio 2018

Dedico ésta memoria a mi familia, por ser los pilares fundamentales en todo lo que soy, en mi educación, tanto académica, como de la vida y por su incondicional apoyo perfectamente mantenido a través del tiempo.
Todo este trabajo ha sido posible gracias a ellos.

Esta memoria fue financianciada parcialmente por el proyecto FONDEF IT15I10001 denominado “Sistema de Soporte de Decisiones para la Evaluación del Peligro Por Tsunamis”.

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres y hermana por apoyarme en toda mi formación académica y de la vida, por entregarme las herramientas y valores necesarios que me convierten en la persona que soy hoy en día, los quiero mucho.

A toda mi familia: tíos, primos, abuelos, etc, que estuvieron preocupados por mi durante todos mis estudios y aún más hoy. A cada uno de ellos los tengo presente cada día.

A Lucía, que desde el comienzo me incentivó y apoyó en terminar mi carrera. Gracias por tu apoyo incondicional.

A mis amigos que se mantienen desde mi ciudad Santa Cruz: Alexis, Ismael, Sebastian, Cote. Nos conocemos desde hace muchos años y espero que esta amistad se siga manteniendo por los muchos años venideros.

A la Sra Marta y toda su familia, gracias permitirme conocer a su familia y las múltiples invitaciones a tomar once en su casa y las conversaciones posteriores.

A mis amigos de universidad: Roberto, Jose, Matias, Nacho, Martín, Mauricio, Christhoper, Rojo, Felo, Maca y muchos otros que se me escapan en este instante, gracias todos los momentos vividos.

A mis amigos de Admisión de la universidad, el tiempo que trabaje con ustedes lo disfrute mucho, son un gran equipo.

A todos los integrantes de Infraestructura y Tecnología del DI: Prof. Miguel, Yonathan, Carol, Victor, Adbias, Geordy. Son un gran equipo y he aprendido bastante con ustedes.

A todo el equipo SIPAT: Prof. Javier Cañas, Prof. Patricio Catalán, Alejandra Gubler, Carlos Valdes, Leonardo Pizarro, Rene Mena por todo su apoyo durante la realización de esta memoria.

Al equipo You Can Run, llevo poco tiempo entrenando con ustedes, pero me he dado cuenta del gran equipo que son.

A todos los demás que se me escapan en este minuto, les agradezco por todo.

Gracias totales.

RESUMEN EJECUTIVO

Un sismo puede producir un evento tsunamigénico y causar daño a la población aledaña de zonas costeras. El sistema SIPAT *"Sistema Integrado de Predicción y Alerta de Tsunamis"*, genera alertas de tsunamis a partir de un evento sísmico. Este sistema es alimentado por miles sismos pre-modelados de diferentes magnitudes y distribuidos a lo largo de Chile. Luego de un proceso de toma de decisiones, SIPAT genera alertas para diversas zonas del país. Dichos sismos son modelados usando el poder de procesamiento de máquinas virtuales. En virtud de cambios en los datos entrantes y observando una sobrecarga de trabajo del uso de las máquinas virtuales, resulta primordial proponer un nuevo proceso de modelación, centrado en el uso eficiente de recursos computacionales.

Palabras Claves: Docker, máquinas virtuales, SHOA, modelamiento computacional, sismo, tsunami, SIPAT.

ABSTRACT

Earthquake can produce a tsunamigenic event potentially damaging nearby coastal areas. SIPAT (Sistema Integrado de Predicción y Alerta de Tsunamis) system generates tsunami alerts based on seismic events. This system is fed by thousands of pre-modeled earthquakes from different magnitudes and distributed throughout Chile. After a decision-making process, SIPAT generates alerts for diverse geographic locations around the country. These earthquakes are modeled using virtual-machine-based processing power. Based on changes from input data and observing a work overload from the usage of virtual machines, it is essential to propose a new modeling process focused on the efficient usage of computational resources.

Keywords: Docker, virtual machines, SHOA, Computational modeling, seism, tsunami, SIPAT.

GLOSARIO

- **SISMO CAMPO LEJANO:** Sismo que ocurre en territorio que no es nacional o cercano a éste, es decir, fuera del polígono de Campo Cercano definido por el SNAM.
- **SISMO CAMPO CERCANO**: Sismo que ocurre en Chile o en una zona cercana al territorio Nacional, específicamente dentro del polígono definido por el SNAM.
- **SNAM**: Sistema Nacional de Alerta de Maremotos.
- **SHOA**: Servicio Hidrográfico y Oceanográfico de la Armada de Chile.
- **SIPAT**: Sistema Integrado de Predicción y Alerta de Tsunamis.
- **SIVET**: Sistema Integrado de Visualización de Eventos Tsunamigénicos.
- **oVirt**: Orquestador para virtualización de código libre.
- **FONDEF**: Fondo de Fomento al Desarrollo Científico y tecnológico.
- **PTWC**: Centro de alertas de tsunamis del Pacífico.
- **ONEMI**: Oficinal nacional de emergencia del Ministerio del Interior.
- **GUI**: Interfáz gráfica de usuario.
- **COMCOT**: *Cornell Multi-grid Coupled Tsunami Model*.
- **Python**: Lenguaje de programación interpretado popular entre los desarrolladores.
- **Centos**: Sistema operativo de código libre basado en RedHat.
- **CPU**: Unidad central de procesamiento.
- **RAM**: Memoria de acceso aleatorio.
- **HPC**: Computación de alto rendimiento.
- **Rancher**: Orquestador de contenedores Docker.

- **CLI**: Interfaz de linea de comandos.
- **API**: Interfaz de programación de aplicaciones.

Índice de Contenidos

1. Definición del Problema **1**
1.1. Introducción . . . 1
1.1.1. SHOA . . . 2
1.1.2. Trabajos previos . . . 3
1.2. Situación Actual . . . 6
1.2.1. Infraestructura Tecnológica . . . 6
1.3. Problema a Resolver . . . 7
1.3.1. Situación Actual Modelamiento . . . 7
1.4. Objetivos . . . 8
1.4.1. Objetivo Principal . . . 8
1.4.2. Objetivos Específicos . . . 8
1.5. Estructura del Documento . . . 9

2. Estado del Arte **10**
2.1. Virtualización . . . 10
2.1.1. Conceptos importantes en virtualización . . . 10
2.1.2. Tipos de virtualización de CPU . . . 11
2.2. Docker . . . 15
2.2.1. Componentes de Docker . . . 16
2.2.2. Máquinas virtuales v/s Contenedores . . . 19
2.3. HPC con contenedores . . . 20

3. Diseño de Solución **23**
3.1. Hardware base para el proyecto . . . 23
3.1.1. Servidores . . . 23
3.1.2. Almacenamiento . . . 24
3.1.3. Redes . . . 25
3.2. Docker . . . 26
3.2.1. Conceptos Generales . . . 26
3.2.2. COMCOT para modelamiento . . . 27
3.2.2.1. Archivos de entrada . . . 27
3.2.2.2. Archivos de salida . . . 29
3.2.3. *Imagen* para modelamiento . . . 29
3.3. Almacenamiento para Docker . . . 30
3.3.1. Global File System 2 (GFS2) . . . 30

3.3.2. GlusterFS . . . 31
3.3.3. Almacenamiento individual . . . 33
3.4. Interfaces de administración de containers Docker . . . 34
3.5. Rancher como orquestador de contenedores . . . 40
3.5.1. Conceptos Generales . . . 41
3.5.2. Rancher CLI y API . . . 42
3.5.3. Configuración propuesta de Rancher . . . 42
3.6. Resumen Solución Propuesta . . . 43
3.6.1. Solución lógica . . . 43
3.6.2. Configuración física . . . 44

4. Construcción e implementación de la solución **46**
4.1. Instalación base del sistema operativo . . . 46
4.2. Instalación y configuración de Docker . . . 47
4.2.1. Registry . . . 47
4.2.1.1. Registry Local . . . 48
4.2.1.2. Registry Mirror . . . 48
4.2.2. Imágenes Docker . . . 49
4.3. Almacenamiento . . . 50
4.3.1. GFS2 . . . 50
4.3.2. Gluster . . . 51
4.3.3. Almacenamiento para Docker . . . 52
4.4. Instalación y configuración de Rancher . . . 52
4.4.1. Rancher CLI . . . 53
4.4.2. Contenedor *janitor* para limpieza . . . 54
4.5. Monitoreo del sistema . . . 55
4.5.1. Monitoreo del sistema operativo . . . 55
4.5.2. Monitoreo de Rancher . . . 55
4.5.3. Monitoreo de Gluster . . . 55

5. Validación de la Solución **56**
5.1. Lecciones aprendidas . . . 56
5.1.1. Modelamiento . . . 56
5.1.2. Almacenamiento . . . 58
5.2. Análisis comparativo de uso de recursos . . . 60
5.3. Análisis comparativo de uso de tiempo . . . 61
5.3.1. Comparación con máquinas virtuales . . . 62

6. Conclusiones **67**
6.1. Conclusiones Generales . . . 67
6.2. Conclusiones Específicas . . . 67
6.3. Trabajo Futuro . . . 69

Bibliografía **71**

A. Anexos **73**
A.1. Instalación de docker-distribution y registry 73
A.2. Plan de mantenimiento 74
A.3. Tiempos de modelamiento expresado en horas 74

Índice de Tablas

2.1. Máquinas virtuales v/s Docker. Fuente: Elaboración propia. 20

3.1. Listado de servidores alojados en SNAM. Fuente: Elaboración propia. 24
3.2. Características de Storage. Fuente: Elaboración propia. 25
3.3. Resumen de la elección de la interfaz para Docker. Fuente: Elaboración propia. . . 40

5.1. Speedup de simulaciones en un servidor. Fuente: Elaboración propia. 65
5.2. Tabla comparativa de tiempos y recursos entre máquinas virtuales y contenedores Docker. Fuente: Elaboración propia. 65
5.3. Tabla comparativa de oVirt con Rancher. Fuente: Elaboración propia. 66

6.1. Speedup de simulaciones en un servidor. Fuente: Elaboración propia. 69

A.1. Tiempos de modelación por sismos utilizando todos los servidores disponibles. Fuente: Elaboración propia. 75

Índice de Figuras

2.1. *Hypervisor Tipo 1*. Fuente: Elaboración propia. 11
2.2. *Hypervisor Type 2*. Fuente: Elaboración propia. 12
2.3. Esquema de *full virtualization*. Fuente: http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm 13
2.4. Esquema de *para-virtualization*. Fuente: http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm 13
2.5. Esquema de *os-virtualization*. Fuente: http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm 14
2.6. Antes y después de la virtualización. Fuente: IBM 15
2.7. Infraestructura de Docker. Fuente: http://www.docker.com 17
2.8. Arriba: arquitectura de máquinas virtuales. Abajo: arquitectura de docker. Fuente: http://www.docker.com 21

3.1. Esquema ejemplificador de conexión bonding, en este caso en una máquina virtual. Fuente: oracle.com 25
3.2. Implementación GFS2. Fuente: Elaboración propia. 31
3.3. Esquema propuesto de volumen distribuido. Fuente: gluster.org 33
3.4. Configuración de almacenamiento individual para servidores. Fuente: Elaboración propia. 34
3.5. Interfaz de *Simple Docker UI. Fuente: https://github.com/felixgborrego/simple-docker-ui* 35
3.6. Interfaz gráfica de Portainer.io. Fuente: http://portainer.io 36
3.7. Interfaz de gestión de *Shipyard*. Fuente: https://shipyard-project.com . . . 37
3.8. Interfaz de *Panamax*. Fuente: http://panamax.io 38
3.9. Interfaz de *Rancher*. Fuente: http://rancher.com 39
3.10. Solución lógica propuesta. Fuente: Elaboración propia. 44
3.11. Solución física propuesta. Fuente: Elaboración propia. 45

5.1. Tiempos de Modelación en Horas. Fuente: Elaboración propia. 63

1 | Definición del Problema

Este capítulo tiene como fin presentar la problemática a tratar a lo largo de la memoria, exponiendo un contexto del ámbito de trabajo, objetivo principal y objetivos específicos. Adicionalmente se muestra la estructura del documento.

1.1. Introducción

Chile se ubica en el hemisferio sur del continente Americano y convive con el llamado Cinturón de Fuego del Pacífico, [1] el cual lo hace un país altamente sísmico y al encontrarse bañado por las aguas del océano Pacífico es proclive a recibir eventos tsunamigénicos.

La evaluación que se realiza a partir de un evento sísmico que determina la posibilidad de ocurrencia de un tsunami demanda gran poder computacional y tiempo de espera que resulta de vital importancia al momento de tomar una decisión de evacuación del borde costero. Para realizar esta evaluación de una manera más rápida, se planteó en el proyecto FONDEF D11I1119 (años 2012 a 2016) denominado *Diseño e Implementación de una Base de Datos de Predicción del Peligro por Tsunamis para la Costa Chilena utilizando Modelación Computacional de Alto Rendimiento*; la construcción de una base de datos de sismos simulados junto a un sistema de toma de decisiones que apoyara la labor que realiza el Sistema Nacional de Alarma de Maremotos (SNAM) dependiente del Servicio Hidrográfico y Oceanográfico de la Armada (SHOA).

El proceso de poblamiento de la base de datos se realiza mediante simulaciones computacionales para diversos puntos geográficos a lo largo del país. Cada una de éstas simulaciones se ejecuta en una máquina virtual que es controlada por el orquestador oVirt, [2] de las cuales se obtienen pronósticos de

[1] https://www.thoughtco.com/ring-of-fire-1433460
[2] https://ovirt.org/

amenaza de tsunami en base a sismos de diferentes magnitudes y posteriormente son almacenados en una base de datos. Mientras más resultados de simulaciones se tengan modelados, más exacto será la predicción de peligro de un tsunami.

Éste proyecto finalizó de manera exitosa el año 2016 y gracias a otro proyecto FONDEF IT15I10001 de continuación tecnológica sobre el cual esta memoria se sustenta, se busca consolidar y dar robustez a este sistema dados nuevos requerimientos que han surgido luego de su puesta en marcha.

Esta memoria abarca lo relativo a la simulación computacional de eventos sísmicos buscando optimizar su proceso mediante la adopción de nuevas herramientas tecnológicas que han surgido en el último tiempo, y así, lograr menores tiempos de simulación por cada sismo y una menor sobrecarga de trabajo en los servidores de modelamiento.

1.1.1. SHOA

El Servicio Hidrográfico y Oceanográfico de la Armada de Chile, también conocido por su acrónimo SHOA, tiene como misión proporcionar asistencia técnica e informaciones destinadas a dar seguridad a la navegación en las vías fluviales y lacustres, tanto dentro de las aguas interiores y mar territorial chileno como también en la alta mar contigua al litoral de Chile. Asimismo, constituye el servicio oficial, técnico y permanente del Estado, en todo lo que se refiere a Hidrografía, levantamiento hidrográfico marítimo, fluvial y lacustre, cartografía náutica, confección y publicación de cartas de navegación de aguas nacionales, oceanografía, planificación y coordinación de todas las actividades oceanográficas nacionales relacionadas con investigaciones físico-químicas, mareas, maremotos, geografía náutica, navegación, astronomía, señales horarias oficiales, aerofotogrametría aplicada a la carta náutica y señalización marítima.

Desde 1966, el SHOA opera el Sistema Nacional de Alarma de Maremotos (SNAM) y representa oficialmente al Estado de Chile ante el Sistema Internacional de Alerta de Tsunamis del Pacífico, cuyo centro de operaciones es el *Pacific Tsunami Warning Center* (PTWC) ubicado en Hawaii (Estados Unidos). Como su nombre lo indica, SNAM es el encargado de evaluar técnicamente si la ocurrencia de un evento sísmico puede producir un tsunami en las costas del país y remitir dicho resultado a la Oficina Nacional de Emergencia del Ministerio del Interior (ONEMI) para su posterior divulgación a la población.

SHOA es la entidad mandante del presente proyecto FONDEF IT15I10001 *Sistema de Soporte de Decisiones para la Evaluación del Peligro por Tsunamis*; y es la que indica los requerimientos de uso y valida la solución propuesta.

1.1.2. Trabajos previos

Existen trabajos previos realizados por memoristas anteriores que se encargaron de diseñar e implementar el Sistema Integrado de Predicción y Alerta de Tsunamis (SIPAT), los cuales se resumen a continuación.

Dentro de los primeros memoristas del proyecto se tiene a Gabriel Ruiz (titulado en 2013) [10] quien definió la arquitectura de software para el proyecto Sistema Integrado de Predicción y Alerta de Tsunamis (SIPAT), en la cual logró de manera exitosa:

- Establecer el flujo de información actual en los sistemas utilizados por SNAM para establecer alertas de tsunami.
- Diseñar un flujo de información nuevo para el software de generación de alertas de tsunami.
- Determinar casos de uso, separación de componentes y configuración de despliegue adecuada para el óptimo funcionamiento del sistema, considerando su importancia como herramienta para salvar vidas.
- Definir un estructura de hardware acorde a las necesidades del sistema.

Quedando como trabajo futuro toda la implementación de la infraestructura, con el desarrollo de software y despliegue de la estructura de hardware final.

De manera paralela se encontraba trabajando Diego Yachan (titulado en 2014) [15] cuyo objetivo principal fue "*Crear algoritmos y programas que permitan extraer parámetros desde una base de datos de escenarios sísmicos pre-calculados para ser incorporados al Sistema Integrado de Predicción y Alerta de Tsunamis"*. Realizó una comparación de algoritmos en diferentes escenarios a fin de concluir el que se adaptaba mejor al proyecto. Es así como logró su objetivo principal, además de los siguientes objetivos secundarios:

- Establecer algoritmos de búsqueda considerando mínimo riesgo y mejor tiempo de respuesta.

- Generar recomendaciones para integración con otros sistemas.

De esta memoria quedó como trabajo futuro determinar cuales de los métodos propuestos van a ser utilizados, ya que en ese tiempo no se tenían los escenarios de modelación finales. Además, quedó pendiente la definición de la cantidad de puntos de pronóstico y el hardware final que se utilizaría.

Avanzando hacia el segundo año del proyecto se encuentra la memoria de Carlos Valdés (titulado en 2015) [12] titulada *Sistema Integrado de Visualización de Eventos Tsunamigénicos* en la que se diseña una arquitectura para una interfaz gráfica de usuario (GUI), denominada Sistema Integrado de Visualización de Eventos Tsunamigénicos (SIVET), y que permitiera entregar apoyo al SIPAT en el seguimiento de proceso operacionales de éste. En esta memoria se destacan varios objetivos específicos:

- Identificar las etapas de la cadena operacional del SIPAT, de manera de diseñar una interfaz gráfica de usuario que permita el seguimiento y control de los procesos realizados.
- Proponer los indicadores, mapas y gráficos para las interfaces de la cadena operacional del SIPAT en base a los estándares internacionales de centros de alerta de tsunamis.
- Crear un modelo de base de datos escalable para los resultados generados por las simulaciones en COMCOT y que sean de relevancia desplegar en el SIVET.
- Determinar los casos de uso, componentes, diagramas y configuración de despliegue adecuado para el óptimo funcionamiento del sistema de visualización propuesto.
- Determinar una arquitectura de hardware acorde a las necesidades del sistema SIVET.
- Identificar los requerimientos para futuros desarrollos del SIVET.

Al término de la memoria se cumplió exitosamente el objetivo principal y los objetivos secundarios.

Leonardo Pizarro (titulado en 2014) [8] en su memoria *Implementación del Sistema Integrado de Predicción y Alerta de Tsunami, SIPAT*, buscó como objetivo general implementar un sistema adaptable a los posibles cambios de requerimientos, mediante un diseño modular que permitiera enfocar

el desarrollo del sistema en el rendimiento, la capacidad de ser administrable y con redundancia de sus componentes, a nivel de hardware y software. Sus objetivos específicos fueron:

- Implementar un sistema de virtualización que provea de alta disponibilidad y redundancia al SIPAT.
- Realizar un análisis de seguridad de forma de incorporar estándares correspondientes, teniendo en cuenta la sensibilidad del sistema dado su propósito.
- Validar el proceso de poblamiento de la base de datos.

Además de concluir exitosamente todos los objetivos planteados se entregó una serie de lineamientos en los que se propone como trabajo futuro cambios en las tablas de la base de datos que apuntan a una mejora en los tiempos de extracción de la data que éstas contienen y la sugerencia de crear una versión mas "liviana"de SIPAT a fin de que pueda ser instalada en un computador portátil o algún hardware similar para hacer frente a un estado de catástrofe de gran magnitud.

Samuel Miranda (titulado en 2015) [7] en su memoria titulada *Optimización e Integración de un Sistema de Alerta y Prevención de Tsunami, SIPAT*, desarrolla el objetivo principal de llevar el prototipo que existe actualmente de SIPAT a un sistema en producción, optimizado e integrado en el SHOA, listo para ser utilizado en casos de un sismo que pueda provocar un evento tsunamigénico en las costas del país. Y junto a este objetivo principal, se cumplieron los siguientes objetivos específicos:

- Implementar las funcionalidades faltantes del subsistema Extractor de SIPAT:
  - Llenado rápido de datos sísmicos.
  - Generación y guardado de boletines por ciudades y por bloques de costa.
  - Revisión de boletines previos.
- Actualizar las funcionalidades deprecadas del subsistema Extractor de SIPAT
  - Actualización de boletines.
  - Cancelación parcial de boletines.
  - Cancelación total de boletines.

    - Generación de archivos *kml*.

- Diseñar e implementar un modelo de toma de decisiones que indique, basándose en los escenarios ya simulados, en qué zonas costeras del país un evento sísmico real producirá un tsunami, cuanto tiempo se posee para evacuar y su nivel de alerta.

- Optimizar el sistema SIPAT para reducir los tiempos de búsqueda en la base de datos y de la toma de decisiones, por medio de la implementación de datos geoespaciales, un análisis detallado de su interfaz y mejoras al sistema.

1.2. Situación Actual

Esta memoria presenta una continuación del trabajo realizado por memoristas anteriores que se encargaron de diseñar e implementar el sistema en una primera etapa. Ahora junto a un nuevo proyecto FONDEF de continuación tecnológica, se busca consolidar el sistema con miras a nuevos requerimientos de parte de SHOA.

Uno de los componentes de SIPAT es el modelamiento de sismos. Actualmente este proceso se realiza en máquinas virtuales que se crean con la ayuda de un *script* en lenguaje Python sobre una plataforma de virtualización llamada oVirt. La sobrecarga que presentan las máquinas virtuales se ha convertido en una limitante que es necesario solventar para poder aumentar la cantidad de escenarios en modelamiento de manera simultánea y, en lo posible, disminuir los tiempos de modelación.

1.2.1. Infraestructura Tecnológica

La infraestructura tecnológica que sustenta a SIPAT radica en su totalidad en los datacenters del SHOA, la que ha ido aumentando en número de servidores desde el proyecto FONDEF anterior al actual.

Entre las principales características se puede mencionar que se cuenta con servidores físicos conectados mediante una red de fibra óptica, además de dos servidores dedicados para almacenamiento. Los datacenters están dotados de sistemas de redundancia energética y climatización acorde a las directrices de la industria tecnológica.

Junto a esta infraestructura de datacenters, se tiene un lugar conocido como *SHELTER*, el cual corresponde a una unidad de emergencia y respaldo en caso que la sala principal de alerta de tsunami (SNAM) no estuviese operativa. Cuenta enlaces satelitales y equipos portátiles que permiten difundir información de eventos tsunamigénicos en casos de sismos de gran magnitud que dañen de forma considerable la infraestructura física de los datacenters y la conectividad de internet a nivel país.

Además en cuanto al aspecto de software, Python es principal lenguaje de programación utilizado y Centos el sistema operativo instalado en los servidores. En esta memoria se tiene en consideración estas especificaciones para continuar en la misma línea tecnológica.

Respecto al desarrollo de software del proyecto FONDEF anterior, existen interfaces web para que personal del SHOA pueda ingresar a modelar sismos como también para consultar sismos modelados previamente. Adicional a esta interfaz, existe un sistema de visualización web que permite a personal del SNAM ejecutar la tarea de evaluar la ocurrencia de un tsunami a partir de un sismo. Esta última interfaz cuenta con paneles desplegados en la sala de SNAM para su mejor visualización. Cabe mencionar que estos paneles de visualización están siendo mejorados en una memoria paralela esta, proponiendo nuevas vistas y re-escribiendo las existentes en lenguajes de programación modernos y que permiten adaptarse mejor a los casos de uso del SHOA.

1.3. Problema a Resolver

Los eventos sísmicos que se modelan en el sistema se vuelven cada vez más complejos y los tiempos de simulación aumentan considerablemente. La infraestructura tecnológica que hace frente a esto ha llegado a un punto de gran optimización de software en cuanto a tiempos de respuesta y diseño de arquitectura. Buscar más optimización trae consigo la evaluación de nuevas tecnologías y mejoras en el diseño base de la arquitectura del sistema, como por ejemplo, la utilización de lenguajes de programación mas optimizados o nuevos paradigmas en cuanto a virtualización.

1.3.1. Situación Actual Modelamiento

Todo el proceso de modelación se realiza mediante la tecnología de máquinas virtuales la cual se encuentra fuertemente consolidada en el mundo tecnológico, pero a su vez posee una arquitectura que introduce sobrecarga en el desempeño cuando se hace uso crítico de recursos computacionales en

el sistema operativo virtualizado. Para hacer frente a esto, el equipo que se encargó de implementar la tecnología de virtualización utilizó oVirt[3] como orquestador y optimizó lo mejor posible sus componentes para obtener el mejor desempeño.

La sobrecarga de recursos computacionales debido a la simulación se mantiene por una gran cantidad de tiempo cercano a 1 día y el equipo de trabajo del FONDEF pronostica que ese tiempo para modelaciones que se implementarán durante el año, aumentará a cerca de una semana. Con estas nuevas modelaciones, se generan más resultados que utilizarán espacio adicional de disco y a la vez tiempo de CPU mayor que puede causar comportamientos no deseados por parte oVirt tales como reinicios de la maquina virtual.

La búsqueda de una nueva tecnología similar a oVirt es algo necesario teniendo en mente lo mencionado anteriormente.

1.4. Objetivos

Es importante definir el objetivo principal y los objetivos específicos emanados de éste, de esta forma se tiene delimitado el trabajo a realizar.

1.4.1. Objetivo Principal

- Proponer y evaluar un sistema alternativo a máquinas virtuales para modelamiento computacional de alto desempeño en el proyecto SIPAT.

1.4.2. Objetivos Específicos

1. Evaluar Docker como alternativa a máquinas virtuales de acuerdo al uso que se le da en el proyecto SIPAT.

2. Evaluar e implementar un orquestador para Docker que se adapte a la arquitectura del proyecto SIPAT.

3. Evaluar optimizaciones a la plataforma propuesta para obtener mejor rendimiento.

[3] https://www.ovirt.org/

4. Comparar resultados de tiempos de simulación entre máquinas virtuales y el sistema propuesto.

1.5. Estructura del Documento

El documento presenta la siguiente estructura de capítulos:

- Definición del problema: Se encarga de entregar un contexto de lo que será el presente trabajo, dando énfasis en memorias anteriores y los objetivos que se planean lograr.
- Estado del arte: Consiste en una recopilación de trabajos realizados hasta el momento en la comunidad científica y luego, a partir de esta base, se plantea la solución a ejecutar.
- Diseño de la solución: Posterior al desarrollo del Estado del Arte corresponde aplicar lo aprendido en la construcción de una solución, acorde a los requerimientos del Proyecto FONDEF y materializados en los objetivos planteados en esta memoria.
- Construcción e implementación de la solución: En este capítulo se reporta la solución propuesta, instalando, configurando y enlazando todas las aristas del sistema para que funcionen como un todo.
- Validación de la solución: Entrega resultados del despliegue de la propuesta y su análisis.
- Conclusiones: Para finalizar se recopilan los resultados de la validación y se contrastan con la propuesta realizada. Además, se plantea trabajo futuro que puede suceder a esta memoria.

2 | Estado del Arte

En este capítulo se describe el estado del arte de la tecnología utilizada en la solución propuesta. Se comienza revisando los tipos de virtualización y sus conceptos importantes, para luego dar paso a Docker[4] como alternativa para el modelado computacional.

Siguiendo en la línea de Docker, se estudia el uso de *High Performance Computer* con Docker. Finalmente se estudian las interfaces de administración disponibles para Docker en la actualidad.

2.1. Virtualización

Virtualización es la abstracción de recursos computacionales: CPU, almacenamiento, hardware de red, memoria; a fin de ser utilizados en la creación de máquinas virtuales. Una de sus definiciones es:

> Virtualización es un marco de trabajo o *framework* para dividir los recursos de un computador en múltiples entornos de ejecución, mediante la aplicación de una o más tecnologías o conceptos tales como partición de hardware y software, tiempo compartido (*time-sharing*), simulación parcial o completa de la máquina, emulación, calidad del servicio y muchos otros. Singh [2004] [11]

2.1.1. Conceptos importantes en virtualización

Hypervisor

Es la capa que se encarga de la virtualización. También se conoce como *Virtual Machine Monitor*. Es responsable de crear máquinas virtuales, supervisarlas y asegurarse de que los recursos estén

[4] https://www.docker.com/

asignados correctamente. Existen dos tipos:

- *Tipo 1*: Es considerado un *hypervisor nativo*, ya que se ejecuta directamente en el hardware del *host*. La figura 2.1 representa este tipo de *hypervisor*.

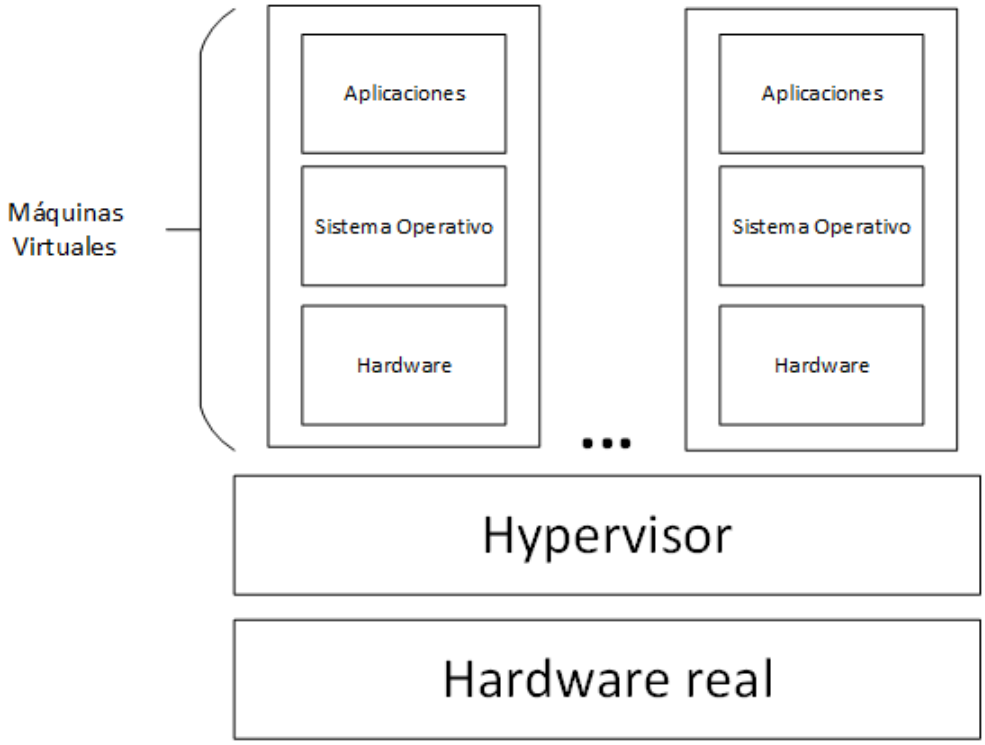


Figura 2.1: *Hypervisor Tipo 1*. Fuente: Elaboración propia.

- *Tipo 2*: Requiere de un sistema operativo sobre el cual se instalan los componentes necesarios para virtualizar. La figura 2.2 representa la arqyutectura de este tipo de *hypervisor*.

2.1.2. Tipos de virtualización de CPU

Existen diferentes tipos de virtualización para la puesta en marcha de máquinas virtuales.

Full virtualization

Proporciona una simulación completa del hardware. En esta virtualización el sistema operativo puede ejecutarse sin modificación en las máquinas virtuales, lo cual es una ventaja en caso de no contar con los códigos fuentes de dichos sistemas. Además de que cada máquina virtual se encuentra

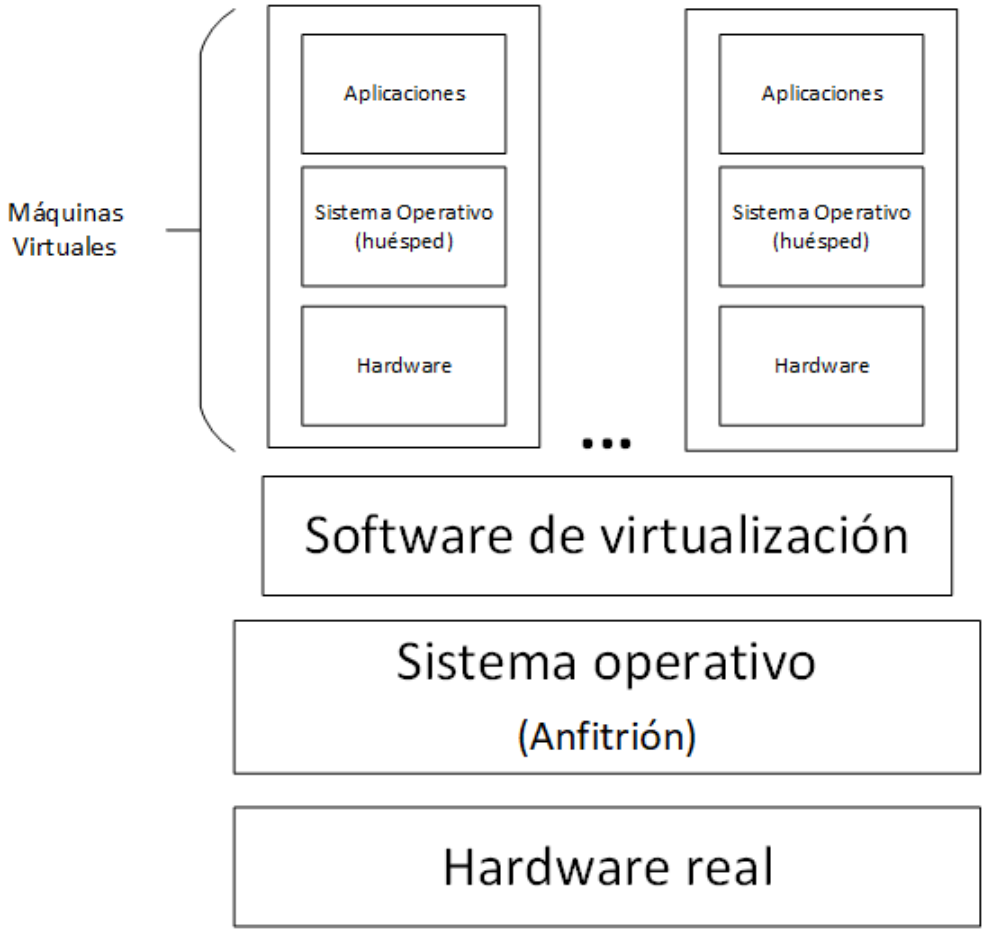


Figura 2.2: *Hypervisor Type 2*. Fuente: Elaboración propia.

aislada de las otras que se ejecutan en el mismo servidor. Ejemplos de *full virtualization* son: Microsoft Virtual Server [5] y VMware ESXi [6]. La figura 2.3 representa este tipo de virtualización.

Para-virtualization

En este tipo de virtualización el sistema operativo que se instala en las máquinas virtuales sabe que se encuentra virtualizado, por lo tanto requiere de *drivers* para poder funcionar. La función de los *drivers* es hacer comandos con el hardware, en vez de hacerlos directamente con el hardware del servidor (el *host*), lo hace al sistema operativo de éste. Ejemplos de este tipo de virtualización son: Xen [7], UML [8], VMware [9]. El esquema 2.4 representea este tipo de virtualización.

[5] https://www.microsoft.com/windowsserversystem/virtualserver/
[6] http://www.vmware.com/products/vsphere-hypervisor.html
[7] https://wiki.xen.org/wiki/Paravirtualization_(PV)
[8] http://virt.kernelnewbies.org/UML
[9] https://goo.gl/8NJTSw

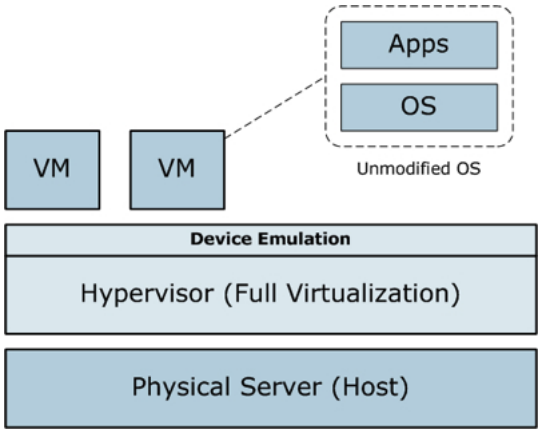


Figura 2.3: Esquema de *full virtualization*. Fuente:
http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm

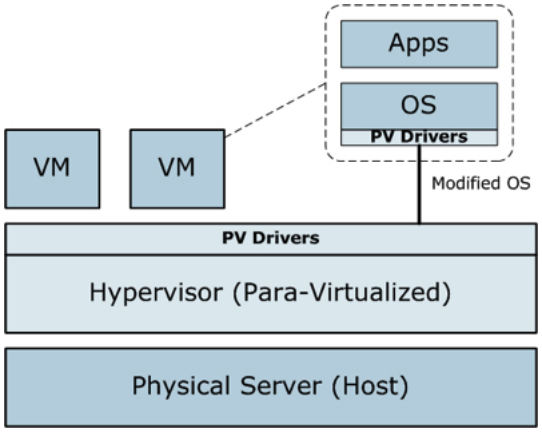


Figura 2.4: Esquema de *para-virtualization*. Fuente:
http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm

Operating System-level virtualization

En este tipo de virtualización no es necesario contar con un *hypervisor*. Es el kernel del sistema operativo del *host* el que se encarga de generar espacios de usuarios aislados (*isolated user spaces* en inglés). El *overhead* en este tipo de virtualización es baja, gracias a que los sistemas *guest* se ejecutan con un kernel compartido. Ejemplos de este tipo de virtualización son: Linux Containers [10], BSD Jails [11], Docker [12]. La figura 2.5 muestra la arquitectura de este tipo de virtualización.

[10] https://linuxcontainers.org/
[11] https://www.freebsd.org/doc/handbook/jails.html
[12] https://www.docker.com/

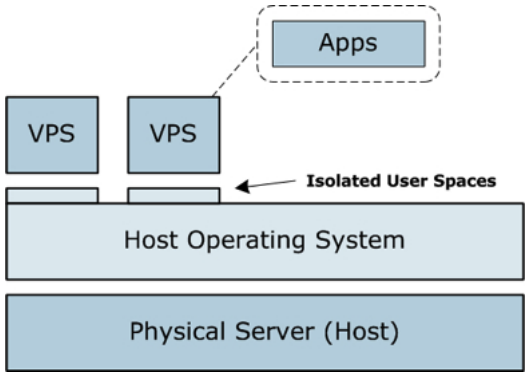


Figura 2.5: Esquema de *os-virtualization*. Fuente:
http://www.datamation.com/netsys/article.php/3884091/Virtualization.htm

Ventajas de la virtualización

La gran ventaja de la virtualización es utilizar un equipo físico para contener una o muchas máquinas virtuales. Antes de la introducción de virtualización, por cada servidor o equipo computacional se tenía un sistema operativo, en el cual el hardware con el software iban estrechamente relacionados. Es decir, por cada aplicación había un servidor que la contenía. Además existía la posibilidad de que en caso de tener más de una aplicación ejecutándose en la misma máquina, se podría generar errores.

Posterior a la virtualización los problemas mencionados anteriormente se superan, ya que al contar con la posibilidad de tener múltiples máquinas virtuales en un servidor, se puede aislar cada aplicación en una máquina virtual, teniendo menos posibilidades de error. Otra ventaja es la reducción de espacio físico y energía para los servidores, ya que un solo servidor puede albergar varias máquinas virtuales.

La figura 2.6 esquematiza lo mencionado anteriormente.

Menascé [6] menciona las siguientes ventajas de la virtualización:

- Seguridad: Como se tienen ambientes aislados, por cada máquina virtual se puede escoger el sistema operativo y las aplicaciones a instalar. Debido a su aislamiento, un ataque en la seguridad de una de estas máquinas no va a comprometer la de las otras.

- Confiabilidad y disponibilidad: Una falla de software en una máquina virtual no afectará a las demás.

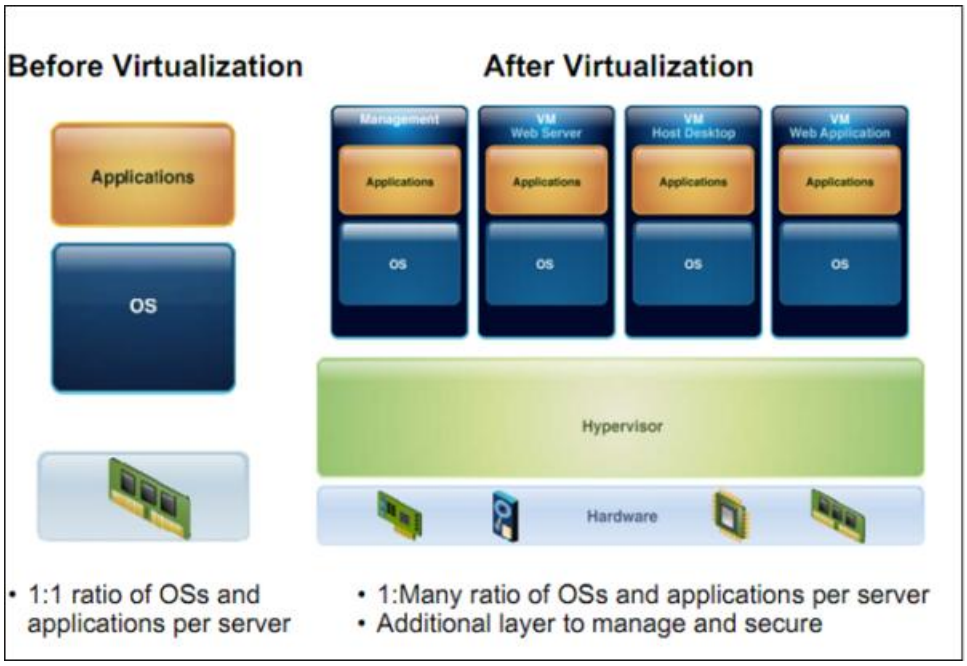


Figura 2.6: Antes y después de la virtualización. Fuente: IBM

- Costos: Es posible reducir costos en términos de personal, espacio físico para los servidores y licencias de software.
- Adaptabilidad a las variaciones de la carga de trabajo: Los recursos computacionales pueden ser asignados a las máquinas virtuales de acuerdo a la carga de trabajo que realicen.
- Balance de carga: Gracias a que las máquinas virtuales son manejadas por el *Hypervisor*, resulta fácil migrar máquinas a otras plataformas para manejar el balance de carga.
- Aplicaciones *Legacy*: Si la compañía desea migrar su infraestructura a otro sistema operativo, es posible mantener una aplicación que funciona en el antiguo sistema operativo ejecutándose en máquinas virtuales, ahorrando así costos de migración.

2.2. Docker

Docker[13] es una herramienta para la *Operating System-level virtualization* [4]. Se engloba dentro del término llamado *containers*, del que se pueden encontrar aproximaciones tempranas con el

[13] https://www.docker.com/

comando *chroot* en Linux. *Linux containers* es la tecnología que está por atrás de Docker. Esta fue lanzada como primera versión en agosto de 2008.

Cada *container* comparte un kernel común que es el de la máquina host, eliminando así la necesidad de tener un sistema operativo para cada uno. Otra característica es la velocidad de inicio de una aplicación en Docker, la cual es más rápida respecto a máquinas virtuales dado el ahorro de la capa de *hypervisor* y a la definición de su arquitectura.

La arquitectura que presenta Docker se muestra en la figura 2.7. En ella se puede observar como elemento de más bajo nivel, la infraestructura, que clásicamente se refiere a un servidor físico, aunque fácilmente puede ser una máquina virtual o una infraestructura en la nube. Lo anterior se debe a la flexibilidad de Docker: tener el poder de ejecutarse tanto en máquinas físicas como virtuales. Otro elemento es el sistema operativo el cual puede ser Microsoft Windows o Linux. Sobre el sistema operativo se encuentra el *Docker Engine*, el cual es un entorno de ejecución de *containers* ligero y robusto, que se encarga de crearlos y ejecutarlos. Es una herramienta poderosa, ya que permite empaquetar todo el código de la aplicación que se desee, junto con todas sus dependencias en un *container* aislado, que permite distribuirlo a otros *host* que tengan previamente instalado Docker. Finalmente sobre la capa de *Docker Engine* se encuentran ejecutándose los *Containers*, en la imagen se puede ver la estructura que estos manejan: poseen en su interior la aplicación que se desee, junto a sus bibliotecas y binarios requeridos. Se puede notar la ausencia de un sistema operativo, ya que se comparte el del *host* a través de la comunicación con el *Docker Engine*. Resulta bastante ventajoso ejecutar varios contenedores en un *host*, ya que permite un mayor aprovechamiento de los recursos de hardware. Una comparativa interesante entre máquinas virtuales y *containers* se muestra en secciones siguientes.

2.2.1. Componentes de Docker

A continuación se explican los principales componentes de Docker que se encuentran en su infraestructura.

- *Docker images*: Una *Docker Image* es un *template* de solo lectura con instrucciones para crear *containers*. Por ejemplo, se puede tener una imagen con la instalación del sistema operativo Centos junto con Apache como servidor web. En esta misma imagen se puede incluir una

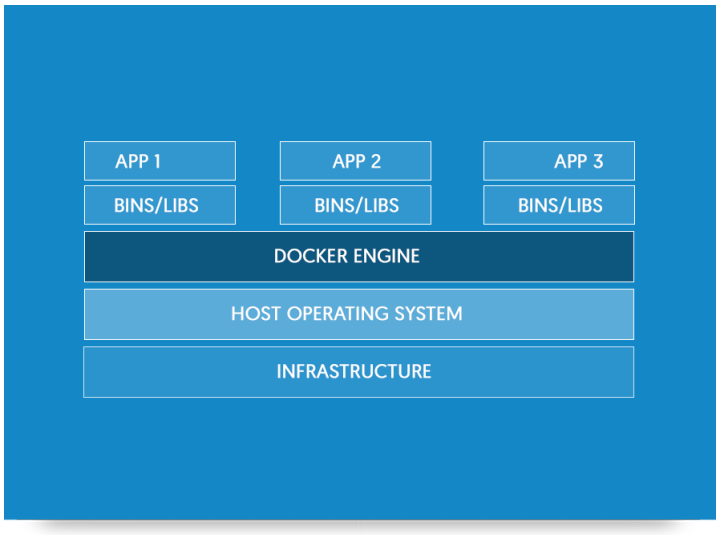


Figura 2.7: Infraestructura de Docker. Fuente: http://www.docker.com

aplicación web; y todo lo anterior vendría siendo una imagen lista de la aplicación web que se desee. Una imagen se arma mediante un archivo de texto llamado *Dockerfile*, luego se ejecuta el comando *docker build* para construirla.

- *Docker containers*: Con la definición de *Docker images* en mente, se plantea el concepto de *Docker containers* que es una instancia ejecutable de una *Docker image*. Esta instancia se puede ejecutar, detener, iniciar o borrar gracias a comandos Docker. Un gran punto a notar es que cada *container* es un entorno aislado y seguro, pero si se desea se puede dar acceso a recursos que se encuentran en otro *host* o en otro *container*, tales como bases de datos o almacenamiento persistente.

- *Docker registries*: Corresponde a una biblioteca de imágenes. Los *registries* pueden ser públicos o privados. Un buen ejemplo de *docker registry* público es: https://hub.docker.com/ el cual cuenta con imágenes tanto en repositorios oficiales de tecnologías como lo es *nginx*[14] o bien repositorios de la comunidad que los comparte a través de esta plataforma.

- *Docker services*: Este componente permite que un *swarm* (o enjambre) de nodos (*hosts*) de Docker trabajen en conjunto, ejecutando un número definido de instancias a partir de una tarea, que corresponde a una *Docker image*. Se puede especificar la cantidad de réplicas concurrentes

[14] https://www.nginx.com/

para una tarea, siendo *Docker service* el que se encargue de ello, y así ser transparente para el usuario.

El objetivo de Docker es ofrecer una herramienta para poder ejecutar y distribuir aplicaciones de software. Esto lo hace mediante un sistema de archivos que contenga todo lo necesario para ello: código, entorno de ejecución, herramientas de sistema, biblioteca de sistema; y posteriormente ejecutar esto en un servidor que tenga instalado el binario de Docker.

De la página web de Docker [15] se mencionan 3 características principales de su arquitectura:

- Liviano: Los containers que se están ejecutando en una misma máquina comparten el mismo kernel del sistema operativo; lo anterior hace que inicien de manera casi instantánea y ocupen menos RAM. Las imágenes son construidas a partir de un sistema de archivos por capas y comparten archivos en común, logrando así, un uso de disco y descarga de imágenes mucho más eficiente.
- Abierto: *Docker containers* están basados en estándares abiertos, permitiendo que los *containers* se puedan ejecutar en la gran mayoría de distribuciones Linux y Microsoft Windows.
- Seguros por defecto: Cada *container* aísla las aplicaciones que contiene de los otros y también de la infraestructura que se encuentra por debajo (como por ejemplo un servidor).

Tecnologías de Docker

Tras Docker hay unas cuantas tecnologías que juegan un rol fundamental en su funcionamiento, las que enuncian a continuación [16]:

- *Namespaces*: Permite proveer un espacio de trabajo aislado para el *container*. Cuando se ejecuta un *container* se crea una serie de *namespaces* para éste, obteniendo así una capa de aislamiento. Un punto ventajoso de lo anterior es que cada aspecto del *container* se ejecuta en un *namespace* separado y su acceso es limitado a ese *namespace*. Algunos *namespace* que usa Docker en Linux son:
  - pid: *Process ID*, para el aislamiento de procesos.

[15] http://www.docker.com
[16] https://docs.docker.com/engine/docker-overview/#the-underlying-technology

  - net: *Networking*, para la administración de interfaces de red.
  - ipc: *InterProcess Communication*, se encarga de administración del acceso a los recursos *ipc*.
  - mnt: *Mount*, se encarga de administrar los puntos de montaje del sistema.
  - uts: *Unix Timesharing System*, necesario para el aislamiento de los identificadores *hostname* y *NIS domain name*.

- *Control groups*: Abreviado como *cgroups* se encarga de limitar los recursos del *host* a una aplicación en especifico. Docker ocupa *cgroups* para gestionar los recursos de hardware para los contenedores y opcionalmente establecer límites para ellos. Un ejemplo sería limitar la cantidad de memoria que puede utilizar un contenedor.

- *Union file systems*: *UnionFS* es un sistema de archivos que opera creando capas, logrando un sistema bastante rápido y liviano. Docker usa *UnionFS* para proveer un aislamiento por capas a cada contenedor y así evitar el duplicado de archivos que son comunes a un grupo de contenedores.

- *Container format*: La conjunción de los elementos anteriores nombrados se llama *container format*. El formato por defecto es *libcontainer*, aunque en el futuro Docker espera soportar otras tecnologías como *BSD Jails* o *Solaris Zones*.

2.2.2. Máquinas virtuales v/s Contenedores

Las máquinas virtuales gozan de bastante popularidad, son usadas en muchos sistemas *clouds* en la actualidad, tales como *Amazon EC2* o *Microsoft Azure*. Para los usuarios finales también existen herramientas para generar máquinas virtuales en sus equipos personales: *VirtualBox*[17] o *VMware*[18] por nombrar algunas. Ambas opciones proveen una manera fácil de crear y gestionar las máquinas virtuales. Un par de puntos a considerar en las máquinas virtuales es la existencia del *Hypervisor* y que de éste dependerá el rendimiento de la aplicación que se tenga en máquinas virtuales.

[17] https://www.virtualbox.org/
[18] https://www.vmware.com/latam/products/workstation-player/workstation-player-evaluation.html

En su contraparte, para el funcionamiento de Docker no es necesario una capa de *Hypervisor*, con lo que de forma teórica se obtiene un mejor rendimiento. Preeth [9] presenta un cuadro de comparación entre máquinas virtuales y Docker, que se resume en la tabla 2.1.

Tabla 2.1: Máquinas virtuales v/s Docker. Fuente: Elaboración propia.

Máquinas Virtuales	**Docker**
Se ejecutan en un hardware virtual y el sistema operativo *guest* debe ser cargado en su propia memoria.	Los contenedores Docker utilizan el kernel del sistema operativo del Host, evitando que cada uno cuente con un sistema operativo propio.
La comunicación entre *guests* es a través de dispositivos de red, aunque también por software.	La comunicacion entre *guests* es a través de *pipes, sockets, bridges, etc.*
La seguridad depende del *Hypervisor*.	Por defecto son seguros gracias a *kernel namespaces*.
Más *overhead* dado su complejidad.	Menor *overhead*, ya que son ligeros.
La compartición de bibliotecas y archivos no es posible por defecto.	Compartición de archivos es posible, por ejemplo utilizando el comando *scp* en Linux.
Toma minutos en iniciar (*booting*).	Inicia en segundos.
Usa más memoria ya que tiene que almacenar el sistema operativo completo por cada *guest*.	Menor uso de memoria ya que comparte el mismo sistema operativo.

En la figura 2.8 se puede observar la infraestructura tanto de las máquinas virtuales como de Docker. Queda clara la capa de *hypervisor* y sobre esta capa las máquinas virtuales, las que poseen un sistema operativo cada una. En su contraparte, Docker (abajo en la figura) presenta una capa *Docker engine* y por encima cada contenedor, con la diferencia que no se observa el sistema operativo en ellos, ya que utiliza el del *host*, logrando así ser más livianos.

2.3. HPC con contenedores

High performance computing[19] es un campo de la informática que abarca simulaciones computacionales de problemas complejos. Las simulaciones que se realizan en el proyecto SIPAT responden a esta área.

[19] https://insidehpc.com/hpc-basic-training/what-is-hpc/

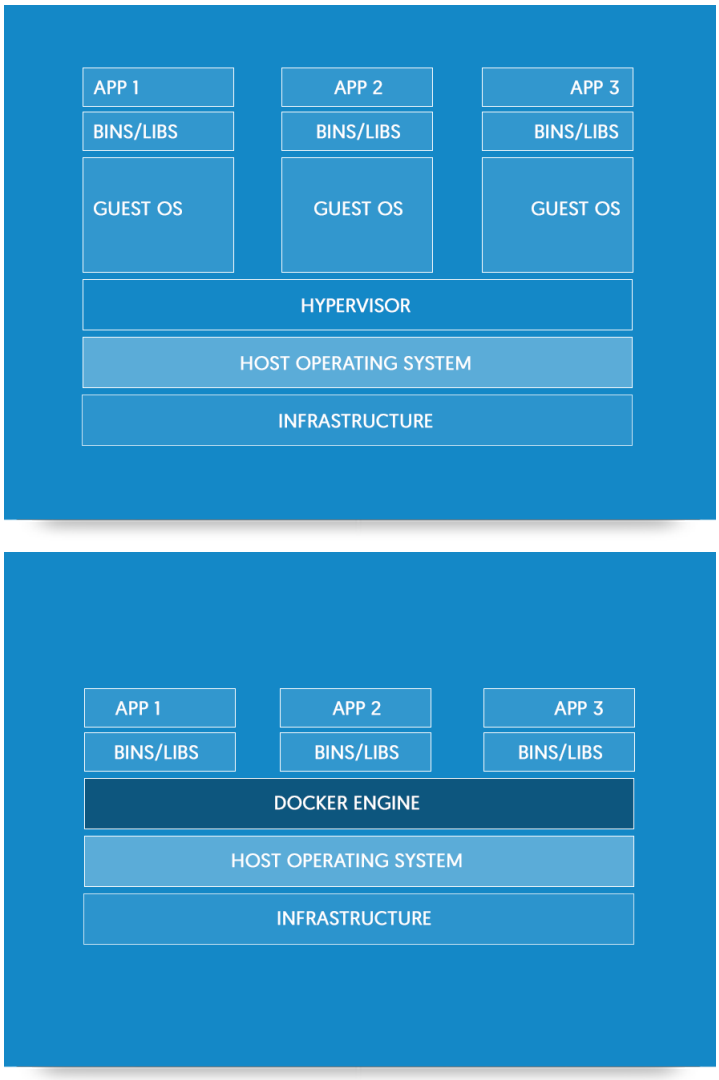


Figura 2.8: Arriba: arquitectura de máquinas virtuales. Abajo: arquitectura de docker. Fuente: http://www.docker.com

En la actualidad en SIPAT las simulaciones computacionales se realizan con el software COMCOT en una arquitectura de máquinas virtuales administrada por *oVirt*[20]. Adufu [2] realizó un trabajo de investigación para lograr responder la siguiente interrogante: ¿la tecnología de *containers* es la indicada para aplicaciones de computación de alto rendimiento?. Usando *Docker* demuestra que durante la ejecución de aplicaciones científicas, en general los tiempos de ejecución para la tecnología basada en *containers* son menores que la tecnología basada en *hypervisores*, dado las diferencias en los procesos de inicio (*booting*). A su vez, al ejecutar *autodock3*,[21] que es un software de simulación

[20] https://www.ovirt.org/
[21] http://autodock.scripps.edu

para el modelado molecular, en entornos de hardware similares y en *containers*, se obtiene como resultado que *Docker* gestiona los recursos de memoria más eficientemente, incluso cuando se le asigna más memoria de la que tiene disponible el *host*. Finalmente, gracias a lo anterior concluye que la tecnología basada en *containers* es más adecuada para aplicaciones de *High performance computing*.

Con lo anterior en mente se tiene una base de que la tecnología de *containers* es adecuada para HPC y puede ser aplicada en el proyecto SIPAT. Resta en este trabajo realizar las pruebas con la herramienta de modelado COMCOT a fin de estudiar su comportamiento en ambiente *Docker* y posteriormente configurar los parámetros de ejecución que permitan obtener el máximo rendimiento posible.

3 | Diseño de Solución

En este capítulo se presenta el diseño de la solución propuesta. Se realizará un análisis tanto de hardware como de software a fin de evaluar los elementos que se tiene a disposición y las características de éstos.

3.1. Hardware base para el proyecto

Es sumamente relevante tener en mente el *hardware* a disposición para la construcción de la solución, ya que corresponde a la base de trabajo. En SHOA existe un datacenter que cuenta con equipamiento adquirido en el Proyecto FONDEF D11I1119 denominado "Diseño e Implementación de una Base de Datos de Predicción de Tsunamis para la Costa Chilena utilizando Modelación Computacional de Alto Rendimiento", cuyo proyecto es anterior al presente. En el año 2016 se comenzó una remodelación del datacenter existente y junto a ella, la migración de parte del equipamiento tecnológico del anterior proyecto FONDEF a un nuevo datacenter existente en la sala SNAM, que viene a agregar características propias de la arquitectura física de los datacenter como: mejoras en ventilación, anclaje más seguro al piso, control de acceso y monitoreo, entre otras.

Se describe el equipamiento en las siguientes subsecciones.

3.1.1. Servidores

Los servidores disponibles para el desarrollo de esta memoria son cinco, las que se describen en la Tabla 3.1:

Tabla 3.1: Listado de servidores alojados en SNAM. Fuente: Elaboración propia.

Nombre	Modelo	CPU	RAM	Disco duro
Sipat 1	Dell PowerEdge R720	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32 Cores	32 GB	100 GB 1 TB
Sipat 8	Dell PowerEdge C6220	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32 Cores	32 GB	100 GB
Sipat 9	Dell PowerEdge C6220	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32 Cores	32 GB	100 GB
Sipat 10	Dell PowerEdge C6220	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32 Cores	32 GB	100 GB
Sipat 11	Dell PowerEdge C6220	Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz 32 Cores	32 GB	100 GB

Dichos servidores se mantendrán intactos físicamente, pues el objetivo es hacer un uso eficiente de los instrumentos a disposición. En vista de lo anterior se utilizarán todos los servidores para ejecutar exclusivamente “Docker”, buscando que no se tenga programas adicionales instalados que no cumplan una función afín al modelamiento.

Existirá un orquestador del cual dependerán los servidores, cuyas principales funciones será distribuir las tareas de modelamiento y carga de trabajo. Este orquestador corresponderá a una máquina virtual creada con oVirt la que tendrá instalado Docker y Rancher. La elección de Rancher se explica con más detención en la sección 3.5.

3.1.2. Almacenamiento

Dentro del apartado Almacenamiento, se dispone un equipo con 5 TB de capacidad. Este se encuentra particionado en volúmenes de 1 TB cada uno, los cuales son accesibles mediante protocolo “iSCSI”[22] por parte de lo servidores.

En la tabla 3.2 se observa en detalle las características del Almacenamiento.

[22] https://tools.ietf.org/html/rfc3720

Tabla 3.2: Características de Storage. Fuente: Elaboración propia.

Característica	**Detalle**
Modelo	Dell EqualLogic PS6110
Cantidad de discos	24
Raid	10
Discos Duros	SAS 500 GB @ 15000 RPM
Conexión	Fibra óptica y ethernet

Lo que se persigue es utilizar de manera óptima el almacenamiento y obtener mejor rendimiento. En el sección 3.3, se estudiarán las posibilidades que se adecúan al entorno de trabajo y las que finalmente se materializarán.

3.1.3. Redes

Todos los servidores que se utilizarán en esta memoria se encuentran en el datacenter del Departamento de Tecnologías de la Información de SHOA, los cuales se unen con fibra óptica al servidor de almacenamiento presente en el datacenter de SNAM.

En cada servidor se encuentran conectados dos enlaces de fibra óptica. Para un mejor desempeño, se configurará un *"bonding"* (ver figura 3.1) de modo 4 (802.3ad), ofreciendo así una mayor disponibilidad y aumento en la velocidad. La decisión de configurar este tipo de solución viene de la mano del proyecto FONDEF anterior, en el que esta configuración resultó ser exitosa, aumentando así las tasas de transferencia entre los servidores.

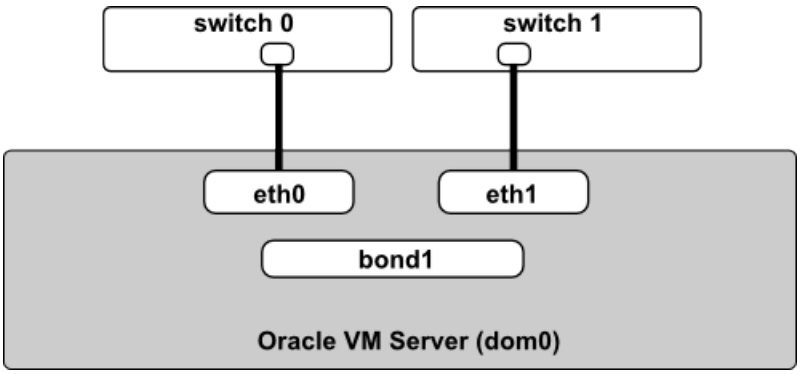


Figura 3.1: Esquema ejemplificador de conexión bonding, en este caso en una máquina virtual. Fuente: oracle.com

3.2. Docker

Docker es una tecnología relativamente nueva que data del año 2013 [23], pero que ha ganado peso en estos años. Hace uso de la virtualización a nivel de sistema operativo y se presenta como una plataforma que permite construir, ejecutar y distribuir aplicaciones de manera rápida y sencilla. Hace uso de tecnologías presentes en el Kernel Linux para su funcionamiento como lo son: *namespaces*, *control groups*, *union file systems*, *container format*.

3.2.1. Conceptos Generales

Como se mostró en el capítulo anterior, existen *papers* [2] [4] [9] en la literatura que dan una aproximación temprana sobre el uso de tecnología basada en *containers* para tareas que requieren gran poder de procesamiento. Esto es gracias a la arquitectura de Docker, ya que presenta una menor sobrecarga que el esquema clásico de virtualización con máquinas virtuales. Un punto de comparación es que Docker permite tener ejecutando seis a ocho veces más *containers* en el mismo hardware, respecto a máquinas virtuales [13].

El uso de Docker junto a sus componentes se realizará de la siguiente manera:

- *Docker engine*: Más conocido simplemente como Docker, se utilizará para crear, ejecutar y administrar *containers*.
- *Docker image*: Un *container* es una instancia en ejecución de una *Docker image*. Luego ésta *image* (*imagen* a partir de ahora) es necesaria crearla considerando el software base necesario para la modelación computacional, vale decir: sistema operativo base, COMCOT [24] (Ver sección 3.2.2), herramientas para compresión de archivos, entre otros.
- *Docker volume*: Es la forma en que se provisiona almacenamiento a un *container*. Cada modelación tendrá un *volume* (*volumen* a partir de ahora) dedicado y luego de terminada se procederá a su eliminación para no hacer uso innecesario del espacio de almacenamiento.
- *Docker registry*: Como se creará una *imagen* a medida para la modelación, es necesario que se almacene en un repositorio que sea capaz de interactuar con *docker engine*. Para lo anterior

[23] https://www.docker.com/company
[24] http://223.4.213.26/archive/tsunami/cornell/comcot_down.htm

se hará uso de *docker registry*, un repositorio de *imágenes* que puede ser implementado de manera local y privada.

- *Docker mirror*: Junto al punto anterior, resulta óptimo la instalación de un caché local de *imagenes de docker*, evitando que cada vez que se ejecute un container se descargue de internet y haga uso de ancho de banda de forma excesiva. Docker provee una configuración llamada *registry mirror*, la cual almacena de manera local cualquier imagen que se descargue de internet, luego los *container* posteriores que se ejecuten a partir de esa *imagen* harán uso del almacenamiento local en vez de descargarla de internet.

3.2.2. COMCOT para modelamiento

COMCOT (Cornell Multi-grid Coupled Tsunami model)[25] es un software escrito en *Fortran*, el cual es capaz de modelar el ciclo de vida de un tsunami, incluyendo su generación, propagación e inundación en regiones costeras. En el presente proyecto COMCOT es el software de modelamiento para los diferentes puntos de pronósticos presentes a lo largo del país. Requiere de un archivo de configuración y archivos de entrada para el proceso de modelación y como resultado se obtienen entre muchos archivos, las series de tiempo que posteriormente se almacenan en una base de datos.

A continuación se presentan los principales archivos de entrada y salida de COMCOT [8]:

3.2.2.1. Archivos de entrada

Para realizar las simulaciones se necesita de un conjunto de archivos de entrada, dentro de los que se encuentran:

- Puntos de pronóstico (mareógrafo virtuales): Archivo en texto plano, que contiene dos columnas que indican latitud y longitud de los puntos de pronóstico a usar. La cantidad de filas determina el número de mareógrafos virtuales a utilizar. En general, este archivo se denomina `ts_location.dat`.

- Batimetría: Topología del fondo marino contenida en un archivo cuyo formato puede ser binario o en texto plano (ASCII), en el que indica las coordenadas geográficas de sectores

[25] http://223.4.213.26/archive/tsunami/cornell/comcot_down.htm

en particular, además de indicar la profundidad, formando una malla que representa el suelo submarino.

- Deformación del fondo marino: Deformación calculada a partir de las ecuaciones de Okada [3], mediante la cual es posible calcular el traspaso de la energía liberada a la columna de agua que genera el tsunami.

- `Comcot.ctl`: Archivo de configuración de COMCOT. Este archivo se denomina `Comcot.ctl` y está compuesto por cuatro secciones de parámetros que controlan la simulación:

  - Parámetros generales: Incluye el tiempo total de simulación, los intervalos de tiempo en que se escriben las salidas parciales a los archivos y los datos de las condiciones de borde entre otros.

  - Parámetros para modelos de falla: Diversos parámetros como ángulos de dislocación, largo de la falla o latitud y longitud del epicentro que se usan para calcular la deformación del fondo marino producto de un sismo, son indicados en esta sección. COMCOT asume que esta deformación se transmite íntegramente a la superficie del mar por la columna de agua.

  - Parámetros para el generador de olas: Sección encargada del deslizamiento submarino o de la olas solitarias.

  - Parámetros para grillas: Sección con los parámetros de la grilla o región en la que sucede la simulación, como las coordenadas. COMCOT permite el uso de grillas anidadas, es decir, el uso de múltiples mallas del fondo marino para aumentar el detalle de la batímetría.

- Comcot: Archivo binario ejecutable, creado a partir de la Compilación del código fuente en Fortran con *ifort*[26](también puede ser compilado con gfortran[27]), el cual procesa los archivos de entrada, calculando las *Shallow Wave Equations*[14] para entregar los archivos de salida.

[26] https://software.intel.com/en-us/fortran-compilers
[27] https://gcc.gnu.org/wiki/GFortran

3.2.2.2. Archivos de salida

Dentro de la información que arrojan los archivos de salida, SIPAT utiliza y almacena en sus bases de datos las series de tiempo así como su forma resumida, los que se describen a continuación:

- Series de tiempo: Archivo que entrega para cada segundo de simulación la altura de ola en la coordenada respectiva a cada punto de pronóstico. El formato es de una columna, donde cada fila representa la amplitud del tsunami en un tiempo dado con respecto al 'nivel medio' del mar.

- Elevación de superficie libre: Los archivos `z_#LayerID_#TimeStep.dat` indican la elevación de la superficie libre en el dominio de la grilla que forma la batimetría, reflejando el movimiento de expansión de los trenes de olas en el tiempo, dentro de la sección geográfica delimitada.

3.2.3. *Imagen* para modelamiento

Es necesario crear una *imagen* para la ejecución de los *containers*, esta se creará a partir del sistema operativo Fedora 25, [28] en la cual se incluirán herramientas necesarias para la puesta en marcha de todo el proceso de modelamiento, similar al de las máquinas virtuales. Un punto importante a considerar al armar la *imagen* es que el software incluido tiene que ser el justo y necesario, para no generar una imagen de gran tamaño, ni mucho menos tener ejecutándose procesos que no son parte de la modelación.

Con la *imagen* para el modelamiento dispuesta y lista para su funcionamiento, se propone ejecutar un *container* por cada escenario, con lo que se tendría una instancia de COMCOT en cada uno. Para el manejo de los archivos generados en el proceso, es necesario presentar el almacenamiento aparte. Este tema se discute con más profundidad en la sección siguiente.

Las instrucciones a ejecutar para seguir los pasos necesarios de modelación son gracias a un *script* en lenguaje de programación *Python* que se adaptará desde el modelamiento en máquinas virtuales hacia Docker. En lo posible se harán los menores cambios, lo anterior permitirá si en un futuro se modifica el *script* correspondiente a las máquinas virtuales, no se tenga que realizar grandes cambios en el de Docker.

[28] https://getfedora.org

Finalmente, una vez que termine el proceso de modelación es necesario borrar el *container* y los archivos de residuo que quedarán en el almacenamiento facilitado.

3.3. Almacenamiento para Docker

El almacenamiento juega un rol fundamental para los archivos que se generarán producto de la modelación. Contar con un *storage* de 5 TB dividido en volúmenes de 1 TB hace necesario idear una estrategia *ad hoc* al uso que se le dará y permitir el uso simultáneo de los 5 TB en cada servidor. Si bien se cuenta con espacio de disco en cada servidor, éste se prefiere dejar para el sistema operativo y ocupar el disponible en el *storage* para Docker. Es necesario configurar Docker para que tome conocimiento de cual almacenamiento es el que tiene que utilizar. Esto es algo ya pensado en el diseño de *Docker Engine* así que su configuración resulta sencilla.

A continuación se presentan diferentes alternativas a considerar, luego en el siguiente capítulo se implementará el más adecuado.

3.3.1. Global File System 2 (GFS2)

GFS2 [29] es un sistema de archivos para clúster que tiene como característica principal permitir el acceso coordinado de los nodos de un clúster al mismo dispositivo de bloque. GFS2 ofrece compartición de datos entre los nodos GFS2 del clúster con una vista consistente del sistema de archivos a lo largo de todo el clúster, lo que permite que procesos ejecutándose en diferentes nodos compartan archivos GFS2 como si fueran archivos de un sistema local, Acero [1].

Con lo anterior presente, se propone crear un clúster entre los servidores disponibles y sobre éste instalar un sistema de archivos distribuido, siendo en este caso GFS2.

Se propone el siguiente diagrama 3.2 para la implementación de GFS2.

[29] https://www.kernel.org/doc/Documentation/filesystems/gfs2.txt

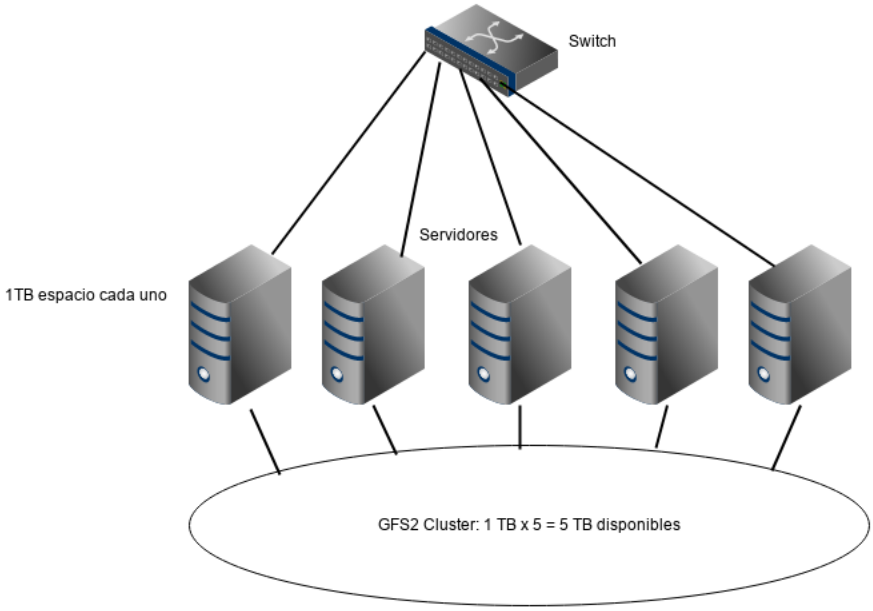


Figura 3.2: Implementación GFS2. Fuente: Elaboración propia.

En este diagrama se observa la disponibilidad de almacenamiento en cada servidor correspondiente a la conexión hacia el *storage* mencionado anterior y unidos entre sí mediante un switch. Sobre estos servidores es necesario armar un clúster y luego configurar CLVM [30] (*Clustered Logical Volume Manager*) . Finalmente con la configuración anterior puesta en marcha se procederá a crear el sistema de archivos GFS2. Como resultado se tendrá disponible la suma de 5 TB, correspondiente a la suma de 1 TB por los cinco servidores de la presente memoria.

Este alternativa se plantea como la primera a implementar, en caso se fallo se trabajará con la siguiente.

3.3.2. GlusterFS

GlusterFS[31] corresponde a un sistema de archivos distribuido cuyo actual desarrollador es la compañía Red Hat. Su arquitectura es de tipo cliente-servidor, en la cual se crea un sistema de bloques de almacenamiento en los servidores y posteriormente los clientes se conectan a éste para hacer uso de ese espacio. Los clientes se conectan a los servidores mediante protocolo TCP/IP, InfiniBand o Sockets, montando el sistema de archivos mediante protocolo nativo vía *FUSE* [32]

[30] https://www.sourceware.org/lvm2/
[31] https://www.gluster.org/
[32] https://github.com/libfuse/libfuse

(*Filesystem in Userspace*).

Algunos conceptos importantes de notar en Gluster son:

- *Cluster:* Sistema de computadores unidos. En el caso de esta memoria corresponderían a los servidores.
- *Trusted storage pool:* Corresponde a un red confiable de servidores de almacenamiento.
- *Brick:* Es la unidad básica de almacenamiento. La idea es que cada servidor exponga un *brick* (*bloque* a partir de ahora) para su uso en gluster.
- *Volume:* Es una colección lógica de *bloque*. Sobre el *volumen* generalmente se realizan todas las operaciones de gluster.
- *Server:* Equipo que forma parte de *Trusted storage pool*. Cada servidor pone a disposición un bloque de almacenamiento.
- *Client:* Equipo que monta el/los *volumen(es)* para su uso.

Existen diferentes tipos de *volumen* que se puede crear:

- *Distributed glusterFS volume:* Es la opción por defecto si no se especifica el tipo de *volumen* cuando se está creando. Los datos a almacenar se distribuyen a través de los *bloques* del *volumen*, es decir, si se cuenta con 2 *bloques* y un archivo a almacenar, este se quedará en uno de los dos *bloques*, pero no en ambos.
- *Replicated glusterFS volume:* En este tipo de *volumen* los datos se encuentran duplicados en los *bloques* de almacenamiento.
- *Distributed replicated glusterFS volume:* Para este esquema es necesario armar *volúmenes* replicados y sobre estos configurar el volumen distribuido.
- *Striped glusterFS volume:* Dado el caso en que se desee acceder a un archivo de gran tamaño y para no sobrecargar en demasía a un *bloque* y disminuir su rendimiento, es posible configurar el volumen para que divida partes de un archivo que se desea almacenar a lo largo de los *bloques*, facilitando así las tareas de acceso a dicho archivo.

- *Distributed striped glusterFS volume:* Idea similar al ítem anterior, con la excepción de que los datos pueden ser distribuidos a través de una cantidad determinada de *bloques* formando *volúmenes* diferentes los que a su vez forman un sistema distribuido.

La estrategia que se desea ocupar con gluster es utilizar cada TB desde el *storage* como *bloque* en gluster. Luego crear un *volumen* y montarlo en cada servidor.

El esquema deseado de ocupar es *distributed glusterFS volume* distribuido, una imagen ejemplificadora de esto es la figura 3.3:

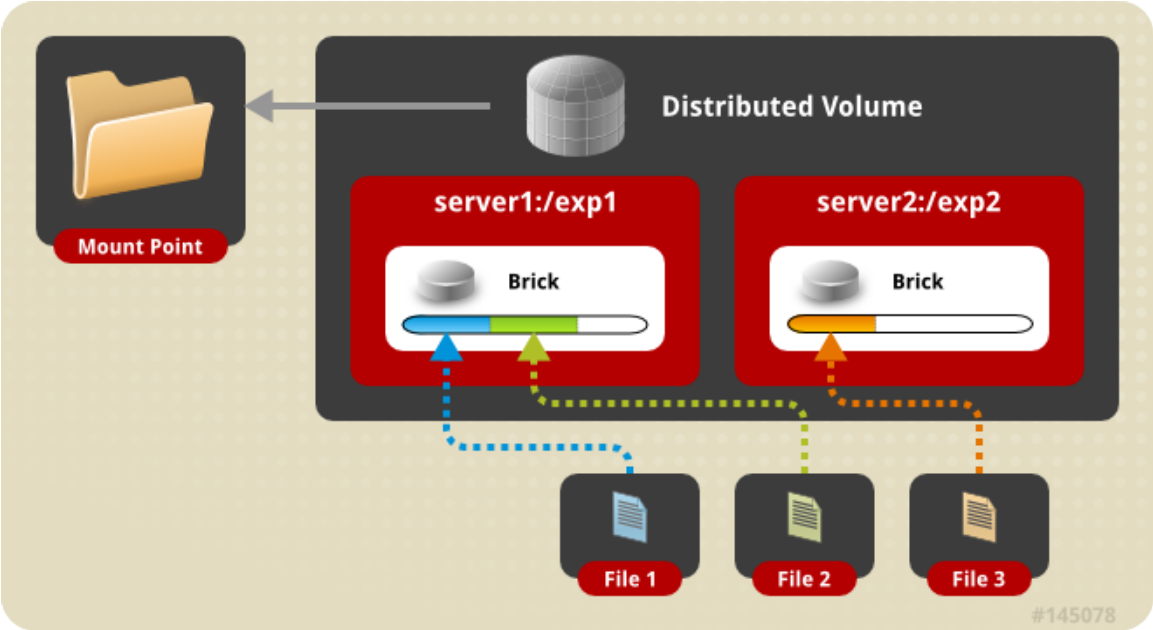


Figura 3.3: Esquema propuesto de volumen distribuido. Fuente: gluster.org

El resultado de la configuración anterior dará lugar a que el espacio disponible para su uso sea de 5TB, correspondiendo a la suma de cada 1TB de los servidores facilitados.

Esta alternativa se plantea como la segunda a utilizar luego de GFS2.

3.3.3. Almacenamiento individual

Otra opción a evaluar es contar con almacenamiento separado en cada servidor, quedando de la forma 1 TB de espacio disponible. Esta opción tiene su ventaja respecto a las anteriores en la facilidad de configuración, ya que de por medio no hay que implementar un sistema de archivos distribuidos para lograr un acceso simultáneo a los 5 TB. Una clara desventaja es la capacidad de

espacio disponible, que se reduce respecto al total disponible en el *storage*, lo cual complica el caso de uso en el cual un servidor requiere de más espacio de almacenamiento respecto a sus pares.

El siguiente diagrama muestra la opción de configuración.

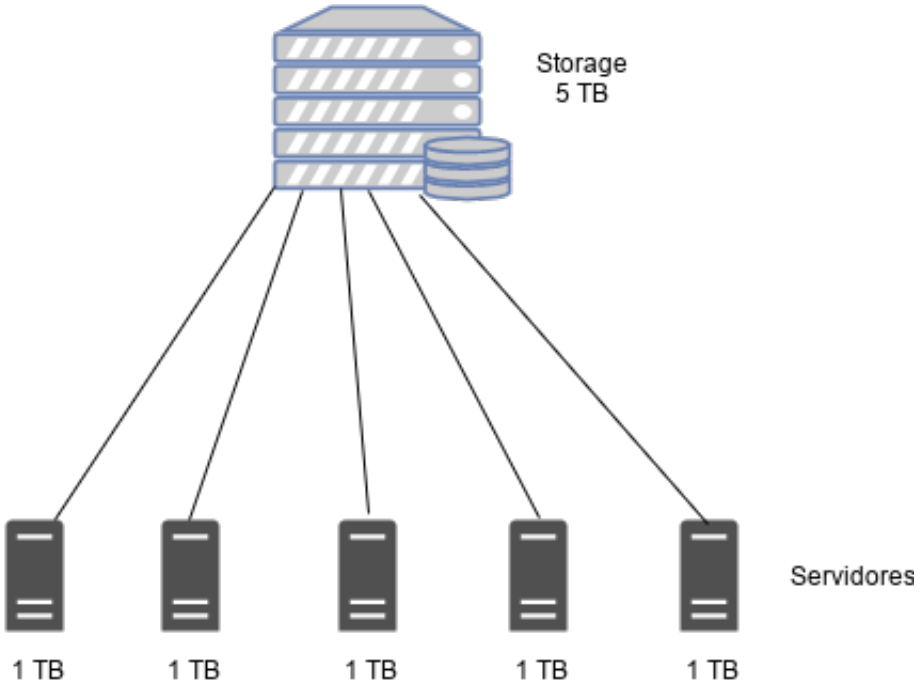


Figura 3.4: Configuración de almacenamiento individual para servidores. Fuente: Elaboración propia.

Como última alternativa de trabajo se utilizará el almacenamiento individual. En caso de problemas con la implementación con GFS2 y GlusterFS, se configurará 1 TB de almacenamiento en cada servidor.

3.4. Interfaces de administración de containers Docker

En esta sección se muestran las interfaces de administración para Docker. Una interfaz gráfica de administración viene a simplificar el uso y configuración, ya que por defecto el trabajo con containers se realiza por interfaz de comandos.

Si bien pueden existir una gran gama de interfaces gráficas, se eligen las más populares y con más trayectoria, a fin de obtener productos maduros y con trayectoria de uso por parte de la comunidad.

- *Simple Docker UI* [33] (ver figura 3.5): Nació como una extensión para Google Chrome para

[33] https://github.com/felixgborrego/simple-docker-ui

posteriormente ofrecer paquetes para Windows, Linux y OS X. Las características que tiene son: remoción de containers sin uso e imágenes; terminales virtuales para entrar a un container; búsqueda y descarga de imágenes de *Docker Hub*; interacción con containers (inicio, detención, borrado). Un punto en contra es que se encuentra en una versión beta.

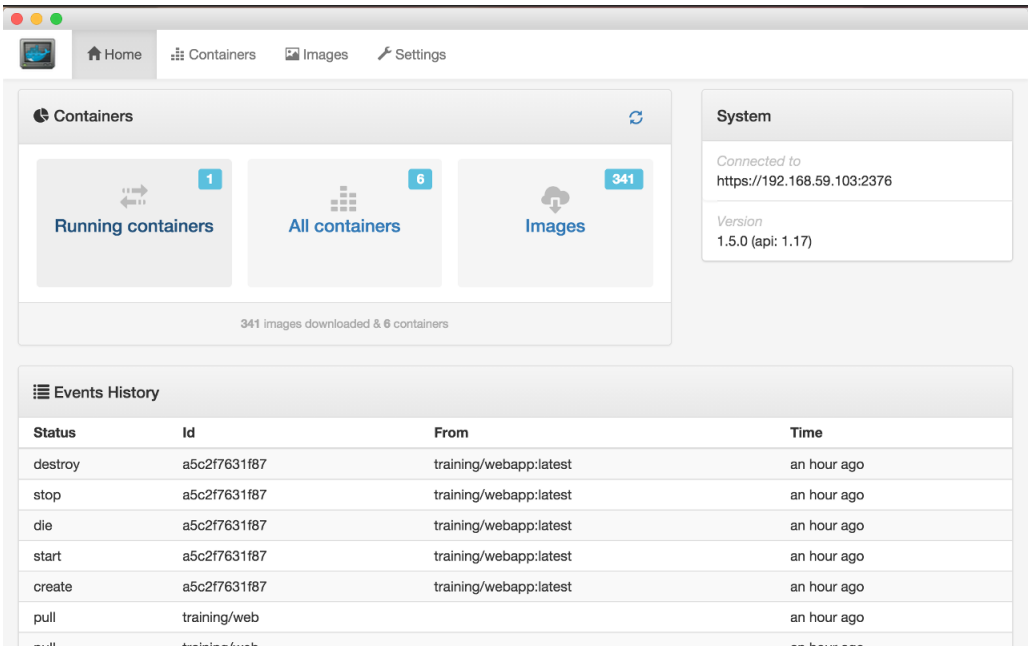


Figura 3.5: Interfaz de *Simple Docker UI. Fuente:* https://github.com/felixgborrego/simple-docker-ui

- *portainer.io* [34] (ver figura 3.6): Se ofrece como una alternativa liviana y de código abierto para gestionar *host* Docker o cluster *swarm*. Se encuentra disponible para Linux, Windows y OS X. Su instalación se realiza mediante un contenedor Docker, para luego acceder a él mediante una interfaz web desde cualquier navegador. Entre sus características se cuenta: creación, estadísticas, logs, consola de containers; vista de redes, volúmenes de almacenamiento, *templates* de containers.

[34] http://portainer.io

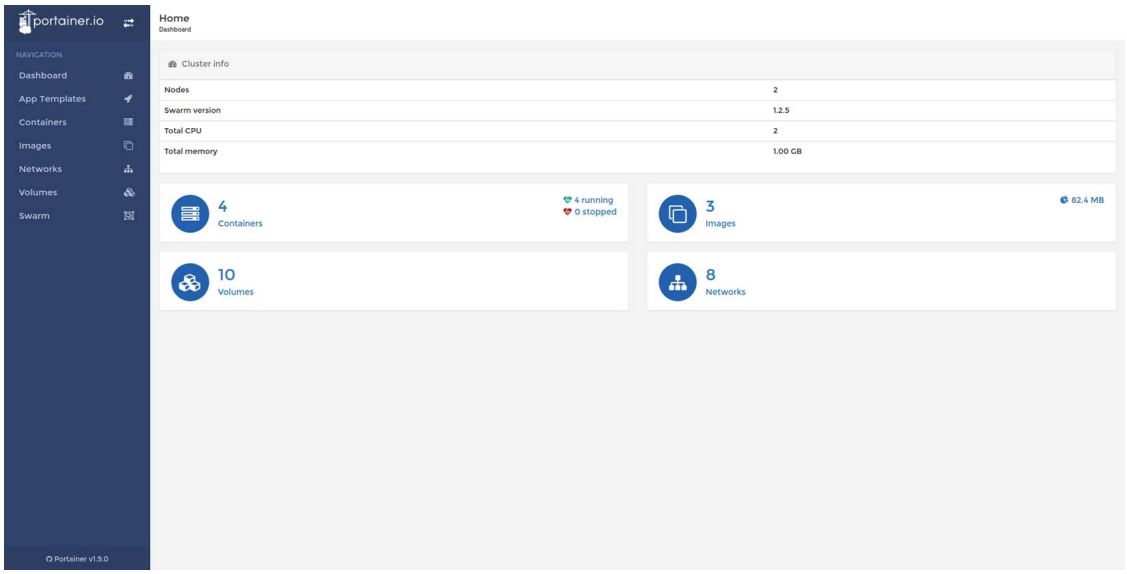


Figura 3.6: Interfaz gráfica de Portainer.io. Fuente: http://portainer.io

- *Shipyard* [35] (ver figura 3.7): Otra alternativa web, que mediante el despliegue de un contenedor muestra una interfaz sencilla, de fácil uso, para el trabajo de Docker. Se basa en Docker Swarm para poder manejar contenedores, imágenes, repositorios y más. Además del manejo de contenedores, permite el control de acceso a su interfaz mediante cuentas locales o bien a través de LDAP. Junto a lo anterior se almacena una traza de los eventos realizados, dando así la posibilidad de realizar auditorías a futuro.

[35] https://shipyard-project.com/

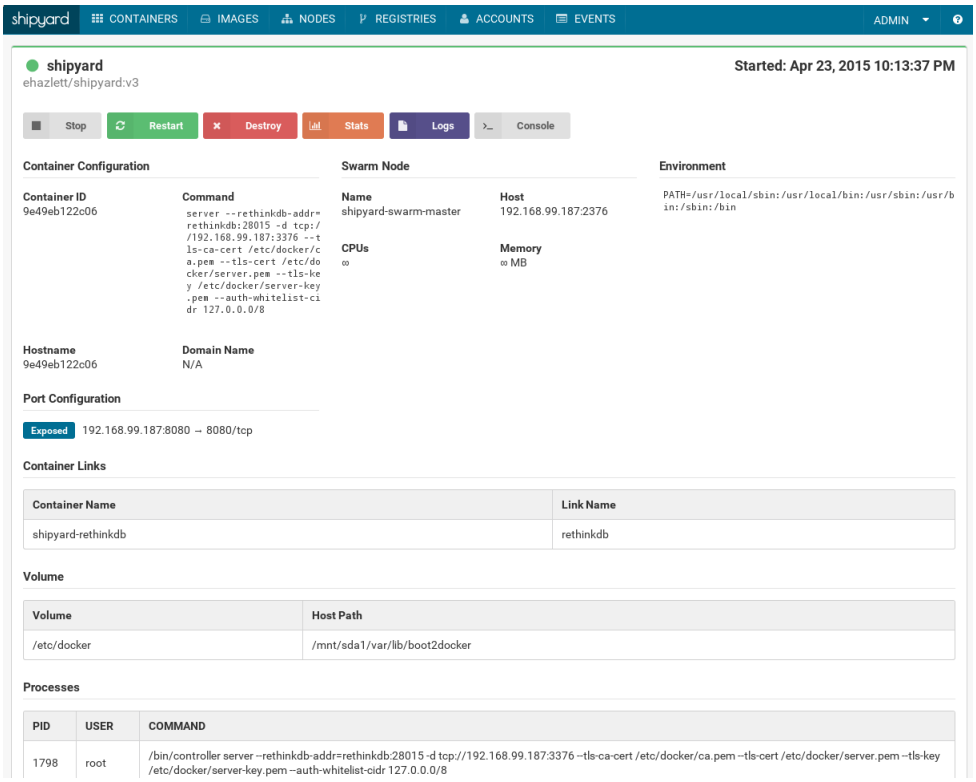


Figura 3.7: Interfaz de gestión de *Shipyard*. Fuente: https://shipyard-project.com

- *Panamax* [36] (ver figura 3.8): Se ofrece como una interfaz lista para el despliegue de aplicaciones complejas en contenedores, tan simple como arrastrar y soltar. Cuenta con una interfaz limpia y buena documentación. CenturyLink[37] es la compañía que está tras este proyecto, ofreciendo a la comunidad de usuarios su código fuente. Posee un repositorios de *templates* para facilitar el despliegue de aplicaciones. Entre sus características se tienen el manejo de contenedores desde el repositorio público de Docker[38], exportado de plantillas de aplicaciones creadas para su posterior reutilización y manejo de credenciales de acceso para diferentes usuarios.

[36] http://panamax.io
[37] https://www.centurylink.com/
[38] https://hub.docker.com/

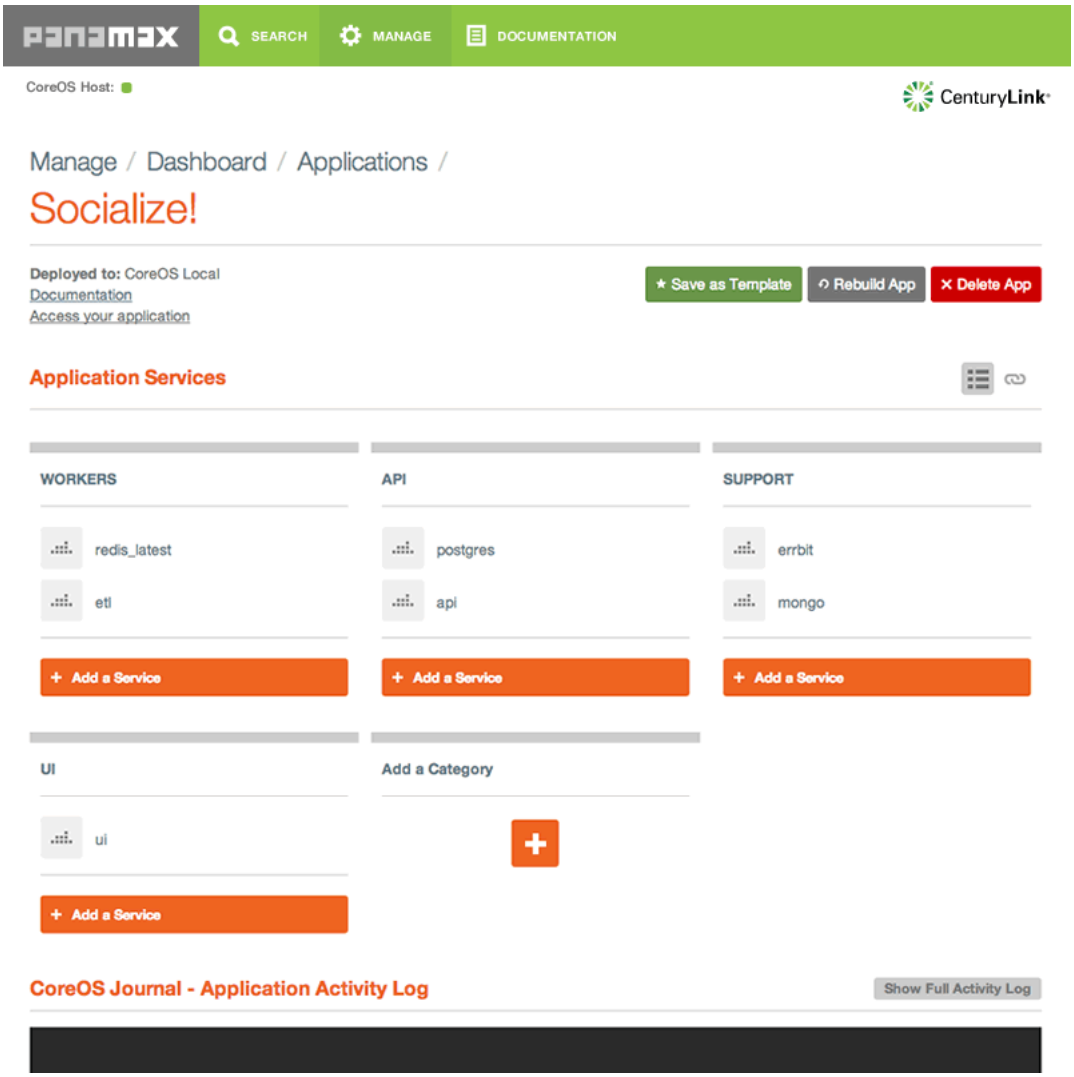


Figura 3.8: Interfaz de *Panamax*. Fuente: http://panamax.io

- *Rancher* [39] (ver figura 3.9): Rancher es una plataforma completa para ejecutar containers. Dentro de las características principales: entornos de trabajos separados, templates de servicios listos para su uso, posibilidad de agregar *host* tanto físicos como de servicios cloud como *Amazon aws*. Su instalación es mediante un contenedor, para después desplegar una interfaz gráfica.

[39] http://rancher.com

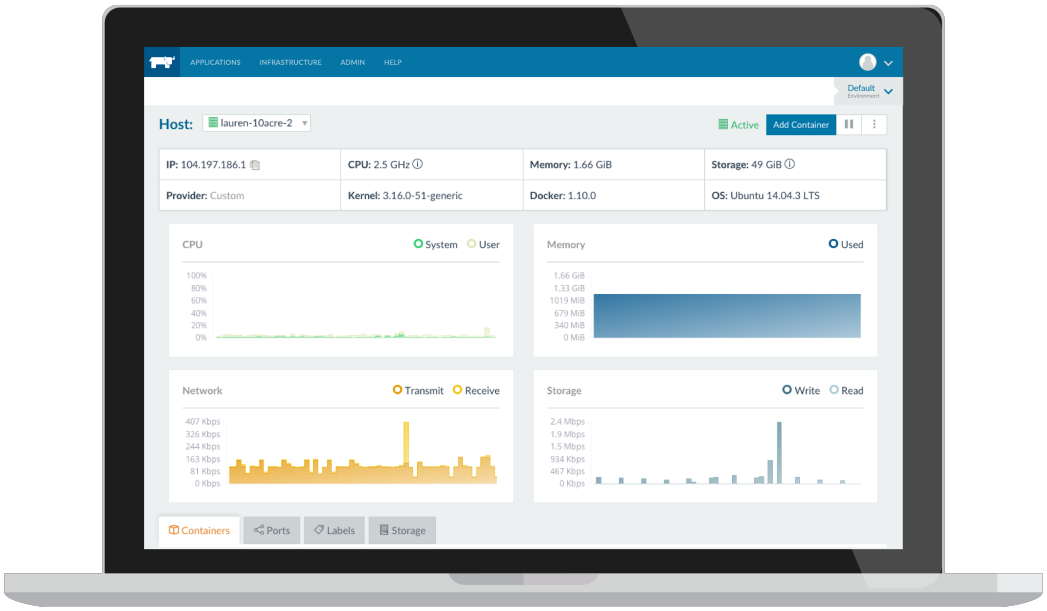


Figura 3.9: Interfaz de *Rancher*. Fuente: http://rancher.com

Para la elección de la mejor interfaz de manejo para Docker se toman en cuenta los siguientes factores:

- Documentación: contar con una buena documentación tanto para procedimiento de instalación como para el correcto uso del sistema.
- Usuarios: verificar que la comunidad de usuarios sea amplia, a fin de tener respaldo de una solución sólida.
- Licencia: se busca un sistema con licencia de código abierto.
- Soporte: poner a disposición de los usuarios un medio de contacto con los desarrolladores para resolver dudas del sistema.
- Robusto: Docker cuenta con una gran variedad de opciones en su linea de comandos, se busca que la interfaz a implementar pueda manejar todas estas opciones.

- API: es deseable que cuente con opciones de acceso diferente a interfaz web, como lo son mediante una API o bien CLI.
- Compatibilidad: que sea compatible con la versión de Docker a implementar en esta memoria.

La tabla 3.3 resume si las interfaces mostradas cuentan con los factores que se desean. La "X" indica que cumple con la característica deseada.

Tabla 3.3: Resumen de la elección de la interfaz para Docker. Fuente: Elaboración propia.

Característica	Simple Docker UI	Portainer	Shipyard	Panamax	Rancher
Documentación		X			X
Usuarios		X			X
Licencia	X	X	X	X	X
Soporte	X	X		X	X
Robusto					X
API		X		X	X
Compatibilidad	X	X	X	X	X

Por lo tanto, Rancher cumple con todos los factores necesarios para utilizarlo en esta memoria. En la sección 3.5 se discute con mayor detalle Rancher.

3.5. Rancher como orquestador de contenedores

Rancher [40] nace de la necesidad de contar con una forma sencilla de orquestar contenedores Docker. El equipo a cargo es Rancher Labs, el cual remonta su desarrollo hacia el año 2015 y hasta ahora se mantiene ofreciendo mejoras, nuevas características y solución de errores mediante la filosofía de código abierto, además de contar con una comunidad activa que brinda soporte en caso de ser necesario.

Rancher permite orquestar una serie de servidores, especificando primeramente un equipo que actuará como maestro y luego instalando el agente Rancher en los servidores que se desea que el maestro controle.

El maestro de Rancher se instalará en una máquina virtual dadas las características[41] que requiere como lo son: 1 GB de RAM y sistema operativo Centos 7, entre otros. La instalación en un servidor

[40] http://rancher.com/rancher/
[41] https://rancher.com/docs/rancher/v1.6/en/installing-rancher/installing-server/

físico sería una sobre-estimación de recursos, los cuales pueden ser utilizados en otras tareas dentro del proyecto.

La instalación de Rancher es mediante *contenedores* Docker, logrando una independencia del sistema operativo en cuanto al empaquetamiento y distribución de los programas.

3.5.1. Conceptos Generales

El ecosistema Rancher corresponde a una serie de conceptos y herramientas, las cuales son detalladas a continuación:

- *Rancher server:* Corresponde al maestro de orquestación. Para su instalación es necesario contar con una versión de Docker soportada por Rancher, además de un equipo con más de 1 GB de RAM. El requerimiento de la versión de Docker necesaria, queda subsanado por la presente en los repositorios de Centos. Cabe mencionar que el ciclo de desarrollo de Rancher está pensado en tener una versión estable y otra *latest* de acuerdo al entorno en el cual se pondrá en ejecución.

- *Rancher agents:* Se instalan en todos los nodos (servidores) que Rancher orquestará. Es responsabilidad de *rancher agent* ejecutar las acciones ordenadas por el *rancher server* y proveer una respuesta del evento ejecutado. Generalmente junto a *rancher agent* se instalan programas adicionales que son requeridos para el funcionamiento de todo el entorno, estos son: *network-services, ipsec* para manejo de servicios de red; *scheduler* como programador de tareas y *healtcheck* para comprobaciones de funcionamiento de los *contenedores*.

- *Rancher services:* Siguiendo la nomenclatura de *docker compose*, *Rancher* define como *service* uno o mas contenedores creados a partir de la misma *docker image*. Luego un grupo de servicios se conoce como *stacks*, los cuales pueden ser usados para agrupar *services* que en conjunto implementan una aplicación.

- *Rancher Catalogs*: Corresponden a plantillas predefinidas para desplegar algún software o herramienta en particular bajo el entorno *Rancher*. Existen *catalogs* soportados de manera oficial por *Rancher* y también creados por la comunidad de usuarios.

3.5.2. Rancher CLI y API

El interactuar con una serie de nodos Docker de tal forma que ejecuten tareas similares resulta costoso sin la ayuda de una herramienta de orquestación. La orquestación viene de la mano de *Rancher*, pero falta el elemento intermedio entre la acción de enviar a modelar desde la interfaz web (ver sección 1.2.1) y su procesamiento en *Rancher*.

Para el esquema de modelación mediante máquinas virtuales, se hace uso de la CLI de oVirt, a la cual se hace envío de las tareas que tiene que ejecutar y también se consulta del estado de éstas mismas. *Rancher* por su parte, ofrece dos formas de interactuar con él sin el acceso a una interfaz gráfica: CLI (*Command Line Interface*)y API (*Application Programming Interface*).

El uso de la API requiere que se adapten los comandos de la CLI que se hace uso en oVirt y de que Rancher cuente con éstos en la API. Una investigación previa da como resultado que es un tanto dificultoso trabajar con la gran cantidad de opciones disponibles. En vista de lo anterior se ocupa el esfuerzo en tomar conocimiento de la CLI de Rancher y compararla con la de oVirt. Resultados preliminares informan que los comandos resultan ser similares en ambos casos, claro está, adaptando el uso de conceptos y terminologías de oVirt a Rancher.

3.5.3. Configuración propuesta de Rancher

Habiendo expuesto lo primordial respecto a *Rancher*, se propone trabajar de la siguiente manera:

- Instalar *Rancher server* en una máquina virtual y desplegar el *agent* en los servidores que es necesario que se controlen.
- Hacer uso de CLI de *Rancher* para interactuar remotamente con él. Esto se materializa en el caso del envío a modelar desde la interfaz grafica de SIPAT y que internamente desde su código se conecte a *Rancher* para la creación del *contenedor* y ejecutar la modelación. *Rancher* por su parte, es el encargado de revisar internamente la carga de trabajo de cada servidor en vista de la cantidad de *contenedores* y enviar a ejecutar el *contenedor* deseado en el servidor adecuado.

3.6. Resumen Solución Propuesta

En vista de lo expuesto anteriormente se puede dividir la propuesta solución en dos partes: lógica y física. La parte lógica aborda el uso de software y configuración inherente a él. En el otro lado, la solución física aborda la configuración y uso óptimo del hardware disponible.

3.6.1. Solución lógica

Dentro del apartado lógico se unirán diferentes partes de software, teniendo que convivir en conjunto. Éstas son: Docker, *Rancher*, Gluster, GFS2 y COMCOT.

El detalle del uso de cada uno a continuación y en la figura 3.10:

- En cuando al sistema operativo, se instalará Centos en su versión mínima a partir de la cual se agregarán los programas necesarios. El motivo de la elección de Centos responde a que se encuentra instalado en los servidores del proyecto FONDEF anterior, lo cual ha mostrado un buen desempeño. Además, experiencia del equipo humano del proyecto actual con Centos, la posibilidad de escalar a soporte comercial ofrecido por Red Hat[42], y la compatibilidad con Docker, consolidan la elección.

- La instalación de Docker se realizará desde los repositorios de Centos. Gracias a que se realizan pruebas de integración con el sistema operativo antes de lanzar de manera estable los paquetes, se da una capa de confiabilidad al programa.

- Se configurará Docker para que almacene sus datos en la configuración a implementar en la solución física.

- Para las *imágenes de docker*, se crearán a partir de la *imagen* oficial de Fedora 25 y se subirá a un repositorio local (*Docker Registry*). A la par de éste se encontrará un repositorio que actuará como espejo (*Registry Mirror*) para *imágenes* descargadas desde internet y que desean ser accedidas posteriormente desde el entorno de trabajo. La *imagen* que se creará tendrá que incluir COMCOT y herramientas necesarias para la modelación. El listado propiamente tal de las herramientas necesarias se mostrará en el siguiente capítulo.

[42] https://www.redhat.com/

- Rancher actuará como orquestador de los servidores. Será responsabilidad de Rancher manejar la creación, eliminación y monitoreo de los *contenedores* en modelación.
- La CLI de Rancher vendrá a ser la unión entre la interfaz web desde el que se ejecuta la acción de modelar (ver sección 1.2.1 y el entorno de trabajo con Rancher.

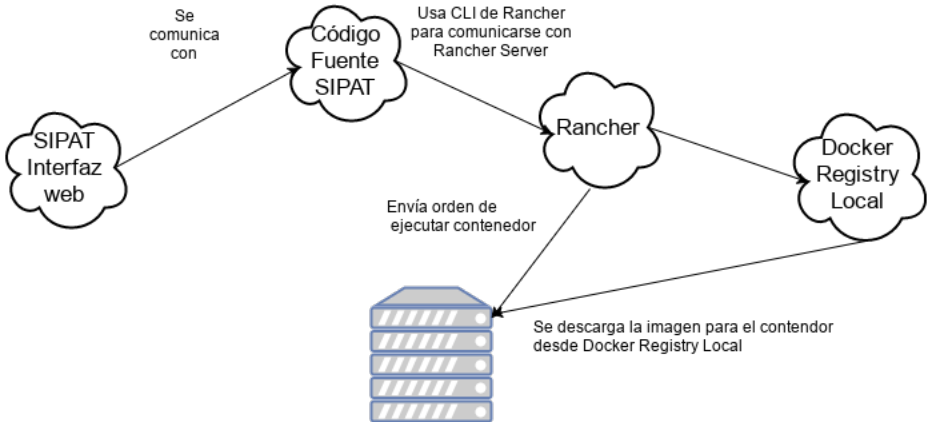


Figura 3.10: Solución lógica propuesta. Fuente: Elaboración propia.

3.6.2. Configuración física

El resumen del uso del hardware y su configuración corresponde al apartado físico. Los siguientes puntos ahondan en la propuesta y la figura 3.11 la ilustra:

- La configuración física de los servidores no sufrirá cambios. Para el almacenamiento se hará uso del provisto por el *storage*.
- Cada servidor contará con dos enlaces de fibra óptica, los cuales trabajarán en conjunto mediante una configuración de *bonding*.
- El *storage* estará particionado en volúmenes de 1 TB y se podrá acceder a ellas desde los servidores. No se realiza cambio en cuanto a los discos. El sistema RAID que está configurado es: Raid 10.
- Se presentará a cada servidor un espacio de 1TB desde el *storage*. Esta configuración permitirá trabajar con GFS2 o en su defecto Gluster (lo cual se evaluará durante la implementación), para crear un sistema de archivos distribuido, logrando 5 TB de espacio disponible.

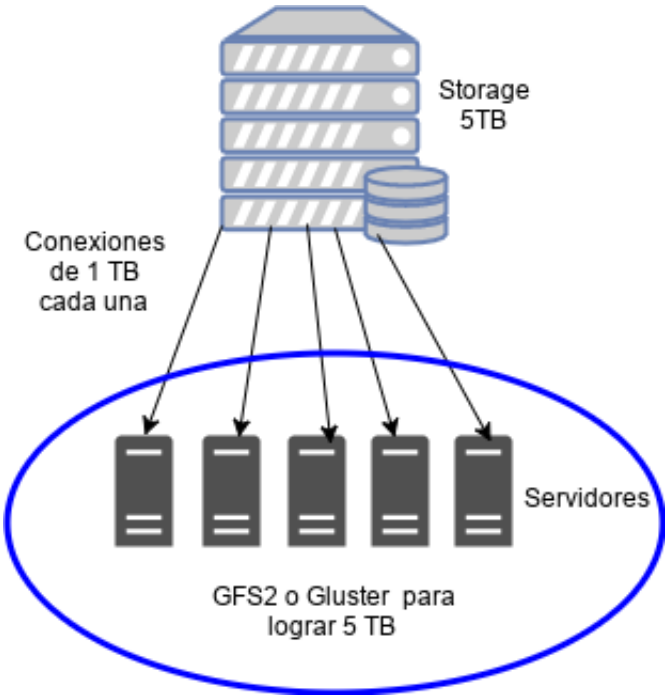


Figura 3.11: Solución física propuesta. Fuente: Elaboración propia.

4 | Construcción e implementación de la solución

En el presente capítulo se expone la implementación de las tecnologías propuestas en el capítulo anterior y su configuración. Junto a ésto, en cada apartado se explica el funcionamiento del elemento configurado y los detalles necesarios para su puesta en marcha.

4.1. Instalación base del sistema operativo

El sistema operativo que se instaló corresponde a Centos 7. [43] Se realizó una instalación en su forma mínima para posteriormente ir agregando el software necesario. Algunos puntos importantes a mencionar sobre la instalación:

- Se ocupó particionado automático del disco duro con configuración de gestor de volúmenes lógicos (LVM), en los discos presentes en cada servidor. La configuración inherente al Storage se mostrará más adelante.
- Se configuró el servicio NTP para que se sincronizara con los servidores del SHOA, los cuales manejan la hora oficial de Chile.
- La conexión de fibra óptica hacia los servidores permite hacer *boundig* tipo 4, con lo que se dejó esa configuración.
- Los repositorios *YUM* del sistema operativo se configuraron para obtener los paquetes desde servidor *Satellite*[44] del proyecto. Con esto se tiene un menor uso de ancho de banda al momento

[43] https://www.centos.org/
[44] https://www.redhat.com/es/technologies/management/satellite

de instalar programas y actualizaciones.

- El cortafuegos de los servidores se configuró de modo de aceptar conexiones solo de servicios que se encuentran ejecutando, evitando así exponer puertos sin uso.

4.2. Instalación y configuración de Docker

Para la instalación de Docker se puede elegir principalmente entre dos alternativas: instalación desde repositorio mantenido por Docker, o bien instalar desde repositorio de Centos. Se escogió instalar mediante el repositorio Centos, porque los paquetes *RPM* que se construyen son probados antes por la comunidad a fin de que funcionen de manera óptima en Centos 7.

Ahora bien, dentro de los repositorios de Centos se pueden encontrar dos paquetes de Docker: uno llamado *docker* y otro *docker-latest*. La diferencia entre ambos es la versión y la estabilidad [5]:

- *docker*: se encuentra enfocado a brindar una versión estable del binario. Si bien no con las últimas características, pero asegurando estabilidad que en muchos casos se requiere en la infraestructura de la empresa. La versión disponible en este escrito es: 1.12.6.

- *docker-latest*: trae consigo las últimas características del software. De principal interés para equipos de trabajo que necesiten hacer uso de funcionalidades agregadas recientemente. La versión disponible en este escrito es: 1.13.1.

En esta memoria se decidió trabajar con el paquete *docker*, para brindar estabilidad a la infraestructura tecnológica presente en SHOA.

Dentro del apartado configuración del binario Docker, se tienen principalmente: configuración de *Registry*, *Registry Mirror*, *driver* de almacenamiento y ruta de almacenamiento; los que se explicarán a continuación.

4.2.1. Registry

Docker registry es una aplicación que permite almacenar y distribuir imágenes Docker. [45] Las principales razones para contar con un Docker registry son: acelerar el despliegue de nuevos

[45] https://docs.docker.com/registry/

contenedores en el entorno local y asegurar la privacidad de las imágenes. Para su instalación existen básicamente dos opciones: implementar un contenedor específicamente para Docker registry u obtener el paquete presente en los repositorios de Centos llamado: `docker-distribution`.

Se escoge la instalación desde los repositorios de Centos por razones similares a las del binario Docker: el paquete `RPM` cuenta con pruebas de testing en el sistema operativo Centos. El paquete `docker-distribution` contiene la aplicación Registry, luego de su instalación es necesario configurar parámetros de acceso tales como usuario y contraseña, definir ruta de almacenamiento, puerto de acceso entre muchas otras que se pueden consultar en la documentación de Docker [46]. Para esta memoria resulta de interés las comentadas.

4.2.1.1. Registry Local

Es necesario agregar a la configuración de Docker la ruta hacia el registry local, las instrucciones para lograrlo se encuentran en el anexo A.1. Un punto a mencionar, es que no se tuvo éxito al configurar el acceso mediante certificado auto-firmado ssl, ya que Docker no lo reconocía en el sistema operativo. Por esto, se planteará como trabajo futuro una investigación en este ámbito y su implementación.

Los pasos para agregar una imagen a registry local son:

- Crear la imagen.
- Agregar un *tag* a la imagen indicando: `ip_host:puerto/usuario/nombre_imagen:version`.
- Ejecutar `docker push` con el *tag* creado anteriormente.

Una buena práctica es agregar un *tag* de acuerdo a la versión que se esté implementando. Esto es, si es un código nuevo agregar un *tag beta* a fin de hacer pruebas necesarias y una vez terminadas exitosamente agregar el *tag latest*. Al momento de desplegar el contenedor se puede elegir con facilidad el *tag* que se quiere utilizar, si ningún *tag* es declarado, se usa por defecto *latest*.

4.2.1.2. Registry Mirror

La configuración de *Registry Mirror* nace de la necesidad de contar con las distintas imágenes de Docker de manera rápida. Por defecto, imágenes que no se encuentran en el Registry local van a

[46] https://docs.docker.com/registry/configuration/#list-of-configuration-options

ser buscadas en el repositorio público de Docker. [47] Lo anterior implica un uso de ancho de banda por tiempo que se demore la descarga. Considerando que aproximadamente la imagen de Fedora comprimida pesa 70 MB y multiplicando por la cantidad de servidores (cinco), hace necesario implementar una estrategia para disponer de imágenes que se utilizan frecuentemente de manera rápida.

Para ello se implementó un Registry Mirror, que en configuración es similar al Registry Local, pero con la diferencia de que guarda las imágenes que se descargan de internet una vez y luego, las veces que se requiera posteriormente, son utilizadas desde el mirror. Un punto a notar es que no permite que se agreguen imágenes como si fuera un Registry Local, es por esto que se tuvo que configurar de manera separada el Registry Local del Registry Mirror.

Para agregar el Registry Mirror a Docker es necesario especificar el siguiente argumento en el archivo de configuración de Docker: `-registry-mirror`.

Los pasos que realiza internamente Docker al ejecutar el comando *docker run* y que hacen uso del Registry Mirror configurado son:

1. Buscar si la imagen se encuentra en el Registry Local y utilizarla.

2. Si la imagén no se encuentra en el punto anterior, buscar si existe en Registry Mirror y descargarla.

3. Si no existe en Registry Mirror, descarga la imagen de `https://hub.docker.com` y luego automáticamente la almacena en el Registry Mirror para posterior utilización en caso de que se requiera.

4.2.2. Imágenes Docker

Los contenedores Docker son instancias de las imágenes Docker en ejecución, las que necesitan ser creadas a partir de una estructura detallada en la documentación oficial. [48] Dockerfile es el archivo que contiene todas las indicaciones para la construcción de la imagen y es interpretado por el comando `docker build`. El armado de la imagen siguió unas cuantas directrices que se muestran a continuación:

[47] https://hub.docker.com/
[48] https://docs.docker.com/engine/reference/builder/

- La imagen base necesaria se obtiene desde Docker Hub[49] en donde están almacenadas las imágenes oficiales de Fedora.
- En el archivo Dockerfile se agrega comandos tales como actualización del sistema operativo e instalación de paquetes necesarios y que no vienen por defecto en la imagen oficial desde Docker Hub.
- En caso de contar con código que es necesario agregar a la imagen, se declara un clonado de repositorio Git[50] en el Dockerfile.
- Si la imagen que se está creando necesita exponer puertos, también es necesario declararlo.
- Finalmente es recomendable establecer un comando que se ejecutará una vez que el contenedor se inicie.

Una vez que se crea la imagen, esta es puesta a prueba en un servidor del Datacenter de SHOA para evaluar su funcionamiento. Si todo sale correctamente, es necesario enviarla al Registry Local para que esté disponible para todos los servidores.

Una consideración importante a mencionar es la arquitectura de las imágenes. En el caso de esta memoria se utiliza x86_64 y las imágenes base en su gran mayoría en Docker Hub están con esta arquitectura. Ahora bien, si se desea armar una imagen para una arquitectura en particular tal como ARM, es necesario armar la imagen base de forma local y luego seguir las directrices expuestas con anterioridad. Los pasos necesarios para construir la imagen en arquitecturas ARM por lo general se encuentran en los propios repositorios de las imágenes a través de GitHub.

4.3. Almacenamiento

4.3.1. GFS2

GFS2 [51] es un sistema de archivos distribuido que permite utilizar el espacio de disco disponible en un conjunto de servidores como si fuera uno solo. Las instrucciones de instalación pueden

[49] https://hub.docker.com/
[50] https://git-scm.com/
[51] https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/filesystems/gfs2.txt?id=HEAD

verificar en el siguiente tutorial [52] y a modo resumen se enumeran:

1. Instalar GFS2.
2. Crear un cluster con todos los servidores disponibles.
3. Definir el espacio de disco que se usará para el cluster.
4. Crear el sistema de archivos compartido.
5. Montar en cada servidor el sistema de archivos.

Luego de la implementación de GFS2 se obtiene un espacio de 5 TB, lo que indica una correcta configuración. Con el paso del tiempo surgieron situaciones que pusieron a prueba el sistema de archivo como lo fue el reinicio de un servidor. En caso de reinicio de un servidor, el sistema de archivos GFS2 es capaz de reconocer y solventar la ausencia de un equipo en su clúster; posteriormente, cuando el equipo se encuentre preparado para ser ingresado nuevamente al cluster, GFS2 también es capaz de tomar las medidas correspondientes para que el sistema de archivos permanezca estable.

Luego de un reinicio no planeado se observó lo descrito anteriormente, con la salvedad de que la reincorporación del servidor reiniciado demoró sobre 3 horas. En un comienzo se pensó que correspondía a una situación fortuita y que el servidor no podría ser ingresado al cluster ya que el reinicio fue no planeado. Se confirmó que finalmente se pudo unir nuevamente al cluster, pero el tiempo que demoró no fue el mejor: 3 horas. Se realizó reinicio de otros servidores para estudiar si ocurría algo similar y así fue: el tiempo promedio fue de 3.5 horas.

Ésta situación fue el gatillante para evaluar otra alternativa a GFS2, tal que se comporte de una manera mejor en caso de reinicio. Se encontró como alternativa a Gluster.

4.3.2. Gluster

La instalación de Gluster[53] es proporcionada desde los repositorios Centos y su configuración fue de acuerdo a la documentación oficial. [54] Con la instalación y configuración finalizada, se cuenta

[52] https://www.server-world.info/en/note?os=CentOS_7&p=pacemaker&f=3
[53] https://www.gluster.org/
[54] http://docs.gluster.org/en/latest/

con un sistema de archivos distribuido listo para ser utilizado. Hay cinco servidores proporcionando 1 TB cada uno para lograr el total de 5 TB.

Es necesario el montaje del volumen de 5 TB en cada servidor para que pueda ser utilizado. El tipo de montaje puede ser principalmente mediante NFS o mediante el cliente nativo de gluster. Se decidió montar mediante este último, ya que por experiencias previas del equipo de trabajo se habían obtenido poco rendimiento con NFS.

Finalmente para poner a prueba el sistema configurado, se crearon una serie de archivos en el punto de montaje de Gluster. Se pudo observar como exitosamente se sincronizaban a través de los servidores y dichos archivos iban apareciendo disponibles para su acceso.

4.3.3. Almacenamiento para Docker

Luego de tener configurado el almacenamiento en su forma generalizada para los servidores, es necesario indicar a Docker donde se encuentra ese almacenamiento. Para esto se configura la carpeta donde se almacenan los contenedores, volúmenes e imágenes al punto de montaje de Gluster. Con esto, cada vez que se inicie un contenedor se creará en el espacio provisto por Gluster, asegurando mayor espacio para los contenedores.

4.4. Instalación y configuración de Rancher

Rancher[55] es una herramienta que permite gestionar de una manera sencilla mediante una interfaz gráfica contenedores Docker, ya sea en un solo host como en múltiples hosts. Para realizar la instalación es necesario tener ejecutando Docker, ya que hace uso de un contenedor. Rancher posee dos versiones para su uso: *stable* y *latest*. La primera versión es la recomendada para los entornos de producción, mientras que la segunda es cuando se requiere acceso a las últimas funcionalidades. En esta memoria se utilizará *stable*, la cual se encuentra en su versión 1.6.2.

Los pasos que se siguieron de acuerdo a la documentación [56] fueron:

- Ejecutar comando *docker run* para su instalación.
- Configurar mediante navegador web un usuario y contraseña.

[55] http://rancher.com/
[56] http://rancher.com/docs/rancher/v1.6/en/quick-start-guide/

- Por defecto Rancher guarda su configuración dentro del contenedor, para mejorar la persistencia de esos datos es recomendable configurar una base de datos externa, lo cual se realizó.
- Se configura base de datos mysql[57] externa.
- Se detiene contenedor y se vuelve a ejecutar, añadiendo como parámetro las opciones de conexión hacia la base de datos.

Hasta este punto se tiene ejecutando Rancher con su configuración delegada a una base de datos externa. Ahora resta el paso siguiente de agregar los servidores.

Rancher se está ejecutando en una máquina virtual y tiene que hacer uso de cinco servidores que son exclusivos para la ejecución de contenedores Docker. Teniendo el sistema operativo base instalado en cada servidor, se instaló y configuró Docker de acuerdo a la sección anterior. Un punto importante a considerar es que cada host que se anexe debe tener la hora configurada correctamente, ya que Rancher realiza un chequeo de seguridad antes de permitir la conexión mediante *shell* a los contenedores. En caso de no contar con la hora correcta, no permitirá la conexión. Posteriormente para el agregado a Rancher de cada *host*, es necesario ir a la interfaz e indicar la IP del servidor, así Rancher generará un comando con esta información. Luego de ejecutar ese comando en el host de destino, se descarga el agente Rancher y herramientas de red. Con lo anterior, se finaliza el agregado del host al orquestador Rancher. Dicho proceso es necesario ejecutarlo tantas veces como servidores existan.

4.4.1. Rancher CLI

Rancher CLI es la herramienta de conexión entre el servidor creador de modelaciones y Rancher. La forma en que se implementó corresponde a un binario disponible para la descarga desde el sitio GitHub de Rancher, [58] luego es necesario indicar la *url* del servidor en que se encuentra ejecutando Rancher y agregar credenciales de acceso.

El código fuente para los pasos de modelación interactúa con el binario Rancher CLI dependiendo de las tareas a ejecutar. Las interacciones y respuestas del binario son:

[57] https://www.mysql.com/
[58] https://github.com/rancher/cli/releases

- Obtener la cantidad de contenedores en ejecución, para que la creación de un nuevo contenedor no sobrepase un límite definido y no disminuir el desempeño.
- Crear el contenedor a partir de la imagen para modelamiento creada anteriormente. Rancher internamente se encargará de crear el contenedor en el servidor con los mejores recursos disponibles y como respuesta informará datos de creación como lo es el ID del contenedor.
- Una vez creado el contenedor, se envía una orden a este para ejecutar una copia de archivos desde el servidor de modelación hacia el contenedor.
- Si la copia finalizó exitosamente, se puede comenzar el proceso de modelación.
- Una vez terminada la modelación, se envía la orden a Rancher CLI para que elimine el contenedor creado y así dar paso a un nuevo proceso de modelado.

El éxito de Rancher CLI, basado en las pruebas ejecutadas de los puntos expuestos, viene de la mano de una configuración similar de los servidores de modelado como también el armado correcto de la imagen a partir del cual se ejecutará el contenedor.

4.4.2. Contenedor *janitor* para limpieza

En el proceso de modelado explicado en el punto anterior, existe un paso en el cual se elimina el contenedor utilizado luego de terminar la modelación. Se detectó que por defecto quedan datos en el disco duro, tales como: archivos de modelación, archivos del contenedor, entre otros. Para hacer frente a esto, existe una herramienta llamada "janitor"[59], la cual se despliega en todos los host un contenedor con la herramienta. Esta herramienta se ejecuta, cada cierto tiempo establecido por el usuario, en busca de contenedores cuyo estado sea eliminado y que cuenten con archivos residuales, volúmenes e imágenes sin uso, y los limpia del sistema. Lo anterior asegura que luego de terminar una modelación se recupere el espacio de manera automática.

[59] https://github.com/meltwater/docker-cleanup

4.5. Monitoreo del sistema

La configuración efectuada en los pasos anteriores requiere de un monitoreo para asegurar su buen funcionamiento. Este monitoreo se puede dividir en tres dependiendo del componente.

4.5.1. Monitoreo del sistema operativo

El monitoreo del sistema operativo como de los recursos de cómputo y almacenamiento de los servidores viene de la mano de Nagios, [60] ya que se cuenta con una instalación previa de este sistema y personal de SHOA está capacitado en su funcionamiento.

Para que Nagios sea capaz de monitorear los servidores del SHOA, es necesario instalar el cliente en cada uno y declarar en el servidor estos nuevos elementos a monitorear.

4.5.2. Monitoreo de Rancher

El monitoreo de Rancher lo provee su interfaz gráfica en la cual se puede observar el estado de los contenedores y de los hosts. En el supuesto de fallo de un contenedor, Rancher mostrará de forma gráfica un error para alertar la situación y ofrecerá revisión de logs para encontrar el problema y establecer una solución.

4.5.3. Monitoreo de Gluster

Gluster ofrece la opción de monitoreo mediante un script anexable a Nagios escrito por un tercero [61] o bien un script python que se puede encontrar en el repositorio GitHub de Gluster [62]. En esta etapa se escogió el script python por su fácil ejecución, teniendo en mente una futura migración a Nagios una vez consolidado el almacenamiento Gluster.

[60] https://www.nagios.org/
[61] https://github.com/atlantos/nagios-check-gluster
[62] https://github.com/gluster/gstatus

5 | Validación de la Solución

En este capítulo se mostrarán los resultados de la solución implementada, dando énfasis en la comprobación de su funcionamiento de forma óptima y los ajustes necesarios.

5.1. Lecciones aprendidas

5.1.1. Modelamiento

El proceso de modelación involucra diferentes etapas las cuales se explican con más detalle a continuación.

Inicio del proceso

Esta etapa comienza cuando se hace click en el sismo que se desea modelar desde la interfaz web y entra en trabajo un *script* Python que se encarga de generar los archivos necesarios que se enviarán hacia los contenedores. Con los archivos listos, comienza la tarea de Rancher de crear un contenedor e informar que se encuentra preparado para su uso mediante su CLI. Esta acción en tiempo se demoró 2 a 17 segundos, tiempo bastante razonable considerando su símil de máquina virtual que demora en promedio 2 minutos. La diferencia en tiempo se debe al uso de recursos de la cola de creación en el orquestador Rancher y a la descarga de la imagen Docker para modelar en caso de que no se encuentre anteriormente en el servidor.

Copia de archivos hacia contenedor

Luego de la creación, es necesario copiar y enviar archivos hacia el contenedor. La copia se realiza mediante *SSH*[63] ejecutado desde el contenedor hacia la máquina virtual en la que se encuentran los archivos creados para la modelación. El tiempo que demora la copia es del orden de un minuto. Este tiempo se logra gracias a la conexión de fibra óptica configurada en los servidores. Cabe mencionar, que la máquina virtual que donde se crean los archivos para la modelación no se intervino, ya que no era el alcance de esta memoria.

Una vez copiados los archivos se empieza el proceso de modelación mediante *Comcot*, generando diversos archivos de salida de datos. El detalle de los tiempos logrados se mostrará con detalles en la sección 5.3.

Término de la modelación

Luego de terminada la ejecución del binario *Comcot* comienza el proceso de "Categorización de Riesgos", conocida dentro del proyecto como *Hazcat*. Es la penúltima etapa del proceso de modelado en la que con un *script* Python se analizan los resultados de *Comcot* y se almacena su información en una base de datos. El último paso corresponde a comprimir todos los archivos generados y almacenarlos en un lugar permanente (carpeta situada en equipo storage). El método de compresión es *xz*[64] el que ha conseguido un archivo de menor tamaño frente a otras opciones de compresión como *gz*[65] de acuerdo a pruebas del equipo del trabajo del proyecto FONDEF.

Tanto *Hazcat* como el proceso de modelación hacen uso en forma intensa de la CPU durante toda su ejecución. Además, el proceso de compresión y envío del archivo hacia el equipo storage hace uso de escritura y lectura de disco más que los otros procesos.

Una vez finalizado el envío del archivo comprimido, se procede a borrar el contenedor. Para esto Rancher recibe la orden y la realiza en un promedio de 5 segundos, proceso mediante el cual se procede a detener el contenedor y luego a borrarlo. Luego, *Janitor* entra en ejecución para borrar archivos residuales de todo el proceso. *Janitor* se configuró para realizar la limpieza cada 5 minutos.

Realizando una comparación frente a máquinas virtuales se puede observar la rapidez al momento

[63] https://tools.ietf.org/html/rfc4251
[64] https://tukaani.org/xz/format.html
[65] http://www.gzip.org/

de detener el contenedor, el cual toma en promedio 2 segundos versus un promedio de 3 minutos en máquinas virtuales. Con ésto, la combinación de *script* Python para enviar a modelar, Rancher y *Janitor* resultó ser un caso de éxito.

Las etapas mencionadas anteriormente se pueden ordenar de la siguiente manera correlativa:

1. Etapa de creación: en donde se crea la máquina virtual o el contenedor, para ejecutar las modelaciones.

2. Etapa de encendido: corresponde al encendido de la máquina virtual o a la ejecución del contenedor.

3. Etapa de copia de archivos: para iniciar las modelaciones es necesario copiar los archivos propios de estas.

4. Etapa de modelamiento: etapa principal de todo el proceso, ya que se realizan los cálculos matemáticos de las simulaciones.

5. Etapa de compresión: para hacer un uso eficiente del espacio disponible, se comprimen los resultados obtenidos en la etapa de modelamiento.

6. Etapa de limpieza: es la última etapa y corresponde a limpiar los archivos residuales del proceso.

5.1.2. Almacenamiento

Respecto al almacenamiento, se encontraron diversas situaciones durante la marcha blanca del proceso de simulación en contenedores que se explicarán a continuación.

GlusterFS se utilizó en un principio para contar con un almacenamiento distribuido. Tal almacenamiento se evidenció de forma exitosa una vez configurado todo lo necesario. Posteriormente fue puesto a prueba con tareas tales como creación de contenedor y modelación. Con respecto a la primera tarea de creación de contenedor, se notó que GlusterFS no tiene compatibilidad completa con SELinux[66] (al menos con la versión que se trabajó en esta memoria), por lo que obligó a crear el

[66] https://github.com/gluster/glusterfs/issues/55

contenedor en modo *privilegiado* pudiendo hacer uso de características directas del hardware del *Host* y a la vez generando una posible brecha de seguridad.

El segundo punto que se evidenció con GlusterFS fue durante la modelación. En algunas etapas de la modelación se hacia uso excesivo de Gluster en el servidor (alrededor de 400 % de cpu) y al mismo tiempo el proceso *COMCOT* disminuía su uso de cpu al 20 % por 10 minutos. Pruebas posteriores con modelaciones simultáneas en un único host mostraron que lo anterior se tornaba más grave llegando a tiempos de espera superiores a dos horas. Esto hizo reconsiderar drásticamente el uso de Gluster.

Un análisis más detallado logró concluir que:

- El problema de demora en tiempo se debió a que partes del proceso de modelación requieren bastante uso de disco, ya sea creando o leyendo archivos.
- Se intentó con una nueva versión de Gluster para asegurarse de que no era algún *bug* presente en la versión.
- Se identificó que el problema radica en el cliente Gluster. Lo anterior fue gracias a un montaje de la misma arquitectura usada en esta memoria en un entorno fuera de SHOA, instalando componentes desde cero, y evidenciando la situación de exceso de uso de CPU en el cliente de Gluster.

Debido a las dificultades anteriores y para no retrasar la puesta en marcha de las modelaciones, se configuró un almacenamiento separado en cada servidor. Esto representa una desventaja en cuando a espacio disponible para las modelaciones, pero se genera una ganancia en tiempo bastante considerable: con Gluster una modelación podía tomar 20-23 horas, mientras que con el almacenamiento por separado se demora 13 a 15 horas. También el aplicar almacenamiento por separado tiene como consecuencia una liberación de CPU, ya que no se cuenta con Gluster ejecutando en cada servidor.

Otro cambio que se realizó durante una actualización de rutina del sistema operativo de los servidores fue el cambio del controlador de almacenamiento utilizado por Docker. Por defecto se utiliza la opción *devicemapper* en Centos, pero de acuerdo a la documentación de Docker es preferible utilizar *OverlayFS2*. [67] Al principio de toda la implementación de la arquitectura no se

[67] https://docs.docker.com/engine/userguide/storagedriver/selectadriver/

configuró este controlador de almacenamiento por limitaciones de la versión del sistema operativo, la cual se solventó con una actualización reciente en la que se incluyó *OverlayFS2* de forma más estable. [68] Razones adicionales para el cambio de almacenamiento son errores que surgían con el controlador *devicemapper* al intentar borrar un contenedor que ha sido detenido en forma forzada, como se puede evidenciar en el link[69] y cuyo resultado confirma el uso de *OverlayFS* por sobre *devicemapper* para una mayor estabilidad y desempeño.

5.2. Análisis comparativo de uso de recursos

La utilización de recursos computacionales se torna de vital importancia a la hora de evaluar el desempeño de la solución propuesta. El detalle de los elementos involucrados se muestra a continuación:

- **Preparación de archivos:** Esta etapa es ejecutada por la máquina virtual *creador*. Se hace uso del programa *Octave*[70] que es el que ocupa mayor tiempo de CPU. Como salida del programa se obtienen los archivos listos para ser copiados al contenedor. Al realizar una comparativa frente a las máquinas virtuales no se observa diferencia, ya que es el paso previo común a ambos tipos de simulación (máquinas virtuales y contenedores).

- **Conexión con Rancher y creación de contenedores:** En este paso primero se ejecuta la CLI de Rancher la cual consumió recursos mínimos de la máquina virtual. Luego para la creación de los contenedores, se ejecuta una parte en la máquina virtual de Rancher y otra parte en el servidor en el cual quedará el contenedor. En la primera parte el mayor trabajo lo realiza Rancher mediante un programa Java que se encarga de enviar la orden hacia el servidor, para tener un buen rendimiento se asignaron 4 CPU y 6 GB de RAM. Respecto a la segunda parte, Docker ocupa tiempo de procesamiento en el servidor creando el contenedor, en promedio se demora 6 segundos. Finalmente haciendo una comparación frente a máquinas virtuales se tiene que existe un proceso similar que involucra a oVirt CLI [71] y luego una etapa de creación de la máquina virtual por parte de oVirt en un servidor determinado. Una diferencia a notar

[68] https://goo.gl/9GYxjt
[69] http://blog.cloud66.com/docker-with-overlayfs-first-impression/
[70] https://www.gnu.org/software/octave/
[71] https://www.ovirt.org/develop/release-management/features/infra/cli/

es que la etapa de creación del contenedor resultó mucho más rápida que la de una máquina virtual, de 3 minutos en promedio en máquinas virtuales contra 12 segundos en promedio con Docker . Lo cual se debe a la arquitectura de trabajo de Docker, tipo de almacenamiento configurado para cada servidor (Overlay2) y el inicio casi instantáneo de un contenedor.

- **Conexión SSH:** Es el primer proceso que se ejecuta dentro del contenedor una vez creado. Su uso en recursos corresponde más a disco duro y CPU en el contenedor. Respecto a la máquina virtual, no hay demasiado que comentar, ya que el proceso es similar; por lo tanto no hay diferencias en este paso.

- **Modelación:** En esta etapa se consumió CPU del contenedor en todo el proceso de modelación al 100 %, detalle sobre tiempo se mostrará en la sección 5.3. Este consumo de CPU era el esperado ya que COMCOT es el responsable de esto. En comparación a una máquina virtual se observa un mismo uso de CPU, pero con un tiempo menor de modelación. Este tiempo se analiza más a fondo en la siguiente sección, pero a priori se debe a la arquitectura de Docker.

- **Compresión:** Utilizó el 100 % de CPU en el contenedor durante su ejecución, lo cual igual que el proceso de modelación, es esperable. Frente a las máquinas virtuales se observa una disminución del tiempo gracias a Docker, concordante a la disminución de tiempo en modelación.

5.3. Análisis comparativo de uso de tiempo

Respecto a los tiempos de modelación con la solución implementada se pueden destacar los siguientes puntos ocupando Gluster y almacenamiento separado en cada servidor.

El uso de Gluster introdujo demoras bastante considerables en el proceso. Un ejemplo claro es el comienzo de la modelación en el cual se preparan archivos pequeños por parte de COMCOT. Para una simulación demoró 10 minutos y para una cantidad de 20 simulaciones simultáneas demoró más de 2 horas. Este tiempo en comparación con máquinas virtuales es mucho mayor, ya que en éstas se demoran 2-3 minutos en una simulación. Para una cantidad cercana a 20 máquinas virtuales, el tiempo era tambien de 2-3 minutos.

Esta demora se debía a un uso excesivo de CPU por parte del cliente Gluster, el cual a la larga disminuía la cantidad de contenedores ejecutándose de manera simultánea. Cabe notar que en los logs de Gluster no se encontró mayor detalle de esta situación. Finalmente una demora similar de tiempo ocurrió en el proceso de compresión, ya que involucra lectura y escritura mayor a la normal en el servidor, el tiempo de compresión utilizando Gluster se multiplicaba por dos respecto al tiempo de máquinas virtuales, esto es, de 2 horas se pasa a 4 horas. Gluster debía sincronizar los cambios a través de todos los servidores presentes en el cluster. Para asuntos de registro, una modelación con Gluster se demora 23 horas y 20 simulaciones aumenta demasiado, por temas del primer paso de la modelación y luego la compresión, llegando a más de un día.

Por el problema expuesto anteriormente se tuvo que configurar de otra manera el almacenamiento para los servidores. Se eligió almacenamiento por iSCI separado de 1 TB para cada servidor. Ésta solución se puso a prueba con modelaciones las cuales tuvieron mejoras considerables respecto a Gluster. Por un lado, una modelación rondó las 14 horas (versus 23 con Gluster) y no se observó la ralentización del comienzo de la modelación. Además el proceso de compresión se ejecutó exitosamente con un tiempo menor que en máquinas virtuales. Ahora bien, al aumentar el número de simulaciones en un servidor, se verificó un aumento de tiempo en simulación cercano a 17 horas. Este aumento de tiempo es normal si se toma en consideración el gran uso de recursos por parte de las simulaciones en forma simultánea, en especial el de disco duro y RAM.

Respecto a tiempos en Rancher se puede mencionar que cada contenedor que se crea tiene que ser orquestado de manera que se decida en cual servidor disponible levantarlo. Este tiempo de decisión se lleva a cabo en la máquina virtual de Rancher y toma 10 segundos como máximo. Si se compara este tiempo con el orquestador oVirt, demora 1 minuto como máximo. Luego el proceso de ejecución del contenedor, toma como máximo 5 segundos, tiempo que frente a máquinas virtuales lleva entre 2 a 3 minutos.

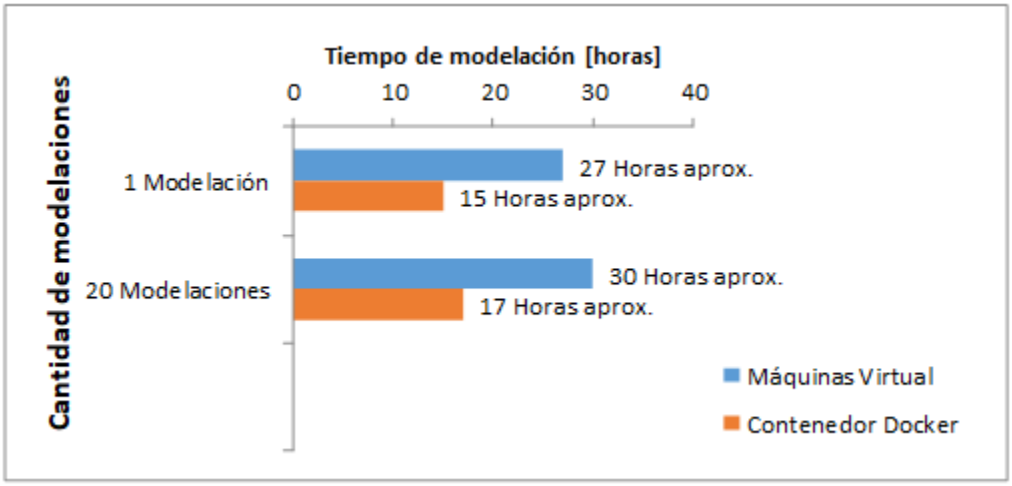
5.3.1. Comparación con máquinas virtuales

Para establecer una comparativa entre máquinas virtuales y contenedores Docker es necesario tener en cuenta los siguientes puntos configurados en el proyecto:

- El almacenamiento es provisto por el equipo *storage* con 1 TB para cada servidor.

- En promedio cada servidor se encuentra ejecutando de 2 a 3 contenedores adicionales a los de modelación, que corresponden a código de otras aplicaciones que necesita el sistema.
- Los servidores que se encuentran ejecutando los contenedores corresponden a una parte de los servidores que se encontraban anexados a oVirt.
- Los tiempos de máquinas virtuales corresponden a las últimas simulaciones realizadas en estas.
- Los tiempos de simulación corresponden a territorio continental sin la Isla de Pascua, ya que ésta requiere un análisis más a fondo.

Figura 5.1: Tiempos de Modelación en Horas. Fuente: Elaboración propia.



Del gráfico 5.1 es importante destacar la cantidad de simulaciones que se compararon, ya que 20 modelaciones representa aproximadamente un 75 % de la capacidad total en toda la plataforma de virtualización mediante máquinas virtuales; mientras que con la solución de contenedores, este número representa la capacidad máxima de modelaciones de un solo servidor.

Se puede observar que para el apartado de una modelación existe una mejora en porcentaje del 45 % y para 20 modelaciones simultáneas en el mismo servidor de un 44 %. Estos porcentajes traducidos en horas y realizando la sumatoria de tiempo ganado, da como resultado una mayor cantidad de simulaciones paralelas, aproximadamente 75, que se pueden realizar en un tiempo determinado, por ejemplo de una semana, y así contar con una base de datos más poblada.

Además, se puede comentar la ventaja de la arquitectura Docker que viene a reducir drásticamente la sobrecarga que presenta la solución de máquinas virtuales provista por oVirt. Esta ganancia se atribuye en la totalidad a la arquitectura de Docker, ya que permite hacer un uso más eficiente del hardware del servidor, eliminando la capa de *hypervisor*. El binario Comcot que realiza las tareas matemáticas de simulación no se modificó en esta memoria, por lo que no tiene impacto en este tiempo ganado. También, se permite una mayor cantidad de modelaciones de forma simultánea, estableciéndose como número óptimo entre 17 a 20 contenedores adicionales a los que no son de simulaciones y que se encuentran ejecutando en los servidores. La razón de este número viene fundada gracias a dos puntos:

- La cantidad de almacenamiento: Al ser un espacio fijo, es una de las restricciones a tener en cuenta y que juega un papel importante dada la gran cantidad de datos que se generan producto de las simulaciones.

- RAM: Este es un elemento bastante interesante, ya que se detectó que la cantidad requerida por cada simulación es de aproximadamente 1.5 GB. Luego, si se hace uso excesivo de RAM se ocupará el espacio de intercambio (SWAP) el que ralentiza las modelaciones ya que hace uso del disco duro del servidor. En base a lo anterior se modificó el parámetro de kernel *swappiness*[72] para utilizar el SWAP solo cuando se llene al 90 % la memoria RAM. Además de lo anterior se utilizó *zram*[73] para comprimir RAM, lo que redujo tiempos de modelación en aproximadamente un 40 %. Por lo tanto se presenta como recomendación a SHOA la compra de más RAM para los servidores de modelación para hacer frente a esta limitación y tener como limitante solo el espacio disponible. Cálculos a priori establecen un aumento de 5 a 7 contenedores por servidor si se concreta el cambio dejando cada servidor con mínimo 90 GB de RAM.

El *speedup* logrado se muestra en la figura 5.1. El muestreo de simulaciones en contenedores es de aproximadamente 300 simulaciones. Existe una buena ganancia de tiempo favor, lo cual es gracias a la arquitectura de Docker.

En la tabla 5.2 se muestra una comparativa de los tiempos y recursos utilizados en máquinas virtuales y en Docker.

[72] https://www.kernel.org/doc/Documentation/sysctl/vm.txt
[73] https://www.kernel.org/doc/Documentation/blockdev/zram.txt

Tabla 5.1: Speedup de simulaciones en un servidor. Fuente: Elaboración propia.

	Máquinas virtuales	**Contenedores**	**Speedup**
Una simulación	27 Horas promedio	15 Horas promedio	1.8x
Veinte simulaciones paralelas	30 Horas promedio	17 Horas promedio	1.76x

Tabla 5.2: Tabla comparativa de tiempos y recursos entre máquinas virtuales y contenedores Docker. Fuente: Elaboración propia.

Característica	**Máquinas Virtuales**	**Contenedores Docker**
Etapa de creación	2 minutos aprox.	2-7 segundos
Etapa de encendido	2 minutos aprox.	5 segundos
Etapa de copia de archivos	menos de 1 minuto	menos de 1 minuto
Etapa de modelamiento	30 horas aprox.	17 horas aprox.
Etapa de compresión	2 horas aprox.	1,5 horas aprox.
Etapa de limpieza	3 minutos aprox.	5 segundos aprox.
Tasa de muestreo	5000 simulaciones aprox.	300 simulaciones aprox.

A modo se resumen se muestra la tabla 5.3 con elementos comparativos entre la tecnología clásica de virtualización por máquinas virtuales y su contraparte contenedores Docker.

Tabla 5.3: Tabla comparativa de oVirt con Rancher. Fuente: Elaboración propia.

Característica	**Máquinas Virtuales**	**Contenedores**
Orquestador	oVirt	Rancher
Tiempo de creación y encendido	4 minutos aproximadamente.	12 segundos aproximadamente.
Espacio requerido	Igual que una máquina física (orden de GB).	Más liviano, un tercio aproximadamente respecto a una máquina virtual, ya que comparte librerías del *host*.
Software incluido	Todo lo que viene con la ISO de la distribución instalada.	Software base desde el cual se puede ir agregando lo necesario.
Tiempo de vida	Pensadas en ejecutarse por siempre y con pocos cambios de configuración.	Pensados en ser efímeros, para la ejecución de un proceso y luego detenerse.
Seguridad	Seguras por defecto gracias a su capa de aislamiento provisto por el *hypervisor*.	Menos seguros, pero se puede mejorar siguiendo recomendaciones de ejecución.
Sobrecarga de trabajo	Considerable dado que se ejecuta un entorno de trabajo virtual de forma completa.	Poca, ya que se tiene lo justo y necesario dentro de él.
Almacenamiento	Pensado en ser persistente.	Limitado al tiempo de vida del contenedor, aunque se puede indicar que use almacenamiento permanente.
RAM	Uso considerable dado la cantidad de bibliotecas del sistema operativo que carga durante su ejecución.	Poco uso, ya que las bibliotecas se encuentran compartidas con el sistema operativo del *Host*.
Usos	Ejecución de proyectos completos (web + base de datos, etc).	Pensado en ejecutar en un contenedor un solo proceso.
Interfaces de administración	CLI provista por oVirt.	CLI de Rancher.

6 | Conclusiones

A continuación se muestran las conclusiones obtenidas del desarrollo de esta memoria. En primer lugar se presenta la conclusión general que va de mano del objetivo principal y luego las conclusiones específicas de acuerdo a los objetivos planteados al comienzo.

6.1. Conclusiones Generales

Es necesario señalar que se cumplió cabal y exitosamente el objetivo principal planteado, es decir, *Proponer y evaluar un sistema alternativo a máquinas virtuales para modelamiento computacional de alto desempeño en el proyecto SIPAT*. Esto se logró gracias a Docker y el diseño de su arquitectura, ya que presenta una menor sobrecarga, entendiéndose ésta como una ventaja por sobre las máquinas virtuales.

Los resultados obtenidos sitúan a Docker como una tecnología que ha logrado madurez con el paso de los años y que se adapta a todos los requerimientos del proyecto SIPAT. Junto a lo anterior, vino a mejorar el desempeño de la arquitectura tecnológica de SIPAT, evidenciándose menores tiempo requeridos para simular tanto un escenario tsunamigénico, como múltiples escenarios. En consecuencia se pudo modelar una mayor cantidad en forma paralela respecto a máquinas virtuales con el mismo hardware disponible.

6.2. Conclusiones Específicas

Estas conclusiones van alineadas con los objetivos específicos planteados. Se analizará el detalle de cada uno a continuación:

- **Evaluar Docker como alternativa a máquinas virtuales de acuerdo al uso que se le da en**

el proyecto SIPAT: Se logró comprobar que Docker es una tecnología idónea para modelación computacional en el proyecto SIPAT, ya que de acuerdo a los resultados obtenidos, logra ejecutar el programa *Comcot* y obtener los mismos resultados que en máquinas virtuales, pero con una ganancia de tiempo a favor.

- **Evaluar e implementar un orquestador para Docker que se adapte a la arquitectura del proyecto SIPAT:** Al momento de escribir esta memoria existían diversas herramientas orientadas a la orquestación de Docker. Luego de un análisis de cada una se decidió implementar *Rancher*, el que provee una interfaz gráfica de uso sencillo para poder administrar los contenedores a lo largo de los servidores de trabajo. El que *Rancher* cuente con una CLI hace mas sencillo el trabajo, dando la posibilidad de administrar no necesariamente desde la interfaz gráfica todo el stack de contenedores.

- **Evaluar optimizaciones a la plataforma propuesta para obtener mejor rendimiento:** El optimizar la plataforma implementada permite obtener un mejor rendimiento en la modelación, lo que se traduce a un mayor número de contenedores modelando en forma paralela o bien en un menor tiempo de modelación. Entre los aspectos que se logró evidenciar e implementar a partir de los resultados encontrados son:

  - RAM: Existió un uso notable de memoria de intercambio (*swap*) en las primeras pruebas de modelación, lo que llevó a un modificación en la configuración del kernel del sistema operativo para el mejor manejo de ram. Este cambio fuerza el uso de *swap* solo cuando la ram se llena al 90 %. Dentro de este mismo apartado se hicieron pruebas de implementación de *zram* y los tiempos de modelación bajaron en aproximadamente un 40 %. Finalmente, en vista de estos resultados se entrega como recomendación al SHOA la compra de más memoria RAM para mejorar tiempos de modelación y aumentar el número de simulaciones en paralelo.
  - Sistema de archivo: Docker por defecto ocupa en Centos *devicemapper* como controlador de almacenamiento. De acuerdo a la documentación de Docker se cambia a *overlay2*, lo que provee un mejor rendimiento.
  - Espacio de almacenamiento: Las pruebas realizadas con distintas formas de almacenamiento compartido plantean un trabajo futuro de investigación sobre alternativas que

resulten óptimas para el proyecto. La opción que se implementó por razones de rendimiento fue que cada servidor cuente con un espacio de almacenamiento no compartido, así cada servidor será capaz de trabajar con solo 1 TB, pero con un mejor desempeño.

- **Comparar resultados de tiempos de simulación entre máquinas virtuales y el sistema propuesto:** Las resultados obtenidos notan mejoras en el rendimiento del sistema (*speedup*) de acuerdo al cuadro 6.1:

Tabla 6.1: Speedup de simulaciones en un servidor. Fuente: Elaboración propia.

	Máquinas virtuales	**Contenedores**	**Speedup**
Una simulación	27 Horas promedio	15 Horas promedio	1.8x
Veinte simulaciones paralelas	30 Horas promedio	17 Horas promedio	1.76x

Estos resultados de simulaciones se encuentran distribuidos en tiempos desde 13 hasta 20 horas. Se pronostica que luego de la instalación de RAM adicional en cada servidor, estos tiempos se van a ir acercando más a 13 horas. Lo anterior es gracias que al contar con mas RAM, se reduce el uso de memoria de intercambio, la cual ralentiza las modelaciones.

6.3. Trabajo Futuro

Al concluir esta memoria se plantean lineamientos sobre lo que es posible establecer una pauta de trabajo para posibles continuaciones:

- Almacenamiento: un punto interesante de tratar es establecer una evaluación de formas de almacenamiento óptimas para modelaciones en el proyecto SIPAT. Dentro de esto a primera vista existen conceptos como CEPH, versiones más actuales de GlusterFS, Lustre, LizardFS, entre otros. Además de estudiar una posibilidad de tener como opción un manejo mediante volúmenes lógicos a partir del mismo hardware storage.

- Automatización: Durante el desarrollo de la memoria se notó que la modificación de un Dockerfile para armar un contenedor es sencilla, pero al crear un contenedor a partir del *Dockerfile* se genera un tiempo de espera que se puede optimizar. Con ésto se plantea como posible trabajo futuro la automatización de la creación de contenedores, pruebas de validación

y distribución hacia el *Registry Local* mediante algunas herramientas de integración continua (CI), como por ejemplo *Jenkins*[74].

- Infraestructura: En el último tiempo ha surguido el concepto de infraestructura como código, lo que apunta a automatizar el despliegue de servidores y servicios mediante herramientas como *Puppet*[75] o *Chef*[76]. Sería recomendable que los sistemas de SIPAT estén automatizados haciendo uso de estas herramientas.

[74] https://jenkins.io/
[75] https://puppet.com/
[76] https://www.chef.io/chef/

Bibliografía

[1] Wilson A Acero, Jorge A Aguilar, and R David Mejía. Cluster de alta disponibilidad para virtualizar los servidores de adquisición y procesamiento de datos del instituto geofísico, April 2016. 3.3.1

[2] Theodora Adufu, Jieun Choi, and Yoonhee Kim. Is container-based technology a winner for high performance scientific applications? In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 507–510. IEEE, 2015. 2.3, 3.2.1

[3] François Beauducel. Okada: Surface deformation due to a finite rectangular source - file exchange - matlab central. https://www.mathworks.com/matlabcentral/fileexchange/25982-okada--surface-deformation-due-to-a-finite-rectangular-source?requestedDomain=www.mathworks.com, May 28 2014. (Accessed on 08/06/2017). 3.2.2.1

[4] Sean J Deal. Hpc made easy: Using docker to distribute and test trilinos, 2016. 2.2, 3.2.1

[5] Scott McCarty. Container tidbits: Understanding the docker-latest package – red hat enterprise linux blog. http://rhelblog.redhat.com/2016/10/31/understanding-docker-latest-package/. (Accedico el 03/07/2017). 4.2

[6] Daniel A Menascé. Virtualization: Concepts, applications, and performance modeling. In *Int. CMG Conference*, pages 407–414, 2005. 2.1.2

[7] Samuel Miranda González. Optimización e integración de un sistema de alerta y prevención de tsunami, sipat. Memoria, Universidad Técnica Federico Santa María, 2015. 1.1.2

[8] Leonardo Pizarro Baeza. Implementación del sistema integrado de predicción y alertas de tsunami, sipat. Memoria, Universidad Técnica Federico Santa María, 2014. 1.1.2, 3.2.2

[9] EN Preeth, Fr Jaison Paul Mulerickal, Biju Paul, and Yedhu Sastri. Evaluation of docker containers based on hardware utilization. In *Control Communication & Computing India (ICCC), 2015 International Conference on*, pages 697–700. IEEE, 2015. 2.2.2, 3.2.1

[10] Gabriel Alejandro Ruiz Miranda. Diseño de arquitectura de software para un sistema integrado de predicción y alerta de tsunamis. Memoria, Universidad Técnica Federico Santa María, 2013. 1.1.2

[11] Amit Singh. An introduction to virtualization. *kernelthread. com, January*, 2004. 2.1

[12] Carlos Valdés Flores. Sistema integrado de visualización de eventos tsunamigénicos. Memoria, Universidad Técnica Federico Santa María, 2015. 1.1.2

[13] Chenxi Wang. What is docker? linux containers explained | infoworld. http://www.infoworld.com/article/3204171/linux/what-is-docker-linux-containers-explained.html, June 29, 2017. (Accessed on 08/06/2017). 3.2.1

[14] Xiaoming Wang. User manual for comcot version 1.7 (first draft), cornel university, page 65, 2009. 3.2.2.1

[15] Diego Yachan Galarce. Módulo de soporte de decisiones para sistema integrado de predicción y alerta de tsunamis. Memoria, Universidad Técnica Federico Santa María, 2014. 1.1.2

A | Anexos

A.1. Instalación de docker-distribution y registry

A continuación se muestran los pasos de instalación de *docker-distribution* que es necesario para tener el *docker registry*

1. Instalar desde repositorios de Centos.

```
yum install docker-distribution
```

2. Abrir puertos para conexión desde otros servidores SIPAT.

```
firewall-cmd --permanent --add-port=5000/tcp
firewall-cmd --reload
```

3. Inicio del servicio y arranque automático al *boot*.

```
systemctl start docker-registry
systemctl enable docker-registry
```

4. Para el acceso desde servidores SIPAT es necesario modificar el archivo */etc/sysconfig/docker* agregando las lineas:

```
ADD_REGISTRY='--add-registry docker_registry:5000'
INSECURE_REGISTRY='--insecure-registry docker_registry:5000'
```

5. Luego reiniciar docker:

```
systemctl restart docker
```

Luego de la configuración anterior se tiene un *docker registry* local y se pueden ejecutar operaciones como *pull* y *push* como se describe en https://github.com/docker/docker.github.io/blob/master/registry/index.md

A.2. Plan de mantenimiento

Se plantea una serie de recomendaciones a fin de garantizar el correcto funcionamiento de la plataforma con el paso del tiempo:

- Realizar un monitoreo activo de la infraestructura mediante la herramienta Nagios [77].
- Revisar periódicamente actualizaciones de seguridad y nuevas funcionalidades en los sistemas operativos base de los servidores.
- Actualizar Rancher a las nuevas versiones manteniendo el tag *stable*.
- Realizar respaldos de la base de datos de Rancher.
- Actualizar periódicamente Stacks utilizados en Rancher.
- Actualizar periódicamente las imágenes base necesarias para modelamiento.
- Revisar logs tanto de los sistemas operativos de los servidores, como también de los contenedores Docker.
- Realizar mantenimiento físico de servidores consistente en limpieza de componentes mécanicos.
- Mantener registros de compra y garantías de hardware para tomar medidas preventivas cuando éstas finalicen.

A.3. Tiempos de modelamiento expresado en horas

[77] https://www.nagios.org/

Tabla A.1: Tiempos de modelación por sismos utilizando todos los servidores disponibles. Fuente: Elaboración propia.

ID sismo	Tiempo de modelación [Horas]
8500087	17.96
8500086	21.36
8500085	18.71
8500084	18.3
8500083	19
8500082	20.05
8500081	21.56
8500080	19.54
8500079	20.02
8500078	20.02
8500077	19.56
8500076	20.41
8500075	18.69
8500074	19.43
8500073	16.97
8500072	19.31
8500071	18.53
8500070	18.4
8500069	16.71
8500068	18.34
8500067	13.93
8500066	12.86
8500065	21.47
8500064	17.96
8500063	16.08

Continúa en la siguiente página.

Tabla A.1 – continúa de la página anterior.

ID sismo	Tiempo de modelación [Horas]
8500062	16.31
8500061	20.28
8500060	17.69
8500059	17.19
8500058	17.92
8500057	18.13
8500056	18.44
8500055	12.05
8500054	14.23
8500053	18.33
8500052	18.45
8500051	13.11
8500050	17.49
8500049	18.6
8500048	13.87
8500047	11.73
8500046	12.1
8500045	11.72
8500044	13.59
8500043	12.88
8500042	18.44
8500041	17.88
8500040	17.98
8500039	12.11
8500038	13.51
8500037	17.84

Continúa en la siguiente página.

Tabla A.1 – continúa de la página anterior.

ID sismo	Tiempo de modelación [Horas]
8500036	13.94
8500035	11.77
8500034	12.54
8500033	14.11
8500032	18.23
8500031	12.57
8500030	12.44
8500029	12.3
8500028	12.38
8500027	12.66
8500026	11.67
8500025	13.95
8500024	11.55
8500023	11.91
8500022	11.56
8500021	18.61
8500020	11.7
8500019	12.79
8500018	12.11
8500017	11.81
8500016	11.85
8500015	12.09
8500014	12.67
8500013	11.85
8500012	14.17
8500011	17.82

Continúa en la siguiente página.

Tabla A.1 – continúa de la página anterior.

ID sismo	Tiempo de modelación [Horas]
8500010	14.3
8500009	11.59
8500008	14.13
8500007	12.76
8500006	12.17
8500005	17.71
8500004	12.36
8500003	12.62
8500002	12.59
8500001	12.08
8500000	11.56
Promedio	15.47
Desviación Estándar	3.2