Proyecto de curso Experimental. Containerización del ecosistema SDV_UN_ROS

Andrés Holguín Restrepo
Ingeniería Mecatrónica
Universidad Nacional de Colombia
Bogotá, Colombia
aholguinr@unal.edu.co

***Resumen*—En el desarrollo de este informe se va a presentar todo el proceso ejecutado para lograr llegar a los resultados del proyecto de curso experimental.**

I. Introducción

En este proyecto, nos enfocaremos en la containerización de un repositorio de SDV_UN_ROS utilizando Docker. SDV_UN_ROS es un conjunto de software y herramientas relacionadas con vehículos autónomos y robótica, y nuestro objetivo es crear un entorno aislado y portable para su funcionamiento en diversos sistemas operativos.

Para lograr esto, utilizaremos Docker, una plataforma de código abierto que permite empaquetar aplicaciones y sus dependencias en contenedores. Estos contenedores son unidades livianas y portables que pueden ejecutarse en cualquier entorno que tenga Docker instalado, lo que nos brinda una gran flexibilidad y portabilidad.

En este caso, estaremos trabajando con un repositorio de SDV_UN_ROS que ha sido desarrollado en un entorno Ubuntu 18.04. Utilizaremos Docker para crear una imagen que contendrá todas las dependencias y configuraciones necesarias para ejecutar el repositorio de manera aislada.

Una vez que hayamos creado la imagen de Docker, podremos utilizarla para ejecutar el repositorio SDV_UN_ROS en diversos sistemas operativos, sin preocuparnos por las diferencias en las configuraciones o las dependencias del sistema. Esto nos permitirá simplificar el proceso de configuración y despliegue, y asegurarnos de que el repositorio funcione de manera consistente en diferentes entornos.

En resumen, este proyecto se centrará en la containerización del repositorio SDV_UN_ROS utilizando Docker, lo que nos permitirá crear un entorno aislado y portable para su funcionamiento en diversos sistemas operativos. Esta solución nos brindará flexibilidad, portabilidad y consistencia en el despliegue del repositorio, facilitando así su utilización y desarrollo en diferentes entornos.

II. KPIs del proyecto

Lo primero a identificar en el proyecto son sus KPIs, ya que partiendo de esto se puede entender el verdadero espectro del mismo. Estos KPI se definen a continuación:

- Sistemas operativos compatibles con los SDV-UN. 2 distros de Ubuntu y Windows 10.
- Tiempo promedio para configurar y utilizar los SDV en el entorno del contenedor. Sin tener en cuenta tiempos de descarga, 30 minutos.
- Número de usuarios del laboratorio LabFabEx que utilizan los SDV-UN. Sin interrumpir comunicaciones con el SDV-UN, 3 usuarios pueden conectarse al SDV-UN.
- Porcentaje de replicación exitosa de la ejecución de archivos fuente en el laboratorio LabFabEx. 75 % de replicación exitosa.
- Porcentaje de éxito en las pruebas de compatibilidad con diferentes computadores y sistemas operativos. 75 % de éxito.
- Número de fallos en las comunicaciones entre el contenedor y el SDV y tiempo promedio de solución. Disminuir los fallos a máximo uno por operación, y solución en un tiempo estimado de 15 minutos.
- Nivel de seguridad del sistema de monitoreo y registro de actividades y comunicaciones del contenedor y los SDV-UN. Llevar un registro de uso de contenedores para uso de SDV-UN.

III. FMECA

Ahora bien, un paso importante para el desarrollo del proyecto es realizar un análisis FMECA para identificar las posibilidades de falla del proyecto y así preveerlas mediante la incorporación de metodologías DIPP. De este modo, se presentan los principales modos de falla identificados:

1. El primer modo de fallo se refiere a la incompatibilidad del contenedor para ejecutar los archivos del proyecto SDV_UN_ROS debido a la versión no compatible de ROS melodic o a errores en la instalación del contenedor. Esto compromete la funcionalidad del proyecto y la capacidad de controlar adecuadamente los SDV. La severidad se considera alta, ya que afecta directamente la funcionalidad del sistema. La probabilidad de ocurrencia se califica como media, y la detectabilidad es alta, ya que el fallo se detecta rápidamente por la falta de control de los SDV. Para prevenir este fallo, se recomienda revisar la compatibilidad de ROS melodic con el contenedor antes de la instalación y realizar pruebas exhaustivas. En caso de ocurrir, se deben reinstalar una versión compatible de ROS melodic o corregir la instalación del contenedor.

2. El segundo modo de fallo implica la necesidad frecuente de reiniciar el contenedor debido a errores en la ejecución de los archivos del proyecto SDV_UN_ROS. Esto interrumpe la funcionalidad del proyecto y puede ocasionar pérdida de datos. Las causas pueden estar relacionadas con errores en los archivos del proyecto o problemas de compatibilidad con ROS melodic. La severidad y la probabilidad de ocurrencia se consideran medias, mientras que la detectabilidad es alta debido a la falta de control de los SDV. Para prevenir este fallo, se recomienda realizar pruebas exhaustivas antes de la implementación del contenedor y establecer una estrategia de control de versiones para los archivos del proyecto. En caso de fallo, se sugiere reiniciar el contenedor o volver a una versión anterior de los archivos del proyecto.
3. El tercer modo de fallo es la falta de actualizaciones de seguridad en el contenedor. Esto puede resultar en vulnerabilidades de seguridad que podrían ser explotadas por atacantes externos o internos. La causa principal es la falta de actualización del sistema operativo base y los paquetes de software. La severidad se considera alta, mientras que la probabilidad de ocurrencia es baja y la detectabilidad es baja, ya que el fallo podría no ser detectado hasta que se produzca una violación de seguridad. Para prevenir este fallo, se recomienda implementar una política de actualización regular del sistema operativo base y los paquetes de software, y realizar las actualizaciones necesarias tan pronto como se descubra una vulnerabilidad.
4. El último modo de fallo se refiere a la incapacidad de monitorizar adecuadamente el contenedor por parte de los desarrolladores del proyecto. Esto impide detectar y corregir errores en el contenedor. Las causas pueden ser la falta de acceso al contenedor o problemas de configuración. La severidad se considera media, la probabilidad de ocurrencia es baja y la detectabilidad es media, ya que el fallo podría no ser detectado hasta que se produzca un problema en el contenedor. Para prevenir este fallo, se sugiere proporcionar acceso adecuado al contenedor a los desarrolladores del proyecto, implementar una estrategia de monitoreo y registro de actividades y comunicaciones en el contenedor, y revisar y actualizar la configuración del contenedor según sea necesario.

Estos se pueden resumir fácilmente en la tabla I

Modo de fallo	Severidad	Probabilidad de ocurrencia
1	Alta	Media
2	Media	Media
3	Alta	Baja
4	Media	Baja

Tabla I
ANÁLISIS FMECA

IV. DOCKER

En este punto, es importante conocer un poco qué hace Docker.

En la era de la Industria 4.0, donde la digitalización y la automatización son fundamentales, Docker se ha convertido en una herramienta fundamental para el desarrollo y despliegue de aplicaciones en entornos empresariales. Docker es una plataforma de contenerización que permite empaquetar aplicaciones y sus dependencias en contenedores ligeros y portables.

La contenerización es un enfoque revolucionario para el desarrollo de software, ya que permite encapsular una aplicación y todas sus dependencias en un entorno aislado y autónomo, independiente del sistema operativo subyacente. Esto significa que una aplicación de Docker puede ejecutarse de manera consistente y sin problemas en cualquier infraestructura que tenga Docker instalado, ya sea un servidor local, una máquina virtual en la nube o incluso dispositivos IoT en el entorno industrial.

En la Industria 4.0, donde la flexibilidad, la escalabilidad y la eficiencia son primordiales, Docker ofrece numerosas ventajas. A continuación, se presentan algunas de las principales ventajas que Docker proporciona en el contexto de la Industria 4.0:

- Portabilidad y compatibilidad: Los contenedores Docker son autónomos y portables, lo que significa que se pueden ejecutar en cualquier sistema operativo que tenga Docker instalado. Esto facilita la migración y el despliegue de aplicaciones en diferentes entornos industriales, sin preocuparse por las diferencias en los sistemas operativos o las configuraciones de hardware.
- Aislamiento y seguridad: Los contenedores Docker ofrecen un alto nivel de aislamiento entre las aplicaciones y el sistema operativo subyacente. Esto ayuda a evitar conflictos entre diferentes componentes del sistema y proporciona una capa adicional de seguridad, ya que un contenedor comprometido no afectará al resto del sistema.
- Escalabilidad y flexibilidad: Docker permite escalar aplicaciones de manera fácil y eficiente, ya sea horizontalmente agregando más contenedores idénticos o verticalmente ajustando los recursos asignados a un contenedor. Esto permite adaptarse a las demandas cambiantes de la Industria 4.0 y maximizar la eficiencia del sistema.
- Despliegue rápido y consistente: Gracias a la contenerización, el despliegue de aplicaciones se vuelve rápido, predecible y consistente. Docker permite empaquetar todas las dependencias de una aplicación en un contenedor, lo que elimina los problemas de compatibilidad y simplifica el proceso de implementación en diferentes entornos.
- DevOps y colaboración: Docker fomenta una cultura de DevOps al proporcionar un entorno estandarizado y reproducible para el desarrollo y despliegue de aplicaciones. Esto facilita la colaboración entre los equipos de desarrollo y operaciones, acelerando los ciclos de entrega y mejorando la eficiencia general del proceso de desarrollo de software.

En resumen, Docker ha demostrado ser una herramienta

esencial en la Industria 4.0, gracias a su capacidad para proporcionar portabilidad, aislamiento, escalabilidad y una forma eficiente de implementar aplicaciones en diferentes entornos. Al aprovechar las ventajas de Docker, las empresas pueden mejorar su agilidad, eficiencia y seguridad en el desarrollo y despliegue de aplicaciones, impulsando así su éxito en la era de la Industria 4.0.

V. Mapa mental

Con todo y lo anterior, se realiza a modo de condensación de conceptos y de relación entre estos, un mapa mental capaz de conectar las ideas principales del proyecto, este se puede ver en la figura 1

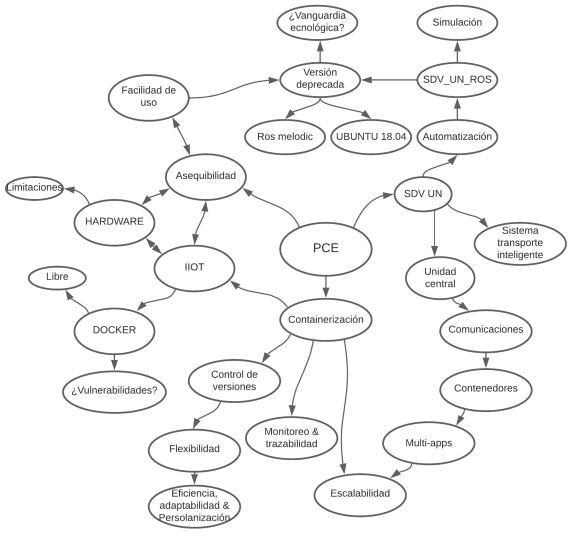

Figura 1. Mapa mental de desarrollo del PCEx

V-A. *Cronograma*

En la figura 2 sepuede evidenciar el cronograma de activiades propuesto inicialmente.

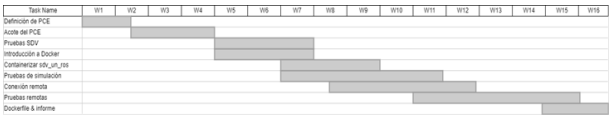

Figura 2. Cronograma de actividades

Debido a complicaciones ocurridas en el transcurso del semestre como la anormalidad académica, el corrimiento del calendario académico, indisponibilidad del laboratorio, e inicio de pasantía internacional, no se logra cumplir a cabalidad el seguimiento de este calendario, motivo por el cual se firma el OTRO SÍ correspondiente.

VI. Desarrollo de solución

Ahora bien, para realizar el desarrollo del proyecto se emplea la escritura de un Dockerfile capaz de generar una imagen desde Docker con todos los paquetes requeridos para que el contenedor creado opere adecuadamente. Es importante aclarar que uno de los entregables del proyecto consta del manual de uso y guía de instalación de todos los requisitos de software, lo cual es clave para la comprensión del proyecto, sin embargo no se va a profundizar en ese documento ya que el lector puede hacer lectura de este en los entregables del proyecto.

De este modo, el Dockerfile proporcionado describe los pasos necesarios para construir una imagen de Docker basada en la imagen .ºsrf/ros:melodic-desktop-fullz configurada específicamente para el proyecto SDV_UN_ROS. A continuación, se detalla cada una de las instrucciones principales del Dockerfile generado:

- `FROM osrf/ros:melodic-desktop-full`: Establece la imagen base para la construcción de la imagen de Docker. En este caso, se utiliza una imagen de ROS Melodic Desktop completa como punto de partida.
- `COPY catkin_ws /∼/catkin_ws`: Copia el contenido del directorio çatkin_ws"local al directorio /∼/catkin_ws"dentro del contenedor. Presumiblemente, el directorio çatkin_wsçontiene el código fuente y los archivos relacionados con el proyecto SDV_UN_ROS.
- `RUN apt-get update && apt-get upgrade -y`: Actualiza el sistema operativo dentro del contenedor ejecutando los comandos `apt-get update` y `apt-get upgrade -y`.
- `RUN sudo apt-get install ros-melodic-serial`: Instala el paquete ros-melodic-serial.[en] el contenedor, que es una dependencia para trabajar con comunicación serie en ROS Melodic.
- `RUN apt-get install -y ros-melodic-catkin`: Instala el paquete ros-melodic-catkin.[en] el contenedor, que proporciona herramientas y comandos para trabajar con catkin, el sistema de compilación de ROS.
- `RUN apt-get install tmux -y`: Instala el paquete "tmux.[en] el contenedor, que es una herramienta para gestionar sesiones y ventanas de terminal.
- `RUN apt-get install mesa-utils -y`: Instala el paquete "mesa-utils.[en] el contenedor, que contiene utilidades relacionadas con el controlador de gráficos Mesa.
- `RUN apt install -y xauth`: Instala el paquete "xauth.[en] el contenedor, que permite la autenticación en el servidor X.
- `RUN sudo apt-get install vim-gtk -y`: Instala el paquete "vim-gtk.[en] el contenedor, que es una versión de Vim con soporte para el entorno gráfico GTK.
- `RUN apt-get install -y x11-apps`: Instala el paquete "x11-apps.[en] el contenedor, que contiene aplicaciones y utilidades relacionadas con el sistema de ventanas X.
- `RUN sudo mkdir -p /run/user/0` y `RUN sudo chmod 0700 /run/user/0`: Crea un

directorio /run/user/0ᴢ le otorga permisos de lectura, escritura y ejecución al usuario root. Esto es necesario para configurar el entorno de ejecución XDG_RUNTIME_DIR.

- `RUN sudo apt update -y` y `RUN sudo apt install git -y`: Actualiza los repositorios de paquetes y luego instala el paquete "git.[en] el contenedor, que es un sistema de control de versiones ampliamente utilizado.
- `RUN sudo apt-get update` y `RUN apt-get install -y python-rosinstall python-rosinstall-generator python-wstool build-essential python-catkin-tools python3-vcstool`: Actualiza los repositorios de paquetes y luego instala varios paquetes relacionados con ROS y catkin que son necesarios para el proyecto SDV_UN_ROS.
- `RUN sudo sudo apt-get install python3-catkin-tools -y`: Instala el paquete "python3-catkin-tools.[en] el contenedor, que proporciona herramientas adicionales para trabajar con catkin en Python 3.
- `RUN sudo apt-get update && apt-get install -y -qq -no-install-recommends libglvnd0 libgl1 libglx0 libegl1 libxext6 libx11-6`: Actualiza el sistema e instala varias bibliotecas gráficas necesarias para el funcionamiento adecuado de aplicaciones y herramientas relacionadas con OpenGL y X11.
- `RUN sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool build-essential`: Instala varios paquetes y dependencias adicionales relacionadas con ROS y catkin.
- `RUN sudo apt install ros-melodic-gazebo-ros ros-melodic-gazebo-plugins ros-melodic-move-base ros-melodic-amcl ros-melodic-joint-state-publisher ros-melodic-rosbridge-suite ros-melodic-map-server ros-melodic-hector-slam ros-melodic-robot-state-publisher -y`: Instala varios paquetes de ROS necesarios para el funcionamiento del proyecto SDV_UN_ROS, como Gazebo, Move Base, AMCL, etc.
- `RUN rosdep update`: Actualiza las dependencias de ROS utilizando rosdep.
- `RUN echo 'export DISPLAY=host.docker.interr» /.bashrc, echo 'export LIBGL_ALWAYS_INDIRECT=' » /.bashrc, echo 'export XDG_RUNTIME_DIR=/run/user/$(id -u)'» /.bashrc, echo "source /opt/ros/melodic/setup.bash»> /.bashrc, echo "source /∼/catkin_ws/devel/setup.bash»> /.bashrc`: Estas instrucciones agregan líneas específicas al archivo -/.bashrc"dentro del contenedor, que establecen variables de entorno y fuentes para la configuración adecuada del entorno ROS y la conexión con el servidor X.
- `RUN /bin/bash -c "source /.bashrc"`: Ejecuta el archivo -/.bashrc"para cargar las variables de entorno y las fuentes dentro del contenedor.
- `WORKDIR /∼`: Establece el directorio de trabajo actual dentro del contenedor como /∼".
- `CMD ["bash"]`: Define el comando por defecto que se ejecutará al iniciar el contenedor, en este caso, se ejecutará el intérprete de comandos bash.

Ahora bien, el usuario final no debe usar el Dockerfile. Este Dockerfile crea una imagen de Docker la cual es subida a Docker Hub de manera pública, tal que cualquier persona tenga acceso a ella.

VII. Simulación

Ahora, ya teniendo esta imagen creada, se pueden realizar las simulaciones correspondientes.

Debido a que solo se ejecutan los programas desde una terminal, se tiene instalado el paquete tmux que permite ejecutar varias terminales al mismo tiempo. Se sugiere ver la siguiente guía: https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/ En caso de no tener experiencia con estalibrería.

VII-A. Simulación rápida

La simulación principal inicia una instancia de Gazebo con paredes de LabFabEx y el robot móvil SDVUN1 con la navegación de la pila AMCL. También llama a RViz, que permite enviar comandos de posición y ver todos los mapas y poses de planificación. Para ejecutar esta simulación, se debe ejecutar el siguiente comando:

```
roslaunch sdvun_gazebo sdvun_nav.launch rviz:=true
```

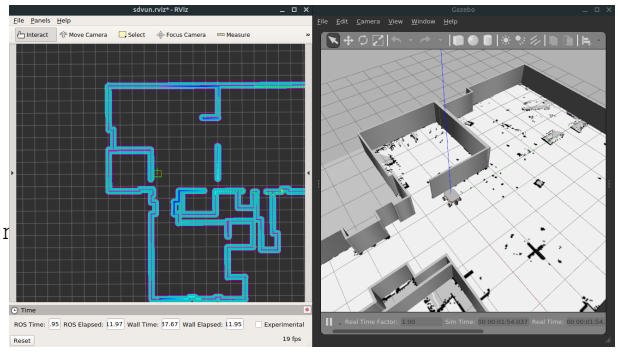

Figura 3. GUI de simulación rápida

VII-B. *Simulación personalizada*

Si se desea ver el mundo LabFabEx en Gazebo sin ningún robot, se debe ejecutar el siguiente comando:

```
roslaunch labfabex_gazebo
labfabex_bringup.launch
```

Para generar un robot SDVUNX en Gazebo con nodos de la pila de navegación, todo dentro de un espacio de nombres, se debe ejecutar este comando:

```
roslaunch sdvun_gazebo
spawn_sdvun_nav.launch robot_model:=sdvun3
namespace:=sdvun3
```

Además, se puede utilizar argumentos como robot_model, localization y namespace para ajustar la simulación según los requisitos.

VII-C. *Mundo vacío*

Si se desea ver el robot móvil SDVUN3 en un mundo vacío, sin paredes, ejecuta esto:

```
roslaunch sdvun_gazebo
sdvun_empty_world.launch
robot_model:=sdvun3
```

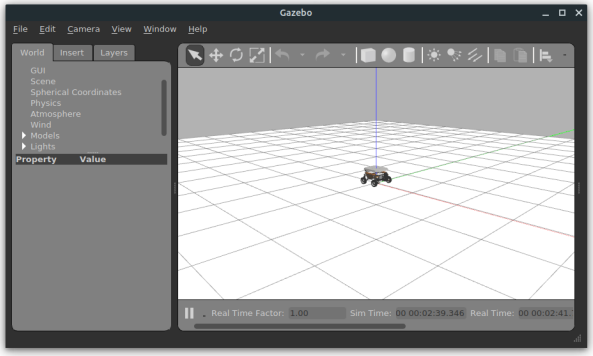

Figura 4. Empty World

VII-D. *Ver modelo en RViz*

Para ver un modelo 3D de SDVUNX en RViz, ejecuta este comando:

```
roslaunch sdvun_gazebo
sdvun_empty_world.launch
robot_model:=sdvun1
```

VIII. Dificultades

A partir de este desarrollo del proyecto, se llegaron a diversas dificultades que deben ser mencionadas para tener un mayor espectro de los alcances de este proyecto.

- Dificultad para generar el Dockerfile desde cero: Crear un Dockerfile completo y funcional puede ser desafiante, especialmente si no se tiene experiencia previa en la creación de imágenes de contenedores. Requiere conocimientos técnicos y comprensión de los requisitos y dependencias del proyecto.
- Operabilidad externa del equipo para el servidor X: Configurar la conexión con el servidor X puede ser

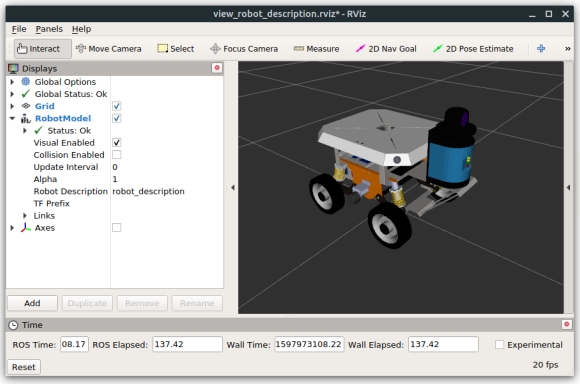

Figura 5. Modelo SDV en RVIZ

complicado, especialmente si se requiere acceder a la interfaz gráfica desde el contenedor. Es posible que se necesiten ajustes y configuraciones adicionales para permitir la visualización gráfica y el acceso adecuado a las aplicaciones y herramientas.
- Imposibilidad de implementar en la vida real debido a la indisponibilidad de espacios de laboratorio: En algunos casos, puede ser difícil acceder a los espacios de laboratorio necesarios para probar y desplegar el contenedor en un entorno real. Esto puede limitar las pruebas y validaciones del proyecto y retrasar su implementación efectiva.
- Dificultad debido a los altos tiempos de generación de cada imagen: La generación de imágenes de contenedor puede llevar tiempo, especialmente si hay muchas dependencias o si se requiere compilar el código fuente dentro del contenedor. Los largos tiempos de construcción pueden ralentizar el proceso de desarrollo y pruebas, lo que puede resultar en una mayor complejidad y demoras en el proyecto.

Estas dificultades son comunes al trabajar con Docker y la containerización en general. Requieren experiencia, resolución de problemas y paciencia para superar los obstáculos y lograr un entorno de contenedor funcional y eficiente.

IX. Conclusiones

Con todo y lo anterior, se puede generar conclusiones acordes al desarrollo del proyecto.

- El proceso de generar un Dockerfile desde cero puede ser desafiante, pero una vez superada esta dificultad, se obtiene un archivo que permite reproducir y distribuir de manera consistente un entorno de desarrollo específico. Esto facilita la colaboración entre equipos, reduce las inconsistencias y simplifica la configuración de múltiples sistemas.
- A pesar de la dificultad de lograr la operabilidad externa del equipo para el servidor X, el Dockerfile incluye las configuraciones necesarias para habilitar la visualización gráfica y permitir el acceso adecuado a las aplicaciones

y herramientas dentro del contenedor. Esto es especialmente útil en entornos que requieren interfaces gráficas para el desarrollo y la depuración de software.
- La imagen de Docker generada proporciona una base sólida para llevar el entorno de desarrollo a diferentes sistemas y entornos de producción. Esto permite una mayor flexibilidad y portabilidad en la implementación del proyecto en distintas infraestructuras.
- A pesar de los altos tiempos de generación de cada imagen, el Dockerfile asegura la correcta instalación y configuración de todas las dependencias necesarias para el proyecto. Esto simplifica el proceso de configuración y evita posibles problemas de compatibilidad y dependencias faltantes.

X. Trabajo a futuro

Para finalizar, es importante hacer mención del trabajo a futuro que se debe realizar, considerando los imprevistos que se obtuvieron en el transcurso del semestre. Se identifican varios trabajos a futuro para mejorar y optimizar el uso de los SDV-UN en el entorno del contenedor. Primero, es necesario realizar pruebas experimentales en diferentes sistemas operativos compatibles de computadores del laboratorio, como Ubuntu y Windows 10, para garantizar la compatibilidad y funcionalidad adecuada. Además, se debe trabajar en reducir el tiempo promedio requerido para configurar y utilizar los SDV en el entorno del contenedor, sin tener en cuenta los tiempos de descarga, a un objetivo de 30 minutos.

Otro aspecto importante es aumentar el número de usuarios del laboratorio LabFabEx que utilizan los SDV-UN, asegurando una conexión sin interrupciones con el SDV-UN. El objetivo es permitir que al menos 3 usuarios puedan conectarse simultáneamente al SDV-UN.

Además, se debe trabajar en mejorar el porcentaje de replicación exitosa de la ejecución de archivos fuente en el laboratorio LabFabEx, apuntando a un objetivo del 75 % de replicación exitosa. Esto implica realizar pruebas exhaustivas de compatibilidad con diferentes computadores y sistemas operativos, con el objetivo de lograr un alto porcentaje de éxito, también establecido en un 75 %.

Un punto crítico es la comunicación entre el contenedor y el SDV, donde se deben reducir los fallos a un máximo de uno por operación y garantizar una solución en un tiempo estimado de 15 minutos. Esto requiere un enfoque en la resolución rápida de problemas y la optimización de las comunicaciones.

Por último, se debe considerar el nivel de seguridad del sistema de monitoreo y registro de actividades y comunicaciones del contenedor y los SDV-UN. Es fundamental llevar un registro detallado del uso de los contenedores para el uso de los SDV-UN, asegurando la integridad y confidencialidad de los datos.

En resumen, se requiere un trabajo continuo y enfocado en mejorar la compatibilidad, la eficiencia, la seguridad y la capacidad de uso de los SDV-UN en el entorno del contenedor, con el objetivo de lograr un despliegue exitoso y una experiencia de usuario óptima en el laboratorio LabFabEx.

Referencias

[1] Docker. (2023). *docker push*. [Online]. Disponible en: https://docs.docker.com/engine/reference/commandline/push/. [Accedido el 27 de junio de 2023].

[2] Docker. (2023). *docker pull*. [Online]. Disponible en: https://docs.docker.com/engine/reference/commandline/pull/. [Accedido el 27 de junio de 2023].

[3] Docker. (2023). *docker exec*. [Online]. Disponible en: https://docs.docker.com/engine/reference/commandline/exec/. [Accedido el 27 de junio de 2023].

[4] Docker. (2023). *docker run*. [Online]. Disponible en: https://docs.docker.com/engine/reference/commandline/run/. [Accedido el 27 de junio de 2023].

[5] DigitalOcean. *How To Install and Use Docker on Ubuntu 16.04*. DigitalOcean Community Tutorials. [Online]. Disponible en: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-16-04. [Accedido el 27 de junio de 2023].

[6] Docker. (s.f.). *ROS*. En Docker Hub. Recuperado de https://hub.docker.com/_/ros/.