UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

TEMA:

"COMUNICACIÓN Y VIRTUALIZACIÓN DE PROCESOS INDUSTRIALES BASADOS EN INDUSTRIA 4.0"

Trabajo de Graduación. Modalidad: Proyecto de Investigación, presentado previo la obtención del título de Ingeniero en Electrónica y Comunicaciones.

LÍNEAS DE INVESTIGACIÓN: Tecnologías de Información y Sistemas de Control

AUTOR: Cushpa Telenchana Paulo David

TUTOR: Ing. Brito Moncayo Geovanni Danilo, Mg.

Ambato – Ecuador

Enero-2020

APROBACIÓN DEL TUTOR

En mi calidad de tutor del Trabajo de Investigación sobre el tema: "COMUNICACIÓN Y VIRTUALIZACION DE PROCESOS INDUSTRIALES BASADOS EN INDUSTRIA 4.0", del señor PAULO DAVID CUSHPA TELENCHANA, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial de la Universidad Técnica de Ambato, considero que el informe investigativo reúne los requisitos suficientes para que continúe con los tramites y consiguiente aprobación de conformidad con el numeral 7.2 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ambato enero, 2020

EL TUTOR

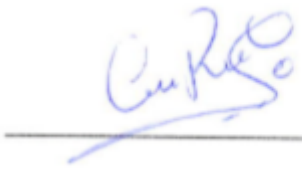
Ing. Geovanni Danilo Brito Moncayo, Mg

AUTORÍA DE TRABAJO

El presente Proyecto de Investigación titulado: "COMUNICACIÓN Y VIRTUALIZACION DE PROCESOS INDUSTRIALES BASADOS EN INDUSTRIA 4.0", es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato enero, 2020

AUTOR



Paulo David Cushpa Telenchana

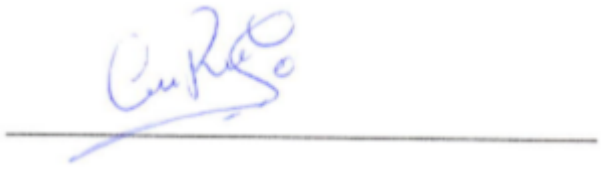
DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación, con fines de difusión pública, además autorizo su reproducción dentro de las regulaciones de la Universidad.

Ambato enero, 2020

AUTOR



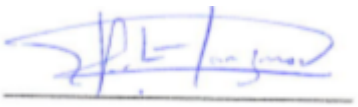
Paulo David Cushpa Telenchana

APROBACIÓN DEL TRIBUNAL DE GRADO

La Comisión Calificadora del presente trabajo conformada por los señores docentes Ing. Mg. Santiago Manzano e Ing. Mg. Patricio Cordova, revisó y aprobó el Informe Final del Proyecto de Investigación titulado "COMUNICACIÓN Y VIRTUALIZACIÓN DE PROCESOS INDUSTRIALES BASADOS EN INDUSTRIA 4.0", presentado por el señor Paulo David Cushpa Telenchana, de acuerdo al numeral 9.1 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ing. Mg. Elsa Pilar Urrutia Urrutia

PRESIDENTA DEL TRIBUNAL



Ing. Mg. Santiago Manzano

DOCENTE CALIFICADOR

Ing. Mg. Patricio Cordova

DOCENTE CALIFICADOR

DEDICATORIA

A mi Madre, quien con todo su amor y sus enseñanzas ha logrado cultivar en sus cuatro hijos un espíritu de superación, fomentando valores como la humildad, el respeto y la responsabilidad, una persona que ha sido capaz de lograr grandes cosas a lo largo de su vida, motivándome a seguir sus pasos y a culminar mi carrera de la manera más satisfactoria posible, gracias por su esfuerzo y sacrificio invaluable. A mi Padre que está en el cielo, que con sus consejos que quedaron grabados en mi memoria logre cumplir uno de sus anhelos más valiosos que tenía en su vida para con sus hijos.

A las personas más importantes en mi vida que son mi familia, por estar pendientes siempre de mí y apoyarme en todos los momentos, demostrándome sus actos de cariño y amor sincero.

Paulo David Cushpa Telenchana

AGRADECIMIENTO

Principalmente a Dios, por permitirme despertar apreciando un nuevo día, brindándome la oportunidad de vivir experiencias inigualables como hijo, hermano, amigo y ahora como profesional.

A mi madre, por el apoyo incondicional que siempre me ha brindado, por tener siempre la fortaleza para salir adelante sin importar los obstáculos, por los valores que me ha inculcado en mí, gracias por cada consejo y por ser la guía en mi camino para alcanzar mis metas. A mi familia y amigos por sus palabras de aliento y sus buenos deseos para cumplir mis objetivos.

Gracias a mi tutor Ing. Geovanni Brito, por compartir sus conocimientos, por su tiempo y paciencia en el trascurso del desarrollo del proyecto de investigación. Al Dr. Marcelo García, por presentarme un reto que supo llevarme a investigar campos que nunca me habría imaginado, gracias por el tiempo brindado.

Paulo David Cushpa Telenchana

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto de investigación tiene como finalidad el diseño y construcción de una arquitectura para la comunicación y virtualización de procesos industriales, El sistema está conformado por varios contenedores de software (Docker), cada uno con su sistema operativo virtualizado, y dentro de cada contenedor se ejecuta una aplicación Forte, esta tiene la configuración de los diferentes procesos creados para los movimientos del prototipo de brazo robótico, como también para la comunicación entre los diferentes dispositivos del sistema.

El diseño de esta arquitectura se desarrolló en una tarjeta Raspberry Pi3 B+, a partir del estándar IEC 61449, se crea un sistema distribuido, mediante una arquitectura de bloques funcionales (FB) con la ayuda del software 4DIAC, estos al ser compilados dan origen a una aplicación "forte", esta se ejecutara dentro de un contenedor de software que actúa independientemente de su sistema operativo con su principal función para la que fue creada, todos los contenedores como los dispositivos Arduino, la Raspberry Pi y un ordenador están compartiendo información mediante una conexión Ethernet por medio de un Router.

En la comunicación entre los diferentes contenedores y dispositivos conectados se utilizó los protocolos UDP, TCP/IP y MQTT (Message Queue Telemetry Transport), tanto para la recepción y envió de datos, al igual que una interfaz web, con la que se puede controlar la ejecución de los diferentes procesos, como para la visualización del sistema. Adicionalmente para la comprobación de la aplicación realizada, se construyó un prototipo de brazo robótico controlado con Arduino, quien recibe los datos desde la Raspberry Pi, al enviar un proceso designado por el operador desde la página web, se comprueba el funcionamiento de la aplicación.

Palabras clave: Virtualización, Raspberry Pi, MQTT, Docker, distribuido.

ABSTRACT

The purpose of this research project is to design and build an architecture for the communication and virtualization of industrial processes. The system consists of several software containers (Docker), each with its virtualized operating system, and within each container. A Forte application is executed, it has the configuration of the different processes created for the movements of the robotic arm prototype, as well as for the communication between the different devices of the system.

The design of this architecture was developed on a Raspberry Pi3 B + card, from the IEC 61449 standard, a distributed system is created, using a functional block architecture (FB) with the help of 4DIAC software, these being compiled give rise to a "forte" application, it will run inside a software container that acts independently of its operating system with its main function for which it was created, all containers such as Arduino devices, Raspberry Pi and a computer are sharing information through an Ethernet connection through a router.

In the communication between the different containers and connected devices the protocols UDP, TCP / IP and MQTT (Message Queue Telemetry Transport) were used, both for the reception and sending of data, as well as a web interface, with which you can control the execution of the different processes, as for the visualization of the system. Additionally, for the verification of the application made, a prototype robotic arm controlled with Arduino was built, who receives the data from the Raspberry Pi, when sending a process designated by the operator from the website, the operation of the application is checked.

Keywords: Virtualization, Raspberry Pi, MQTT, Docker, distributed.

GLOSARIO DE TÉRMINOS Y ACRÓNIMOS

ACI: App Container Image o Imagen del Contenedor

AMQP: Advanced Message Queuing Protocol o Protocolo avanzado de colas de mensajes

CHROOT: Comando que ejecuta un proceso bajo un directorio raíz simulado

CoAP: Constrained Application Protocol o protocolo de aplicación restringida

CONSOLA: Terminal físico

DDS: Data Distribution Service o Servicio de Distribución de datos

DOCKER: Proyecto de código abierto para la implementación de contenedores de

FRAMEWORK: Entorno genérico para escribir código en un lenguaje concreto software.

FB: Bloque Funcional

GO: Lenguaje de programación inspirado en la sintaxis de C

GPIO: General Purpose Input/Output o Entrada/Salida de Propósito General

GUI: Graphical User Interface o Interfaz Gráfica de Usuario

HTTP: Hypertext Transfer Protocol o Protocolo de transferencia de hipertexto

ICMP: Internet Control Message Protocol o protocolo de control de mensajes de Internet

IEC-61449: Estándar que define una arquitectura para aplicaciones reconfigurables de control distribuido

IFR: Federación Internacional de Robótica

INIT, SYSTEMD: Son administradores de sistemas y servicios asociados con sistemas Linux

IoT: Internet de las cosas

IP: Internet Protocol o Protocolo de Internet

ITK: Insight Segmentation and Registration Toolkit o Kit de herramientas de segmentación y registro

I2C: protocolo de comunicación mediante un bus con múltiples maestros

FREEBSD JAIL: implementación de virtualización a nivel de sistema operativo

KVM: Kernel-based Virtual Machine o máquina virtual basada en el núcleo

LXC: Linux Containers o Contenedores basados en Linux

LXD: Administrador de contenedores de software

MES: Manufacturing Execution System o sistema de ejecución de manufactura

Mbps: Megabit por segundo

MQTT: Message Queuing Telemetry Transport o Cola de mensajes telemetría y transporte

MTU: Maximum Transmission Unit o unidad máxima de transferencia

M2M: Machine to Machine o máquina a máquina

OCI: Open Container Initiative o Iniciativa de contenedores abiertos

OMG: Object Management Group o Grupo de Gestión de Objetos

OPEN SOURCE: código abierto

PARALLELS: Software que proporciona virtualización de hardware para ordenadores Macintosh

PLC: Programmable Logic Controller o controlador lógico programable

QUAY: Empresa dedicada a la Automatización de compilaciones de contenedores, con integración a GitHub, Bitbucket y más.

QUOTA: Espacio de disco asignado a un usuario.

RAM: Random Access Memory o M,moria de acceso aleatorio

RHEL: Red Hat Enterprise Linux

RISC: Reduced Instruction Set Computer o Computador con Conjunto de Instrucciones Reducidas

RKT: Motor de contenedores basado en estándares y con mentalidad de seguridad

ROOT: descripción de un usuario que tiene todos los derechos y privilegios necesarios para modificar todos los archivos y programas en el sistema operativo.

RUNC: Herramienta de gestión de contenedores de software

SBC: Single Board Computer

SCADA: Supervisry Control and Data Acquisition o supervisión, control y adquisición de datos

SDN: Software-Defined Networking o Redes definidas por software

SHELL: Intérprete de línea de comandos

SO: Sistema Operativo

SPI: Estándar de comunicaciones para la transferencia de información entre circuitos (Serial Peripheral Interface)

TCP: Transmission Control Protocol o Protocolo de Control de Transmisión

TERMINAL: Entorno de entrada y salida en texto

TTL: Transistor-Transistor Logic o Lógica transistor a transistor

TIC: Tecnologías de la Información y la Comunicación

TLS: Transport Layer Security o seguridad en la capa de transporte

UDP: User Datagram Protocol o Protocolo de datagramas de usuario

VoIP: Voice over IP o Voz sobre protocolo de internet

VPS: Virtual Private Server o Servidor Virtual Privado

VTK: Visualization ToolKit o Kit de herramientas de Visualización

WDS: wireless distribution System o Sistema de Distribución inalámbrico

3D: Visión Tridimensional

CAPÍTULO 1
MARCO TEÓRICO

1.1 Antecedentes Investigativos

El proceso de la investigación Bibliográfica, se realizó en diferentes bases de datos académicas internacionales, obteniendo información de proyectos realizados que sustentaron una ayuda para tema propuesto en el presente trabajo, esto debido a que en los repositorios de la Universidad Técnica de Ambato se encontró poca información sobre el tema tratado.

En el documento presentado por Lucas-Estañ *et al.* [1] proponen el uso de tecnologías de comunicación heterogéneas integradas en una arquitectura jerárquica de comunicaciones y gestión de datos donde las decisiones de gestión descentralizadas y locales son coordinadas por un orquestador central que asegura el funcionamiento global eficiente del sistema, aludiendo al paradigma Industria 4.0, como una nueva revolución industrial en la que las fábricas evolucionan hacia estructuras digitalizadas y en red donde la inteligencia se extiende entre los diferentes elementos de los sistemas de producción, obteniendo una comunicación y la administración de datos basados en una arquitectura flexible y confiable para poder cumplir de manera eficiente los requisitos rigurosos y variables en términos de latencia, confiabilidad y velocidades de datos que exigen las aplicaciones industriales, y con especial atención en la automatización de tiempo crítico.

J. Wan [2] propone un modo de producción reconfigurable impulsado por datos de Smart Factory para la fabricación de productos farmacéuticos. La arquitectura de la fábrica inteligente está compuesta por tres capas principales, a saber, capa de percepción, capa de despliegue y capa de ejecución. Una base de conocimiento basada en MASON (Manufacturing's Semantics Ontology) se introduce en la capa de percepción, que es responsable de la planificación del plan de producción farmacéutica. La investigación se basa sobre la Industria 4.0, y como afecta profundamente a la asistencia sanitaria, así como a otros sectores de producción

tradicionales, esto para dar cabida a la creciente demanda de agilidad, flexibilidad y bajo costo en el sector de la salud, Para una mayor funcionalidad de reconfiguración y control de bajo nivel, el estándar IEC 61499 también se presenta para el modelado de la funcionalidad y el control de la máquina.

En el documento presentado por Akiba *et al*. [3] presentan un diseño de una arquitectura para dispositivos de Internet de las cosas (IoT) para extender sus funciones, estas están virtualizadas y configuradas en un entorno informático. Esta arquitectura se basa en la cooperación de dispositivos IoT y sus extensiones implementadas en otros servidores en un entorno de computación en la nube a través de métodos de virtualización, esto ayudara a expandir las funciones del dispositivo en IoT, obteniendo una viabilidad de los sistemas de IoT en los que se implemente.

Ma *et al*. [4] proponen una infraestructura definida por software, que luego se implementa en un entorno de red Industria 4.0. Esta infraestructura mejorara las operaciones generales del entorno de red, la velocidad de transmisión de datos y la calidad de los servicios prestados. en donde destaca importancia de la industria 4.0 y su gran impacto en el comercio, ya sea en el sector manufacturero, donde las empresas producen muchos productos personalizados, y aquellos que producen algunos y diferentes productos, tienen la oportunidad de alcanzar los mismos estándares de producción y costos entre sí. En una fábrica inteligente, los trabajadores y operadores están cambiando de los operadores de máquinas tradicionales a los controladores de procesos de producción, y al personal que realiza ajustes en la administración y las decisiones, para optimizar los procesos de producción.

En el documento presentado por R. Morabito [5] destaca que la virtualización basada en contenedores, ofrece varias ventajas tales como alto rendimiento, eficiencia de recursos y entorno ágil, lo que puede facilitar la administración de dispositivos IoT. Aunque estudios previos introducen la virtualización basada en contenedores a los dispositivos IoT, no abordan los diferentes modos de red de los contenedores y los problemas de rendimiento. Debido a que el rendimiento de la red es un factor

importante en IoT, el análisis del rendimiento de la red de los contenedores es esencial.

Yiming *et al*. [6] realizan un análisis del rendimiento de datos en Internet of Things (IoT) del cual depende de los servicios de transporte efectivos ofrecidos por la red subyacente. Esto con este fin de implementar un prototipo funcional de nodo de computación en la nube basado en redes definidas por software (SDN). El Transporte de telemetría de Message Queue Server (MQTT) se elige el protocolo IoT candidato que transporta datos generados desde dispositivos IoT a un host remoto (denominado intermediario MQTT). Se Implementa las funcionalidades de intermediario MQTT integradas en los conmutadores de borde, esto servirá como plataforma para realizar análisis simples basados en mensajes en los conmutadores, y entregar mensajes de manera confiable al host final para el análisis posterior a la entrega.

1.1.1 Contextualización del problema

Una de las principales prioridades en grandes empresas del sector Industrial a nivel mundial, es aumentar la eficiencia, desde la parte administrativa hasta las plantas de producción, la eficacia del equipo operativo, la eficiencia de los procesos y una mejor rentabilidad sobre los activos con una menor inversión, constituyen las prioridades esenciales para los directivos de la empresa. Para alcanzar esta eficiencia, los líderes del sector industrial necesitan nuevas tecnologías que garanticen la visibilidad y la disponibilidad de los Sistemas de Control Industrial y eliminen por completo los tiempos de inactividad imprevistos. La tecnología moderna, concretamente la virtualización de procesos unida a la tolerancia de fallas mitiga los tiempos de inactividad, tanto planificados como imprevistos para propiciar un funcionamiento más inteligente de las plantas, y simplificando la gestión de los sistemas en general, obteniendo un ahorro de costes, una gestión simplificada y una mayor eficiencia [7].

Muchos de los países en América Latina aún carecen de una infraestructura robusta con tecnología moderna que pueda proporcionar las bases para una próspera industria y facilitar la recolección de datos, menos modificaciones para mantener las aplicaciones funcionando correctamente, y esto trae consigo comprar más equipos

para ampliar la producción, con una velocidad trasmisión de datos adecuada para el sistema, teniendo en cuenta que un fallo en el sistema, por pequeño que sea el tiempo en restaurarlo, causaría perdidas de dinero para la empresa [8].

La virtualización de procesos en la industria tiene cada vez mayor importancia y este tipo de técnicas ayudan a mejorar su eficiencia en el proceso. Actualmente, en el área industrial, los software de simulación son utilizados bien cuando se quiere implementar una nueva etapa o un nuevo proceso en el sistema actual, o bien cuando se decide realizar un estudio para mejorar el rendimiento de la planta. Esto implica una transformación de una empresa u organización en un potencial creciente en el ámbito tecnológico, pero con el objetivo fundamental de mejorar cada momento, consolidando una empresa única ante la competencia.

El uso de tecnologías de virtualización es una de las tendencias mundiales que ha venido abriéndose paso en el mundo y en América Latina. En términos de industria 4.0, esta evolución puede ser uno de los pasos más importantes en la región, y con una nueva revolución industrial que combina técnicas avanzadas de producción y operaciones con tecnologías inteligentes que se integrarán en las organizaciones, las personas y los activos, para tener una capacidad de ajustarse y aprender de los datos en tiempo real, haciendo que las organizaciones sean más receptivas, proactivas y predictivas, permitiendo a la organización reducir sus riesgos en materia de productividad, y tanto para los empleados puede significar un cambio en el trabajo que van a realizar, haciendo su control más fácil, rápido y seguro.

Desarrollar un sistema que mediante la comunicación y virtualización de procesos industriales permita traer beneficios a los diversos sectores de negocios, en donde se requiera tener una capacidad de división, ejecutando múltiples aplicaciones y sistemas operativos en un mismo sistema físico, teniendo los datos encapsulados, esto un entorno completo de contenedores de software, donde se guardara en un solo archivo, fácil de mover, copiar y proteger, recuperando la información y configuraciones en forma ágil y segura del servicio ante fallas, obteniendo múltiples beneficios en los que se pueden destacar, una mayor utilización de la capacidad de la infraestructura, reducción de costos en mantenimiento de infraestructura física,

reducción de los costos de personal y tiempos de ejecución de tareas de mantenimiento, recuperación y protección de desastres, mejoramiento en la administración y seguridad de equipos de escritorio.

1.1.2 Fundamentación Teórica

Industria 4.0

El concepto industria 4.0 nace en Hannover Alemania hacia el 2011, con el objetivo de promover la manufactura computarizada impulsada por el asombroso aumento de la capacidad computacional y de conectividad, la aparición de nuevas capacidades analíticas y de inteligencia empresarial, nuevas formas de interacción hombre-máquina, como las pantallas táctiles y sistemas de realidad aumentada, además de notables mejoras en la transferencia de instrucciones digitales al mundo físico como la robótica y la impresión 3D [9].

Entre sus principales impulsores en la industria 4.0 son el Internet de las cosas, el Internet Industrial, la fabricación basada en la nube y la manufactura inteligente, entre otras tecnologías que comparten la visión de una futura producción digitalizada. Dichas tecnologías ahora se encuentran en un punto en el que por su mayor confiabilidad y menor costo están comenzando a tener sentido para las aplicaciones industriales [9].

Figura 1 Tecnologías que abarca la industria 4.0 [10] .

Virtualización

Normalmente, cuando se utiliza una computadora, el Sistema Operativo (SO) es instalado y ejecutado directamente sobre el hardware y de esta forma se aprovecha de forma completa su potencial. Sin embargo, existe un tipo de programa llamado software de virtualización, o máquina virtual, encargada de emular un hardware determinado, aprovechando los recursos reales de la computadora y, sobre este software, es posible instalar un sistema operativo como si se tratase de una computadora real. Este huésped funciona, a grandes rasgos, como si fuera un Sistema Operativo principal, por lo que todas sus funciones y características están disponibles, convirtiéndolo en una herramienta ideal para realizar pruebas [11].

Cuando hablamos de virtualización hacemos referencia al proceso de reemplazar dispositivos físicos por dispositivos virtuales, disponibles mediante el uso de un software. Se pueden virtualizar servidores, estaciones de trabajo, redes y aplicaciones, para ello, el software de virtualización administra los recursos físicos de esa máquina: memoria, CPU, almacenamiento y ancho de banda de la red, entre los aspectos más relevantes. Junto con la disminución del uso de equipos físicos, la virtualización trae como beneficios la reducción de costos de mantenimiento y consumo energético. Esto deriva una consolidación de servidores, optimizando el uso del espacio físico [11].

Virtualización basada en Sistema Operativo

La virtualización basada en sistemas operativos, es un método de particionado por el cual el kernel crea diferentes entornos o dominios de ejecución (Contenedores, Prisiones, Zonas) en espacio de usuario para ejecutar grupos de procesos, en lugar de un único entorno como es lo habitual, así todos los entornos o dominios comparten el mismo sistema operativo, se puede apreciar en la figura 2 [12].

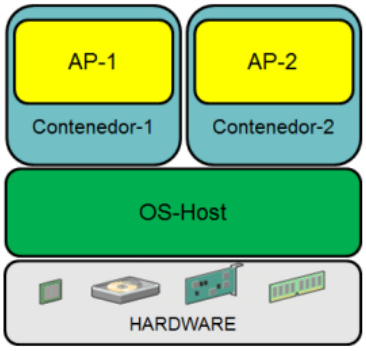


Figura 2 Virtualización basada en Sistema Operativo [12].

Los sistemas operativos ofrecen una Interfaz de Programación de Aplicaciones (API: Application Programming Interfaces) y llamadas al sistema (System Calls o Syscalls) utilizadas por las aplicaciones para solicitar servicios. La virtualización basada en sistemas operativos consiste en interceptar las llamadas al sistema y alterar su comportamiento para simularle a la aplicación un entorno específico. El origen de esta tecnología parte de los entornos chroot, en donde a un proceso, a todos sus hijos y a toda su descendencia se les asigna un pseudo Sistema de Archivos root que es mapeado a un subdirectorio del sistema de archivos root real, de esta forma los procesos que ejecutan en este entorno (conocido como prisión chroot) no disponen de acceso a archivos y directorios que están por fuera de su sistema de archivos root en el árbol de directorios (aislamiento de Sistema de Archivos) [12] se conocen tecnologías para la virtualización de sistemas operativos entre ellas los contenedores de software y máquinas virtuales, que son las utilizadas en la actualidad.

Contenedores de Software

En los sistemas unix-like como Linux, la virtualización a nivel del sistema operativo se basa en una implementación ampliada de mecanismos Chroot nativos. Chroot equivale a “change root”, un comando que se utiliza en sistemas operativos unix-like para modificar el directorio Root y los procesos hijo de un proceso en ejecución. A esto se añade que contenedores como Docker, rkt, LXC, LXD, Linux-VServer, OpenVZ/Virtuozzo y runC utilizan funcionalidades del núcleo de Linux con el fin de gestionar los recursos para implementar entornos de ejecución aislados para aplicaciones o para un sistema operativo al completo, Docker, RKT, LXC son los más acogidos en el mercado, y se los detalla a continuación [13].

Docker

Docker proyectado como uno de los mejores contenedores de software por sus características y sus grandes desarrollos como un proyecto de código abierto en la actualidad, que permite automatizar el despliegue de aplicaciones dentro de "contenedores". Salomon Hykes comenzó como un proyecto interno dentro de dotCloud, empresa enfocada a PaaS (plataforma como servicio) esta fue liberado como código abierto en marzo de 2013, con el lanzamiento de la versión 0.9 (en marzo de 2014) Docker dejó de utilizar LXC como entorno de ejecución por defecto y lo reemplazó con su propia librería libcontainer escrita en Go, que es un lenguaje de programación concurrente y compilado inspirado en la sintaxis de C, que se encarga de hablar directamente con el kernel. Actualmente es uno de los proyectos con más estrellas en GitHub, con miles de bifurcaciones (forks) y miles de colaboradores [14].

Docker recurre a características fundamentales del núcleo de Linux para aislar procesos entre sí y, gracias a su propio entorno de ejecución runC, permite que varias aplicaciones funcionen en paralelo en contenedores aislados sin necesidad de gastar recursos de la máquina. Con esto, esta liviana plataforma se consolida como alternativa a la virtualización basada en hipervisores. A continuación se detalla las ventajas y desventajas de docker.

Tabla 1: Ventajas y desventajas de Docker.

Ventajas	Desventajas
Docker soporta diversos sistemas operativos y plataformas en la nube.	El motor de Docker solo es compatible con su propio formato de contenedor.
Con Swarm y Compose, la plataforma Docker ya ofrece herramientas de gestión de clústeres.	El software está disponible como archivo monolítico que contiene todas sus características.
Con el Hub de Docker los usuarios cuentan con un registro central de recursos.	Los contenedores Docker solo aíslan procesos entre sí, pero no virtualizan sistemas operativos (full system container).
El ecosistema Docker, en constante crecimiento, pone a disposición de sus usuarios diversas herramientas, plugins y componentes de infraestructura.	

Elaborado por: El investigador

RKT (rocket)

En febrero de 2016, con la versión 1.0 de rkt, CoreOS hizo el primer lanzamiento estable del entorno de ejecución de contenedores. El rival de Docker aspiraba a superar a su contrincante en funciones de seguridad. Entre estas destacan, junto al aislamiento de contenedores basado en KVM (Kernel-based Virtual Machine o máquina virtual basada en el núcleo), la compatibilidad con la extensión del núcleo SELInux, así como la validación con firma para imágenes de la especificación App Container (appc) desarrollada por CoreOS. Esta describe el formato de imagen ACI (App Container Image), el entorno de ejecución, un mecanismo de descubrimiento de imagen (image discovery) y la posibilidad de agrupar imágenes en aplicaciones multicontainer, los llamados App Container Pods. El mayor competidor de Docker en el mercado de la virtualización basada en contenedores es el entorno de ejecución rkt del distribuidor de Linux CoreOs [13]. En la tabla 1 se detalla las ventajas y desventajas de RKT.

Tabla 2: Ventajas y desventajas de RKT.

Ventajas	Desventajas
Además de su propio formato, rkt también soporta imágenes de Docker y gracias a Quay convierte cualquier imagen en el formato ACI de rkt.	Existen muchas menos integraciones de proveedores externos para rkt que para Docker.
Tecnologías como KVM y Clear Container de Intel® facilitan separar contenedores de software entre sí de forma segura.	rkt está optimizado para operar contenedores de aplicaciones, pero no soporta contenedores de sistemas operativos (full system container).

Elaborado por: El investigador

LXC (Linux Containers)

LXC se desarrolló para ejecutar diferentes contenedores de sistema (full system containers) en un sistema anfitrión. Un contenedor Linux suele iniciar una distribución completa en un entorno virtual partiendo de una imagen del sistema operativo y los usuarios interactúan con ella de forma parecida a como harían con una máquina virtual. Los contenedores Linux rara vez inician aplicaciones, con lo que se diferencian claramente de Docker: mientras que LXC se centra sobre todo en la virtualización de sistemas, Docker se concentra en la virtualización y el despliegue

de aplicaciones. Al principio también se utilizaban contenedores Linux con este objetivo. Hoy Docker apuesta por un formato de contenedor propio.

Tabla 3: Ventajas y desventajas de LXC

Ventajas	Desventajas
LXC está optimizado para operar contenedores full system.	No se utiliza de forma estándar para operar contenedores de aplicaciones.
	A excepción de Linux, no cuenta con ninguna implementación nativa para otros sistemas operativos.

Elaborado por: El investigador

En la siguiente tabla se presenta las principales características de las diferentes tecnologías de virtualización de contenedores de software, que se mencionó anteriormente:

Tabla 4: Comparación entre las diferentes tecnologías de Virtualización de contenedores.

	Docker	**RKT**	**LXC**
Tecnología de virtualización	En el nivel del sistema operativo	hipervisor	En el nivel del sistema operativo
Contenedor de sistema completo	No	No	Sí
Contenedor de aplicaciones	Sí	Sí	No
Licencia	Apache 2.0	Apache 2.0	GNU LGPLv2.1+
Formato del contenedor	Docker Container	appc, Docker Container	Linux Container (LXC)
Plataformas compatibles	Linux, Windows,macOS, Microsoft Azure Amazon Web Services (AWS)	Linux, Windows, macOS	Linux
Último lanzamiento	Abril de 2017	Febrero de 2017	Enero de 2017
El núcleo de Linux requiere un parche	No	No	No
Lenguaje de programación	Go	Go	C, Python 3, Shell, Lua

Elaborado por: El investigador

Diferencia entre Contenedores de Software y Máquinas Virtuales

Antes de la llegada de los contendores, se utilizaban las llamadas "máquinas virtuales", esta tecnología permitía que un servidor ejecutara muchas aplicaciones independientes unas de otras. El funcionamiento de las máquinas virtuales consiste en empaquetar el sistema operativo y el código juntos. El sistema operativo cree que tiene un servidor exclusivo para su aplicación, pero en realidad este está siendo compartido por muchas otras máquinas virtuales, sin embargo, el hecho de compartir servidor implica que los sistemas operativos tengan una ejecución más lenta esto se puede observar en la figura 3 [15].

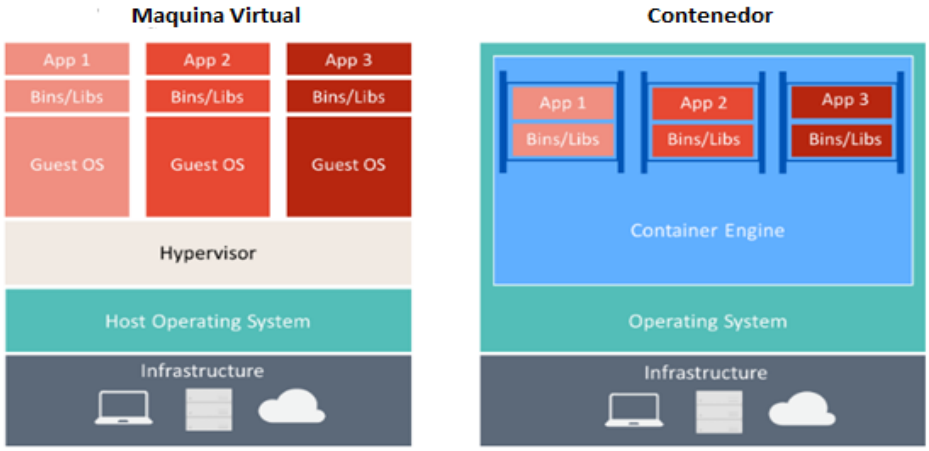


Figura 3 Comparación entre Máquinas Virtuales y Contenedores [16].

Los contenedores por otra parte tienen un funcionamiento poco diferente a las máquinas virtuales, el "Container Engine" se encarga de lanzar y gestionar los contenedores con sus aplicaciones, pero en lugar de exponer los diferentes recursos de hardware de la máquina de manera discreta (procesador y RAM) para cada aplicación, lo que hace es compartirlos entre todos los contenedores optimizando su uso y eliminando la necesidad de tener sistemas operativos separados para conseguir el aislamiento, como los paquetes solo alojan la aplicación y las librerías, los frameworks, etc, esto hace que los contenedores ocupen muy poco espacio y que la sobrecarga sea muy baja, por lo que los tiempos de ejecución son óptimos. Los contenedores comenzaron siendo una característica central de Linux hace tiempo, pero eran muy difíciles de utilizar. Para gestionar todos los contenedores, se necesita el software especializado Kubernetes (desarrollado por Google originariamente) que ayuda a introducir los contenedores en diferentes máquinas, se asegura de que se

ejecutan y permite hacer funcionar varios contenedores más con una aplicación específica cuando la demanda aumenta [15].

Estándar IEC-61499

Este estándar define una arquitectura y unos modelos software para el desarrollo de aplicaciones reconfigurables de control distribuido. Dicha arquitectura está organizada jerárquicamente mediante los modelos de sistema, dispositivo y recurso, como se observa en la figura 4 [17].

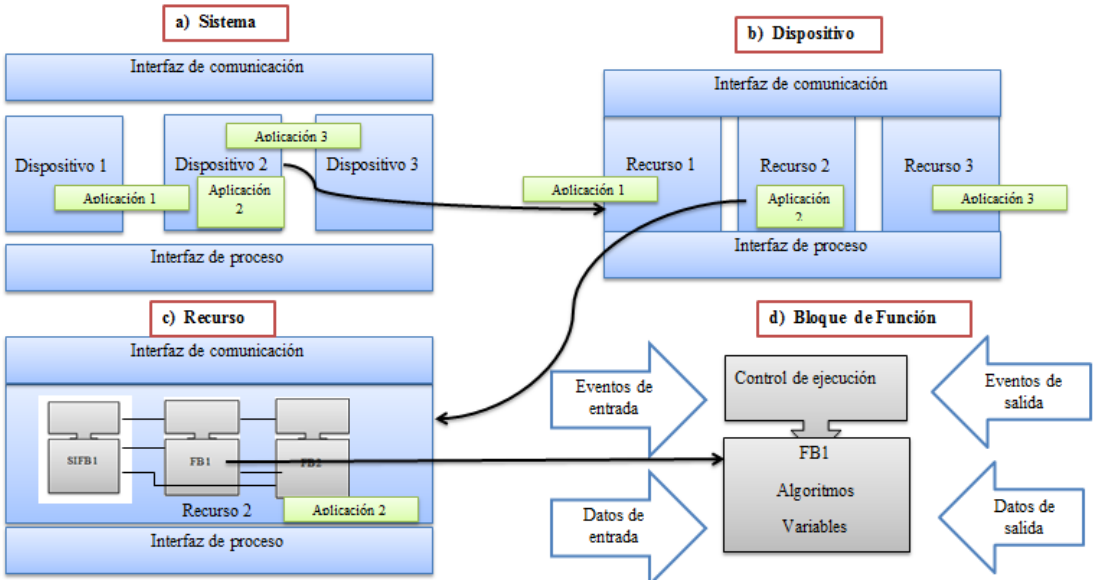


Figura 4 Modelos de IEC 61499.

Elaborado por: Investigador

Los modelos del estándar IEC-61499, se basan en la norma anterior IEC 61131. Estos modelos son descritos de manera breve a continuación [17]:

- **Sistema:** Un sistema consiste en una colección de dispositivos interconectados mediante alguna red de comunicación. La funcionalidad es modelada por aplicaciones que pueden residir en uno o, de manera distribuida, en varios dispositivos (ver Fig. 4a).
- **Dispositivo:** Representa un equipo con capacidad de contener recursos y ejecutar las aplicaciones (ver Fig. 4b). Un dispositivo está constituido por interfaces de comunicaciones y proceso, un gestor de dispositivo y cero o más recursos. Por su parte, la interfaz de proceso permite el acceso a los sensores y actuadores.

- **Recurso:** Representa un entorno para la ejecución independiente de las aplicaciones o parte de ellas (ver Fig. 4c). Estas se componen de FB (Bloque Funcional) que el recurso puede crear, eliminar y conectar entre sí. El estándar organiza el flujo de ejecución de las aplicaciones mediante eventos, siendo el recurso el encargado de gestionar su entrega.
- **Aplicación:** Representa el principal elemento de modelado de la funcionalidad de control. Las aplicaciones se organizan mediante redes de FB (Función Network Block o FBN) distribuidas en diferentes recursos, siendo los FB elementos atómicos respecto de la distribución.
- **Bloque función:** Unidad básica de modelado constituida por: eventos de entrada y salida, datos de entrada y salida, un control de ejecución (Execution Control o EC), algoritmos y variables internas (ver Fig. 4d). Los algoritmos contienen las acciones que realiza el FB, mientras que las variables representan el estado interno del FB.

4DIAC-IDE

Es un IDE de código abierto destinado a sistemas industriales distribuidos. También es aplicable en otras áreas, como la automatización del hogar, las redes eléctricas o donde sea que se necesite algún sistema automatizado. Puede "programar" una aplicación utilizando Bloques de funciones. La misma aplicación puede ejecutarse en diferentes dispositivos, como un Raspberry Pi, un PLC o su computadora, ya que la aplicación es independiente del dispositivo en el que se ejecuta, cada bloque de funciones (FB) es como una función normal que se ejecuta cuando llega un evento (línea roja entrante a la izquierda del FB). Las líneas azules entrantes son las entradas de datos (parámetros de la función) y las líneas azules salientes son los parámetros enviados a otros FB [18], como se observa en la figura 5.

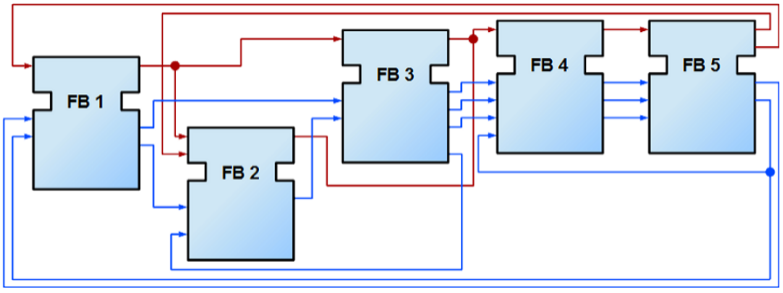


Figura 5 Esquema de Bloques Funcionales 4DIAC [18].

Runtime 61449: Forte

Es una pequeña plataforma de ejecución implementada en C++ y de código abierto, desarrollada por el mismo equipo que 4DIAC. Esta especialmente desarrollada para ejecutarse en pequeños sistemas embebidos y sistemas de tiempo real, ya que tiene los mecanismos necesarios para poder asegurar la ejecución en tiempo real y la gestión de prioridades. Otro de los puntos fuertes de FORTE es que ha sido diseñada para ser independiente de la plataforma sobre la que se ejecute, para poder ser ejecutada en diferentes hardware y sistemas operativos [19].

El entorno de ejecución de 4DIAC-RTE, FORTE es una implementación portable de un entorno de ejecución IEC-61499. Está enfocado a pequeños dispositivos (16/32 bits). FORTE además proporciona una infraestructura de comunicación flexible a través de las llamadas capas de comunicación. Además, proporciona una arquitectura escalable que permite adaptarse al diseño de la aplicación [20].

CMake

CMake es una familia de herramientas multiplataforma de código abierto diseñada para crear, probar y empaquetar software. CMake se utiliza para controlar el proceso de compilación de software utilizando una plataforma simple y archivos de configuración independientes del compilador, y generar archivos de trabajo y espacios de trabajo nativos que se pueden utilizar en el entorno del compilador que elija. Kitware creó el conjunto de herramientas CMake en respuesta a la necesidad de un entorno de construcción potente y multiplataforma para proyectos de código abierto como ITK y VTK. CMake es parte de la colección de Kitware de plataformas de código abierto con soporte comercial para el desarrollo de software [21].

Protocolos de Comunicación en la Industria 4.0

Con la aparición de los dispositivos IoT, surge el concepto de industria 4.0. Podemos definirla como la digitalización completa a través de la integración de tecnologías de procesamiento de datos, software inteligente y sensores, desde los proveedores hasta los clientes. Con el fin de esta discusión, es importante agrupar los protocolos en dos categorías: cliente/servidor (client/server) y publicar/suscribir (publish/subscribe). Los protocolos cliente/servidor requieren que el cliente se conecte al servidor y realice solicitudes. Por otro lado, los protocolos publicar/suscribir requieren que los

dispositivos se conecten a un “tópico” de un gestor intermediario y publiquen la información [22].

Existen varios protocolos: algunos son privados y otros que son estándares abiertos, en los estándares abiertos tenemos protocolos que tienen el potencial de conectar dispositivos industriales con diferentes plataformas. Pero para que esto sea posible, los dispositivos deben poder comunicarse, tanto entre sí, como hacía el exterior. Por ello comentamos algunos de los protocolos de comunicación existentes en la industria [22].

TCP/IP

El protocolo IP (Internet Protocol) es la pieza fundamental en la que se sustenta el sistema TCP/IP, el protocolo IP facilita un sistema sin conexión (connectionless) y no fiable (unreliable) de entrega de datagramas entre dos ordenadores cualesquiera conectados a Internet. El protocolo TCP (Transmission Control Protocol) se podría definir como un protocolo orientado a una conexión fiable y orientada a un flujo de bytes. Aunque el protocolo TCP al igual que UDP utiliza los servicios de IP para su transporte por Internet, es un protocolo orientado a conexión. Esto significa que las dos aplicaciones envueltas en la comunicación (usualmente un cliente y un servidor), deben establecer previamente una comunicación antes de poder intercambiar dato [23].

UDP

El protocolo UDP (User Datagram Protocol) se podría definir como un protocolo simple y orientado a datagrama, de esta forma cada envío de datos se corresponde a un único envío de un datagrama independiente del resto de datagramas y de la misma comunicación, siguiendo los preceptos de encaminamiento de datagramas por Internet, la entrega al destino no está asegurada por el propio protocolo. Debido a que UDP utiliza IP para su transporte por Internet, en caso de ser, este se fragmentará y ninguno de estos fragmentos, proporcionara ningún tipo de seguridad o fiabilidad en la entrega, debido a la sencillez de este protocolo, el diseño de su cabecera, es mucho más simple que el de IP [23].

HTTP

El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo cliente/servidor de la Capa de Transporte y como tal los datos pueden presentarse en múltiples formatos tales como; HTML, JavaScript, JavaScript (JSON), XML, etc. De todos estos formatos el más usado, para el intercambio de información entre dispositivos, es JSON pues es ligero y flexible y disponemos de infinidad de librerías para usarlo con cualquier lenguaje de programación. HTTP está pensado para que los nodos clientes puedan tener acceso a los recursos del nodo servidor a través de solicitudes, donde en la mayoría de los casos, un recurso es un dispositivo. Aunque HTTP es ampliamente usado en la industria (sobre todo en su versión segura HTTPs), básicamente se utiliza para configuración de dispositivos pero no para acceso a los datos que éstos contienen. También existe el problema de la interoperabilidad ya que aunque multitud de dispositivos soportan HTTP (REST/JSON) no son totalmente compatibles entre ellos y normalmente debe realizarse algún ajuste para que puedan comunicarse [22].

MQTT

El protocolo MQTT (Message Queue Telemetry Transport) fue creado por técnicos de IBM y Arcom en 1999 como un modo rentable y seguro de supervisar dispositivos de medición remotos. Usa un modelo de comunicación muchos con muchos a través de un servidor centralizado llamado Broker. Este protocolo que funciona sobre TCP (aunque existe una versión denominada MQTT-S que lo hace sobre UDP), pertenece a los denominados de publicación/suscripción y ha comenzado a utilizarse a gran escala tras la aparición del Internet de Cosas (IoT) pues como veremos dispone muy buenas cualidades para usarse en este campo. MQTT es un protocolo abierto, sencillo, muy ligero y muy fácil de implantar. Su funcionamiento es muy parecido al de Twitter. Los dispositivos publican información sobre un asunto o topic en el broker quien la reenvía a su vez a aquellos clientes que se hayan suscrito al mismo [22].

MQTT está considerado como uno de los mejores protocolos para datos “vivos” ya que el broker desacopla los publicadores de los suscriptores y ambos pueden

desatender las tareas de comunicación para realizar otras, ganando tiempo de proceso y ahorrando energía. Entre sus ventajas destacan las siguientes [22]:

- Está especialmente adaptado para utilizar un ancho de banda mínimo
- Es ideal para utilizar redes inalámbricas
- Consume muy poca energía
- Es muy rápido y posibilita un tiempo de respuesta superior al resto de protocolos web actuales
- Permite una gran fiabilidad si es necesario
- Requiere pocos recursos
- Además, los topics MQTT tienen un grado jerárquico y disponen de los operadores (+,#) para facilitar suscribirse a un determinado nivel de la jerarquía.

Sistemas de Realidad Virtual

Realidad virtual, es un entorno con un conjunto de elementos, imágenes y sonidos por computadora que dan la sensación de realidad, debido a esto la aplicabilidad de esta técnica es bastante extensa, en el área de la salud se encuentra un gran reto de enfocar estas aplicaciones a simuladores que permitan que los profesionales mejoren su experiencia al momento de enfrentarse a un problema de la vida real. Dichos simuladores tienen la particularidad de que son ambientes con condiciones interactivas y reactivas, permitiendo al usuario interactuar con los objetos virtuales y a su vez recibir una retroalimentación háptica, al momento de implementar estas aplicaciones se debe encontrar un equilibrio entre el modelado de los objetos y la velocidad de renderizado en tiempo real del sistema, todo esto representado en el rendimiento de la ejecución de la aplicación, los diferentes investigadores han buscado desarrollar varios métodos para superar dichos problemas [24]. A continuación analizamos las ventajas y desventajas de tres de los más utilizados: Unity, Unreal Engine 4 y Vray .

Unity 3D

El Editor de Unity presenta herramientas múltiples que permiten una edición e iteración rápidas en tus ciclos de desarrollo, lo que incluye el modo Play para tener vistas previas rápidas de tu trabajo en tiempo real, disponible en Windows, Mac y Linux. Incluye una variedad de herramientas ideales para los artistas que les permite

diseñar experiencias y mundos de juegos envolventes, así como un conjunto robusto de herramientas para desarrolladores destinadas a implementar la lógica del juego y un juego de alto rendimiento. Unity apoya tanto el desarrollo de la tecnología 2D como el de la 3D con prestaciones y funcionalidades para tus necesidades específicas en los diversos géneros [25].

Este programa cuenta con la ventaja de que tiene una interfaz gráfica sencilla y de que cuenta con menús que son fáciles de usar y modificar sin que haya que escribir código (aunque sí hay que editarlo). Ofrece una versión que es gratuita, aunque no cuenta con todas las herramientas con las que cuenta la versión de pago; y esta última es muy costosa. A favor es que el programa permite la instalación de muchos plugins, muchos de los cuales podemos encontrar de forma gratuita. Otras de las ventajas es que la comunidad que usa Unity es muy grande, por lo que se pueden encontrar fácilmente vídeos y tutoriales sobre su uso. Finalmente, un gran inconveniente es su calidad gráfica, que deja mucho que desear frente a otros motores de render como Unreal Engine 4 y nunca conseguirá alcanzar la calidad de imagen de los motores de videojuegos Triple A [26].

Unreal Engine 4

Es cierto que Unreal Engine 4 no cuenta con una gran comunidad, lo que dificulta su aprendizaje, pero es sin duda alguna la mejor herramienta de renderizado para el campo de la Arquitectura. Su principal característica es que trabaja en tiempo real, por lo que los cambios se implementan casi al instante, ahorrando mucho tiempo. Es el mejor motor para los videojuegos (creado por Epic Games, ha dado a luz el popular Fornite) pero su usabilidad y su potencia gráfica lo hacen también ideal para los arquitectos, ya que las mejoras de la última versión en iluminación, efectos y texturas lo acercan mucho al fotorrealismo [26].

Además, es completamente gratuito en su integridad para los arquitectos: no hay que comprar upgrades, ya que el software al completo está disponible sin coste alguno y sólo habría que pagar cierto porcentaje (5%) si se comercializan productos elaborados con el motor, y sólo a partir de 3.000 dólares de facturación.

Otra de las ventajas de Unreal Engine es su programación visual y sencilla a través de Blueprints, lo que elimina la necesidad de escribir código [26].

Vray RT

Hubo un tiempo que VRay era el motor de render predilecto y destacaba principalmente por sus algoritmos de iluminación global y otros elementos, como los mapeo de fotones, mapas de irradiación y fuerza bruta. Vray renderiza prácticamente a tiempo real. Esto quiere decir que los cambios se ven a tiempo real en el render, una característica que comparte con Unreal Engine [26].

La principal desventaja de Vray es que su licencia es cara, lo que es difícil de asumir para los pequeños estudios de arquitectura. Más recientemente se ha lanzado V-Ray for Unreal, que permite pasar las escenas con materiales de este programa a Unreal sin ninguna dificultad. En definitiva, Unreal Engine 4 es el que presenta las características que lo hacen mejor para la Arquitectura: es gratis, fácil de usar, más rápido y con una calidad fotorrealista tremenda [26].

En la siguiente tabla se presenta, la comparación entre los diferentes software de realidad virtual descritos anteriormente:

Tabla 5: Comparación entre los diferentes software de realidad virtual.

	Unity	**Unreal Engine 4**	**Vray**
Calidad Grafica	Media	Buena (fotorrealista)	Buena (fotorrealista)
Interfaz gráfica	Sencilla (Fácil uso por el usuario)	Sencilla (programación visual y sencilla a través de Blueprints)	Complejo
Licencia	Gratuita	Gratuita	Pagada
Comunidad que utiliza el software	Grande	Pequeña	Pequeña

Elaborado por: El investigador

Hardware para automatización de bajo costo

La evolución de dispositivos tecnológicos avanza de una manera increíble, obteniendo modernos diseños electrónicos y digitales, tanto una maquinaria real,

cables, transistores, y circuitos, que componen la parte de Hardware; las instrucciones y los datos que forman parte del Software. Dentro del avance tecnológico en los ordenadores llega la microcomputadora, que comenzó a ser utilizado popularmente luego de la introducción de las minicomputadoras. La principal diferencia con su predecesora es que las microcomputadoras reemplazaron múltiples componentes separados, que fueron integrados en un único chip, el microprocesador, generalmente son computadoras que ocupan espacios físicos pequeños, comparadas a sus predecesoras históricas, las mainframes y las minicomputadoras [27].

Una placa computadora u ordenador de placa reducida (Single Board Computer o SBC) es una computadora completa en un sólo circuito. El diseño se centra en un sólo microprocesador con la RAM, E/S y todas las demás características de un computador funcional en una sola tarjeta que suele ser de tamaño reducido, y que tiene todo lo que necesita en la placa base [27].

Entre las ventajas de utilizar un ordenador de placa reducida, tenemos su bajo costo al adquirirla, el consumo bajo de amperaje, el acoplamiento de accesorios de fácil disponibilidad como un monitor, un mouse etc., y su tamaño reducido comparado con los diferentes tipos de ordenadores, Existen muchos ordenadores de placa Reducida, a continuación se menciona algunas de las más destacadas en el mercado [28].

Raspberry Pi

Raspberry Pi es una placa de ordenador desarrollada en el Reino Unido por la fundación Raspberry Pi. La idea del proyecto se concibió en 2006 como una solución orientada a entornos educativos para estimular la enseñanza de ciencias de la computación en las escuelas. Esta placa incluye procesador, memoria RAM, ranura para tarjetas SD, conectores USB, vídeo RCA, Conexión Jack de 3,5 pulgadas para audio, conector HDMI, entre otros; además, permite ejecutar Linux o sistemas RISC. Todo esto posibilita la ejecución de miles de aplicaciones con grandes beneficios y bajo costo [29].

Nvidia Jetson Nano

Es un pequeño ordenador, de gran alcance que le permite ejecutar múltiples redes neuronales en paralelo para aplicaciones como la clasificación de imágenes, detección de objetos, segmentación y procesamiento del habla. Todo en una plataforma fácil de usar que funciona en tan solo 5 vatios. NVIDIA Jetson Nano es un sistema integrado en el módulo (SoM) y un kit de desarrollo de la familia NVIDIA Jetson, que incluye una GPU Maxwell de 128 núcleos integrada, una CPU ARM A57 de 64 bits, memoria LPDDR4 de 4GB, junto con soporte para E / S de alta velocidad MIPI CSI-2 y PCIe Gen2, ejecuta Linux y proporciona 472 GFLOPS de rendimiento de cómputo FP16 con 5-10W de consumo de energía [30].

Odroid XU4

Este dispositivo tiene características como 1 puerto USB 2.0, 2 puertos USB 3.0, un puerto Ethernet que soporta velocidades de transferencia Gigabit, un conector HDMI para monitores 720p y 1080p, y un conector de alimentación de 5V/4ª, también incluye puertos GPIO de 12 pines y 30 pines, un conector de batería RTC externo, un puerto de consola serie USB-UART, un conector para módulos eMMC y una ranura para tarjetas microSD [31].

En la siguiente tabla se presenta las principales características de los diferentes ordenadores de placa reducida, que se mencionó anteriormente:

Tabla 6: Comparación entre los de ordenadores de placa reducida.

	Raspberry Pi3 B+	**Nvidia Jetson Nano**	**Odroid XU4**
Procesador	(ARMv8) 64-bit SoC @ 1.4GHz	Brazo de cuatro corazones A57 de 128 nucleos 1.43 GHz	ARM de 8 nucleos 2 Ghz
Memoria RAM	1.5GB LPDDR2 SDRAM	4GB de 64 bits LPDDR4 25.6 GB/s	2 GB de memoria RAM DDR3
Puertos de red	Gigabit Ethernet sobre USB 2.0 (300 Mbps)	Gigabit Ethernet, M.2 Key E	GigaEthernet 10/100/1000
Puertos USB	4 * USB 2.0	4*USB 3.0, USB 2.0 Micro-B	2 * USB 3.0 y 1USB 2.0
Almacenamiento	MicroSD (No incluida)	MicroSD (No incluida)	Memoria de estado solido

Elaborado por: El investigador

1.2 Objetivos

El objetivo principal del presente proyecto de investigación radica en la Construcción de una arquitectura de comunicación y virtualización de procesos industriales basados en Industria 4.0, con el fin de mejorar el rendimiento de la planta de una empresa u organización, combinando técnicas avanzadas de producción mediante la incursión de tecnologías inteligentes alcanzando los objetivos de una nueva revolución industrial, obteniendo beneficios en la empresa.

Para realizar la virtualización de procesos industriales es necesario conocer los fundamentos y conocer sobre las formas de virtualización de un ordenador, razón por la cual el presente proyecto parte de un análisis de los diferentes contenedores de software para utilizarse en un computador. Para ello requerirá de las siguientes actividades:

1. Recolección de información sobre sistemas virtualizados y protocolos de comunicación que se utilizan en la actualidad.
2. Descripción general y análisis de contenedores de software para la virtualización de procesos industriales.

Realizado un análisis sobre los contenedores de software y escoger el más apropiado para el proyecto, se procede a determinar el protocolo para la comunicación entre contenedores para su uso en Industria 4.0, el cual permitirá obtener un envío y recepción de datos entre los contenedores, para la consecución de este objetivo será necesario realizar las siguientes actividades:

1. Análisis de protocolos de comunicación que se utilizan en los procesos de manufacturación.
2. Determinación del protocolo de comunicación óptimo para la aplicación.

Posterior a la comunicación de contenedores del sistema, con base en toda la información recibida se deberá diseñar una aplicación para la comunicación y virtualización de procesos para la automatización y control del sistema, para ello se deberá realizar las siguientes actividades:

1. Desarrollo de la arquitectura de comunicación y virtualización de procesos industriales

2. Diseño de la aplicación para el control de la arquitectura.

Finalmente, realizado la etapa de control se procede aplicar la arquitectura de virtualización y comunicación de procesos industriales en un prototipo robótico industrial, para la comprobación del funcionamiento de la arquitectura propuesta en el presente proyecto, a través de las siguientes actividades.

1. Construcción del prototipo robótico Industrial para la arquitectura de comunicación y virtualización de procesos industriales.
2. Pruebas, corrección de errores y validación de la arquitectura planteada.

CAPÍTULO 2
METODOLOGÍA

2.1 Materiales

El desarrollo de la metodología para el presente proyecto se requiere de los siguientes materiales: artículos publicados en revistas científicas, proyectos desarrollados de investigación afines al tema principal, libros, información sobre sistemas que utilizan tecnología idéntica al presente proyecto, dispositivos y equipos, en lo que corresponde al Hardware del sistema, por otro lado en el software se utilizó aplicaciones para la configuración de los mismos equipos como también para programación del desarrollo del sistema.

2.2 Métodos

2.2.1 Modalidad de la investigación

La modalidad del proyecto para la Construcción del prototipo con arquitectura para comunicación y virtualización de procesos industriales basados en Industria 4.0 es de Investigación y Desarrollo, la cual se realizará mediante las siguientes técnicas.

Modalidad de Campo

Debido a que se realizó un estudio sistemático de los hechos en el lugar en que se producen los acontecimientos, para tener información que permitan resolver los objetivos del proyecto, planteando un caso real y un ejemplo virtual para conocer los datos obtenidos.

Modalidad Bibliográfica

Porque el sustento científico del tema planteado se obtuvo de libros, publicaciones, artículos científicos y repositorios disponibles en el internet.

Modalidad Aplicada

Debido a que se resolvió situaciones problemáticas, aplicando conocimientos ya existentes para la solución de un determinado problema, y proponiendo la Construcción de un prototipo sobre una arquitectura para la comunicación y virtualización de procesos industriales basados en I4.0.

2.2.2 Recolección de la Información

Para la recolección de información se optó por el uso de documentos, revistas, libros, proyectos desarrollados, por lo que se tomará en cuenta bases de datos confiables que permitirán la obtención de información, cada una de estas relacionadas y vinculadas a la virtualización de procesos industriales y protocolos de comunicación tomando en cuenta normas y estándares basados en industria 4.0 en los últimos años, teniendo una investigación bibliográfica confiable.

2.2.3 Procesamiento y Análisis de Datos

Una vez que se obtenga la información apropiada de la investigación planteada, esta formará parte de un proceso analítico, el cual consistirá en:

- Revisión de la información recogida.
- Revisión del estado actual sobre la virtualización de procesos industriales que se utiliza en las empresas u organizaciones.
- Análisis e interpretación de los Resultados.

Además de fuentes bibliográficas relacionadas a la virtualización de procesos industriales como papers, tesis de grado, páginas web, libros, folletos y revistas científicas. El análisis de los resultados obtenidos se presentará en un informe técnico destacando los datos requeridos en los objetivos planteados

2.2.4 Desarrollo del Proyecto

A continuación, se presenta el desarrollo de las actividades necesarias que se efectuaran para el desarrollo del proyecto de investigación:

1. Recolección de información sobre sistemas virtualizados y protocolos de comunicación que se utilizan en la actualidad.

2. Descripción general y análisis de contenedores de software para la virtualización de procesos industriales.
3. Selección del contenedor de software adecuado para la etapa de virtualización.
4. Análisis de protocolos de comunicación que se utilizan en los procesos de manufacturación.
5. Determinación del protocolo de comunicación óptimo para la aplicación.
6. Desarrollo de la etapa de comunicación mediante el protocolo escogido.
7. Desarrollo de la etapa de virtualización de procesos, de acuerdo con parámetros basados en industria 4.0
8. Calibración y adaptación para la comunicación de los procesos virtualizados.
9. Desarrollo de la arquitectura de comunicación y virtualización de procesos industriales.
10. Virtualización de procesos industriales para el prototipo robótico Industrial.
11. Pruebas, corrección de errores y validación de la arquitectura planteada.
12. Elaboración del Informe Final.

CAPÍTULO 3
RESULTADOS Y DISCUSIÓN

3.1 Desarrollo de la propuesta

En el presente trabajo se desarrolló una arquitectura para la comunicación y virtualización de procesos industriales, se creó aplicaciones llamadas "FORTE", quienes encapsulan, toda la configuración y la comunicación entre los distintos dispositivos del sistema, estas fueron diseñadas en el software 4DIAC-IDE, el cual se basa en la norma IEC-61449, mediante un lenguaje de programación de Bloques Funcionales, además se creó varios contenedores de software quienes en sus sistemas tienen incorporado un directorio con la aplicación FORTE según su función. También se diseñó una interfaz gráfica para el control de los contenedores y la ejecución de las aplicaciones.

En el sistema propuesto se utilizó, diferentes protocolos de comunicación, configurados tanto en las aplicaciones FORTE, como en los dispositivos del sistema, optando el protocolo UDP para la comunicación entre los diferentes contenedores de software, como también se optó por el protocolo TCP/IP para la comunicación con el dispositivo Arduino, quien recibe los datos que son enviados desde la aplicaciones, este los configura y controla un prototipo de brazo Robótico, por otro lado para la visualización de la planta, se desarrolló un modelo en 3D del prototipo de brazo robótico físico en el software Unity instalado en un ordenador, para observar las acciones que está realizando el prototipo, en este software se configuro protocolo MQTT para receptar los datos enviados desde FORTE.

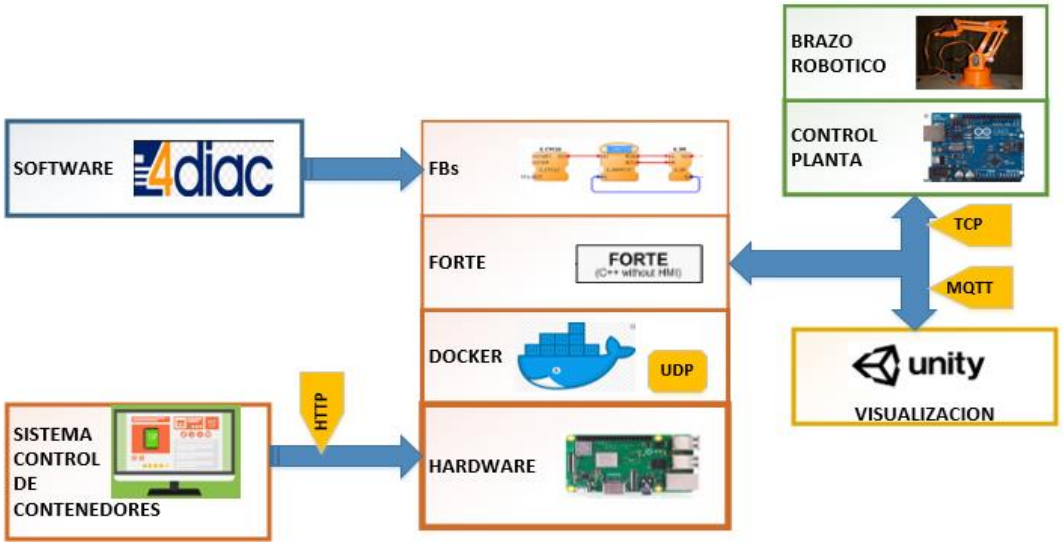


Figura 6 Esquema general del sistema

Elaborado por: El investigador

3.1.1 Selección de Software

Selección de la plataforma de contenedores de software

Se ha optado por Docker como plataforma de contenedores de software, debido a su facilidad de automatizar y crear nuevos contenedores de software en poco tiempo, así como sus instancias se inician en pocos segundos, otro punto importante es que ocupan mucho menos espacio y consumen menos recursos que otros software que tienen el mismo objetivo, obteniendo un mayor rendimiento que la virtualización tradicional.

Selección del entorno de Desarrollo

Se ha optado por 4DIAC-IDE, el cual es un entorno de desarrollo de código abierto destinado a sistemas industriales distribuidos, además de ser una aplicación robusta, intuitiva y tener soporte oficial, puede "programar" una aplicación utilizando Bloques de funciones, proporcionados por el software o crear nuevos según sea las necesidades del proyecto. Las aplicaciones modeladas se pueden descargar a dispositivos de campo distribuidos según los medios definidos por la norma IEC-61499, además es compatible con la mayoría de las herramientas actualmente existentes y ha demostrado su portabilidad con nxtStudio y FBDK.

Selección de la plataforma de ejecución

Como plataforma de ejecución para la implementación del sistema de control de procesos se ha seleccionado 4DIAC-RTE, FORTE debido FORTE debido a ser independiente de la plataforma sobre la que se ejecute, para poder ser ejecutada en diferentes hardware y sistemas operativos, además es compatible con una gran cantidad de tarjetas embebidas y Controladores Lógicos Programables, y proporciona una infraestructura de comunicación flexible a través de las llamadas capas de comunicación, como también una arquitectura escalable que permite adaptarse al diseño de la aplicación.

Selección del entorno Virtual

Para la visualización del sistema en ejecución se seleccionó Unity 3D, debido a que brinda múltiples herramientas que permiten una edición e iteración rápidas en tus ciclos de desarrollo, lo que incluye el modo Play para tener vistas previas rápidas de tu trabajo en tiempo real, con la disponibilidad en diferentes sistemas operativos Windows, Mac y Linux. Además que Unity apoya tanto al desarrollo de la tecnología 2D como el de la 3D con prestaciones y funcionalidades para las necesidades del presente proyecto.

3.1.2 Selección de Hardware

Selección del ordenador de placa reducida

Se optó como ordenador para el presente proyecto la Raspberry Pi modelo PI3 B+, por las características que cuenta y puede aportar al mismo, entre los beneficios más importantes destaca el procesador que funciona a 1.4 Ghz, alcanzando una confianza de no tener complicaciones al correr aplicaciones que necesita el proyecto dentro de este dispositivo, un punto importante es la tarjeta de red, el dispositivo cuenta con Gigabit Ethernet, que es capaz de alcanzar los 300 Mbps al funcionar sobre USB 2.0, añadiendo su bajo costo y características técnicas excelentes, ante los demás competidores.

Selección del controlador de la planta

Como controlador del prototipo de brazo robótico se seleccionó la placa de hardware libre Arduino UNO por sus características técnicas y adaptables al presente proyecto,

por su microcontrolador reprogramable, también cuenta con un software gratis, libre y multiplataforma, ya que permite instalar en nuestro ordenador con Windows, Linux etc., y también escribir, verificar y guardar ("cargar") en la memoria del microcontrolador de la placa Arduino el conjunto de instrucciones que deseamos que este empiece a ejecutar. El modelo Arduino Uno, dispone de un microcontrolador en formato SMD, 14 pines digitales se puede usar como entrada o como salida, dependiendo de la programación compilada en la placa Arduino, también dispone de 6 pines de entrada analógicos que trasladan las señales a un conversor analógico/digital de 10 bits y demás pines destinados para la trasmisión de señales TTL, también pines para la comunicación SPI y I2C.

3.1.3 Diseño del sistema de control

Creación de un Bloque de función (Proceso)

La creación de un Bloque Funcional se llevó a cabo en el software 4DIAC, se realizó el diseño del mismo cumpliendo con la estructura del estándar IEC-61449, escogiendo un modelo básico que nos proporciona el software, y a partir de este se añadió, entradas y salidas de eventos como también entradas y salidas de datos, el diseño que se realizó para el presente proyecto se muestra en la figura 7.

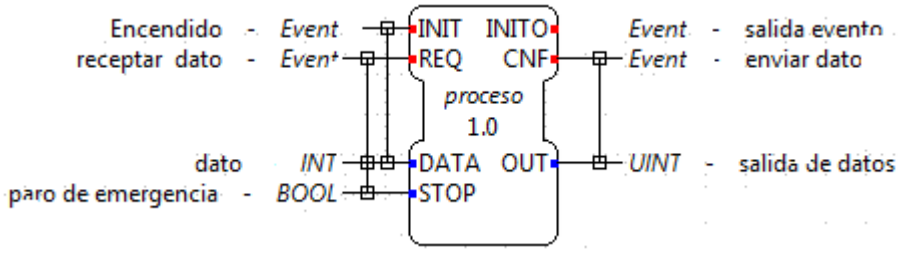


Figura 7 Bloque Funcional creado

Elaborado por: El investigador

El diseño del Bloque Funcional consta de eventos, uno que activa el Bloque "INIT" y el otro envía el dato llamado "REQ", al igual se tiene entradas de datos, la primera llamada "DATA" que se encarga de recibir el dato que llega al bloque funcional y es configurado obteniendo una salida, y la otra entrada llamada "STOP", este se activa en caso de error al no encontrar al servidor que se configuro en Arduino y desactiva él envió de datos. En las salidas de los eventos se tiene "INITO", quien se puede

conectar a otro Bloque funcional para activarlo, También tenemos un “CNF”, que nos proporciona la activación de envió del dato a la salida llamada “OUT”.

También se realizó la configuración del (ECC) grafico de control de ejecución, donde se realiza la unión de los eventos con los datos, cada vez que se active el evento REQ se recepta un dato, como también se activa el evento CNF que activa las salida de datos. El evento INIT que enciende el bloque funcional, será activado por la señal START del sistema, esto se observa en la figura 8.

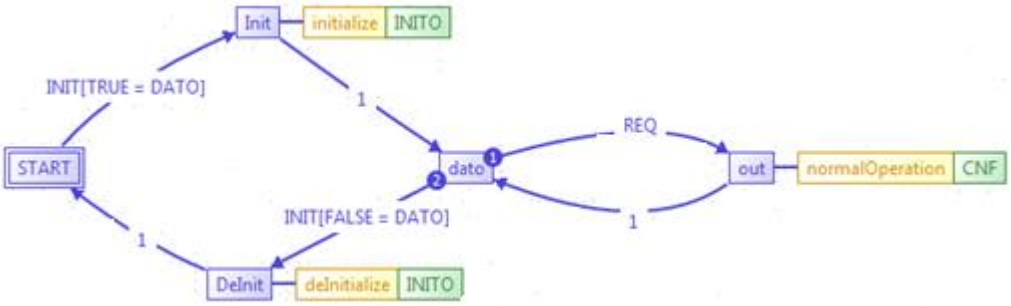


Figura 8 Grafico de control de Ejecución

Elaborado por: El investigador

Para la configuración del bloque funcional y su exportación se sigue una secuencia de pasos descrita en la figura 9**.**

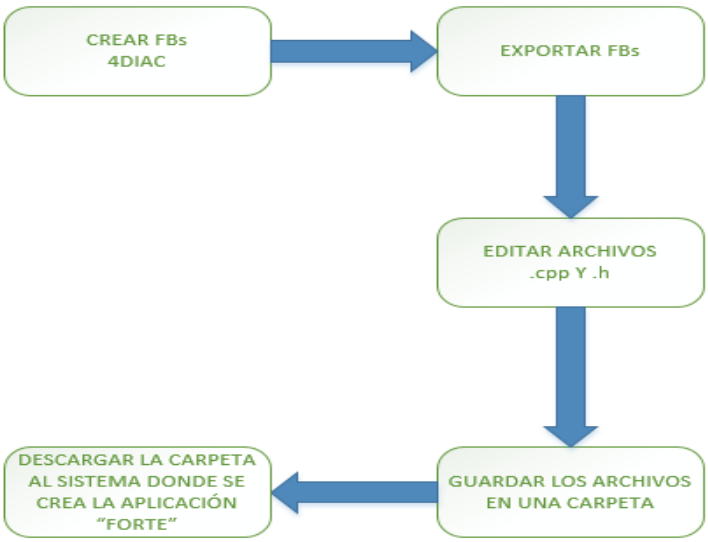


Figura 9 Esquema de configuración y exportación de un bloque funcional

Elaborado por: El investigador

Una vez creado el bloque funcional se procede a exportarlo desde 4DIAC, al realizar la exportación 4DIAC devuelve archivos en lenguaje C++, con el nombre del bloque funcional creado, en este caso se llamara proceso1, específicamente se obtuvo dos archivos importantes proceso1.cpp y proceso1.h, estos archivos se configuraron dando una función única para el bloque funcional. A continuación vemos las características y la configuración de cada uno de estos archivos:

- **proceso1.h:** ene este archivo se añaden las librerías que se van a utilizar, en el proyecto presente solo se utiliza las librerías que vienen por defecto ya creadas (**Anexo 3**).
- **proceso1.cpp:** en este archivo se programa el algoritmo que controla el sistema cuando se ejecute, así como sus entradas y salidas (**Anexo 4**).

Realizado la configuración de los archivos, se procede a descárgalos a un directorio, que posteriormente tendremos que llamarlos para ejecutar la compilación de la aplicación Forte, en el presente proyecto se creó dos bloques funcionales, el anteriormente mencionado proceso1, y otro que lo llamaremos proceso2, estos tienen el mismo diseño creado, pero tienen una configuración diferente dándole una función única e independiente de la otra.

Creación de aplicación Forte

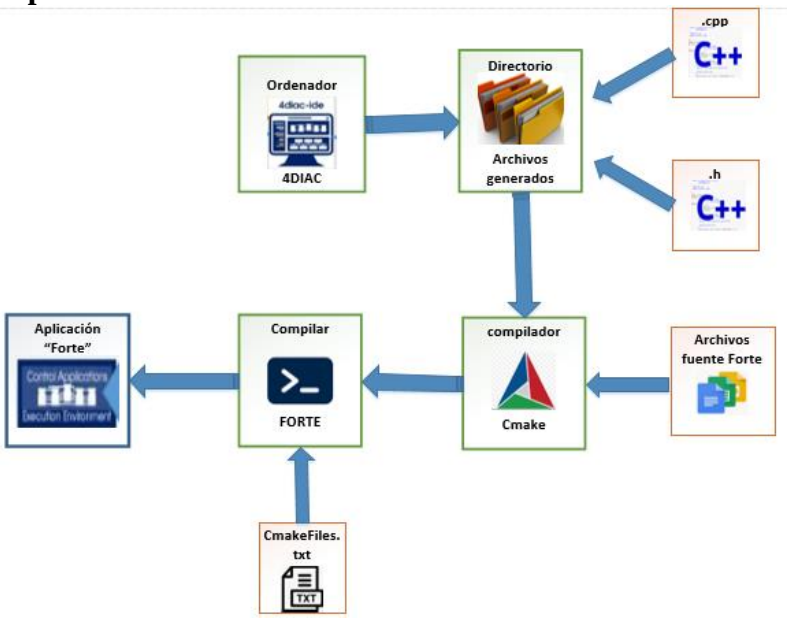


Figura 10 Diagrama de compilación de la aplicación Forte

Elaborado por: El investigador

Finalizado la creación del proceso en 4DIAC, editado los archivos .cpp y .h , se procede a configurar el software CMAKE, añadiendo el directorio donde se encuentran los archivos configurados, una vez hecho esto se configura y se genera la compilación, en donde se configura el archivo CMakeLists.txt, aquí se añaden los ficheros, librerías y el nombre de nuestro bloque funcional creado, hecho esto se procede a compilar, ingresando por la terminal del dispositivo que tiene las archivos generados, y se ingresa al directorio creado que tiene los archivos generados por Cmake, se compila al ingresar la palabra "cmake", dando como resultado una aplicación llamada "FORTE".

Bloques Funcionales (4DIAC)

Una vez creado los bloques funcionales, se procedió al diseño del proceso que va contener la aplicación Forte, esto se realizó en el software 4DIAC, para esto se utilizó bloques funcionales que nos proporciona el software, tanto para la comunicación como para la configuración de los procesos, a continuación detallamos los bloques funcionales que se utilizó para la comunicación entre contenedores y los dispositivos de nuestro sistema.

PUBLIS/ SUBSCRIBE (UDP)

Para la comunicación entre contenedores se utilizó dos bloques funcionales que son PUBLISH y SUBSCRIBE, mediante la configuración correspondiente como se puede apreciar en la figura 11, se puede enviar la información de un dispositivo a otro en este caso desde un contenedor a otro, su principal característica que ocupa una dirección y un puerto especial "239.0.0.1:61000", para la conexión entre estos dos bloques funcionales, en la sección de comunicaciones entre contenedores se puede entender de mejor manera el funcionamiento de los mismos.

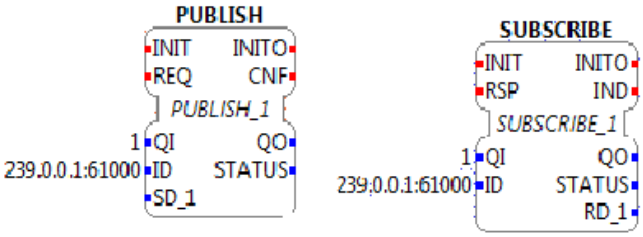


Figura 11 Bloques Funcionales PUBLISH/SUBSCRIBE UDP

Elaborado por: El investigador

CLIENT (TCP)

La placa Arduino es el controlador del Brazo Robótico, y recepta los datos que son enviados desde la pequeña aplicación forte que están en los contenedores, para esta comunicación se utilizó el bloque funcional "CLIENT", quien se encarga de enviar los datos hacia el servidor configurado en Arduino, por medio de la dirección del servidor y un puerto habilitado según estándar IEC-61449, en este proyecto se utilizó 172.88.1.118:61505, como se observa en la figura 12.

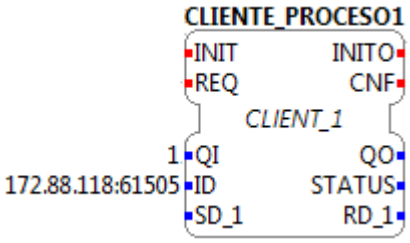


Figura 12 Bloque Funcional CLIENT
Elaborado por: El investigador

PUBLISH (MQTT)

En la comunicación con el ordenador que tiene Unity 3D, se utilizó el protocolo MQTT, para esto desde la aplicación creada forte se implementó un bloque funcional "PUBLISH", configurado para enviar información al bróker mediante la configuración "raw [].mqtt[tcp://172.88.1.104:1883, forte, dato]**,** como se puede apreciar en la figura 13, el dato que se publica al bróker es el que se encuentra en la entrada "SD_1", se envía el dato en el momento que se active el REQ, en comunicación con Unity 3D se puede observar de mejor manera el funcionamiento de esta conexión.

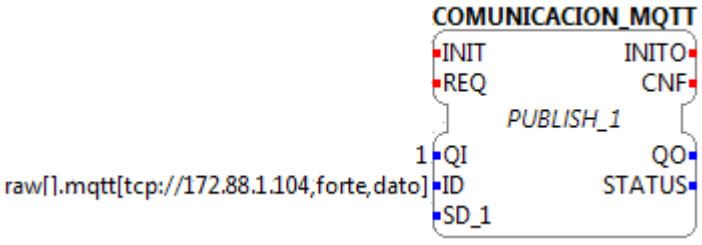


Figura 13 Bloque Funcional PUBLISH MQTT
Elaborado por: El investigador

Diseño de la aplicación de control

El diseño de la aplicación de control se realizó en el software 4DIAC-IDE, en la que se utilizó los bloques funcionales creados (Proceso), los de comunicaciones descritos anteriormente y algunos proporcionados por el software, como se observa en la figura 14.

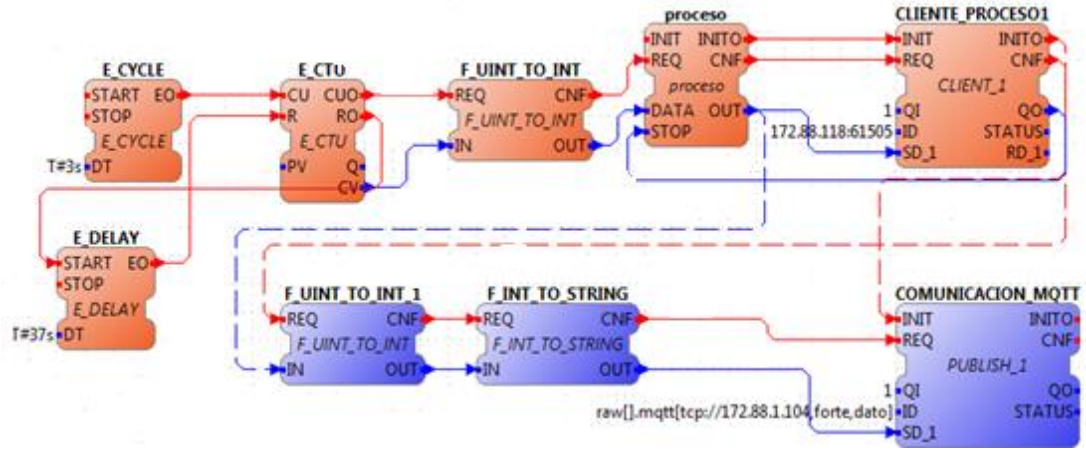


Figura 14 Aplicación de control en 4DIAC

Elaborado por: El investigador

La aplicación de control se distribuye en dos etapas, cada etapa estará configurado en un dispositivo, la primera etapa es donde se encuentra el bloque funcional PROCESO, y se encarga de obtener los datos y enviar al servidor configurado en Arduino, como también enviar los datos a la segunda etapa, para la comunicación entre estas dos etapas, utlizamos los Bloques funcionales PUBLISH/SUBSCRIBE **(**figura 15**)**, a continuacion se explica la configuracion y el diseño de la etapa 1 de la apliacion de control.

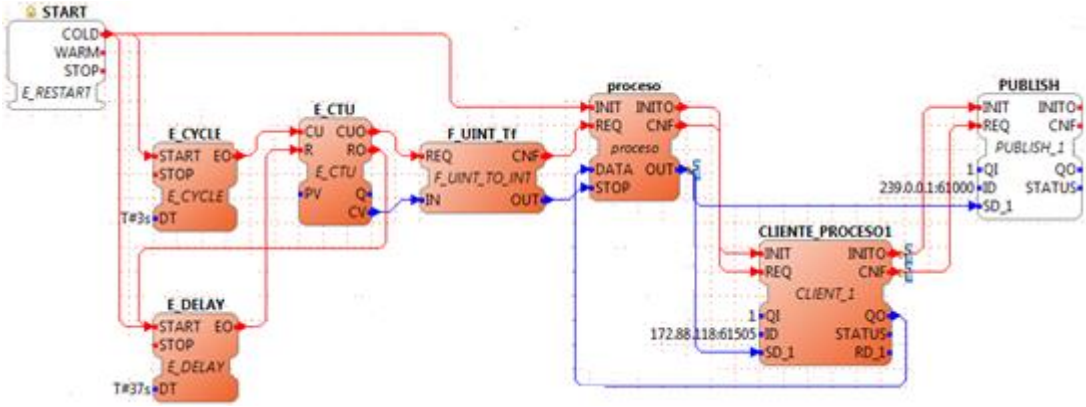


Figura 15 Primera etapa de la aplicación de control

Elaborado por: El investigador

En el diseño de la etapa 1 consta la unión de bloques funcionales, un FB START, que alimenta a los bloques funcionales al encenderse el sistema, mediante la ayuda de un FB CYCLE, que envía un pulso cada 3 segundos, a un FB E_CTU, que es un contador, el FB E_DELAY tiene como objetivo resetear el contador , cada 37 segundos, haciendo que el contador solo envié datos a su salida del 1 al 12, el contador envía estos datos hacia el FB F_UNIT_TO_INT, este convierte a un dato de tipo **int**, este dato es enviado al FB PROCESO1, que es el bloque funcional creado anteriormente, este dato es configurado internamente por el archivo proceso.cpp, descrito anteriormente, y envía el dato de salida hacia el FB ClLIENTE_PROCESO1, este enviara el dato hacia el servidor configurado en Arduino, y también enciende un FB PUBLISH, este bloque funcional es quien recepta el dato que envía el FB_PROCESO, y reenviarlo hacia la etapa 2.

La segunda etapa es donde se encuentra el bloque funcional COMUNICACIÓN_MQTT, y se encarga de obtener los datos y enviar al broker MQTT, los cuales son receptados en el ordenador con Unity, **(**figura 16**)**, a continuacion se explica la configuracion y el diseño de la etapa 2 de la apliacion de control.

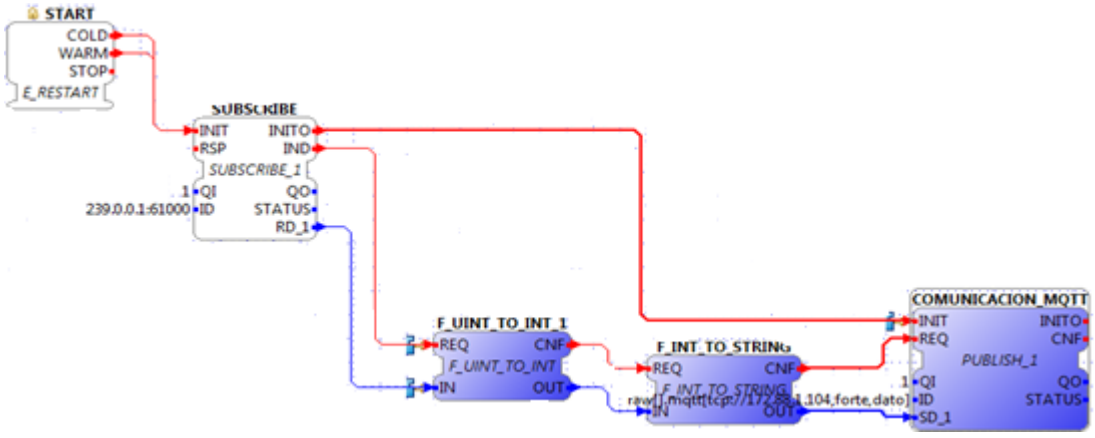


Figura 16 Segunda etapa de la aplicación de control

Elaborado por: El investigador

La segunda etapa de la aplicación de control consta de un FB SUBSCRIBE, quien recibe el dato que se envió desde la etapa 1, este recibe el dato y lo reenvía hacia un FB F_UNIT_TO _INT y convierte el dato a tipo int, este envía el dato hacia el FB

F_INT_TO_STRING, para convertir la información en tipo String, este dato es receptado por el FB COMUNICACIÓN_MQTT, quien es el encargado de enviar la información al bróker mediante el protocolo MQTT. Una vez finalizada la aplicación de control, se extrae la programación creada en 4DIAC, se exporta la programación en un archivo llamado "forte.fboot" y se colocara en la carpeta donde se encuentra la aplicación forte.

3.1.4 Creación de los Contenedores de software

En la Raspberry Pi, se instaló y configuro el software Docker (**Anexo 1**), y se creó seis contenedores, que tienen el sistema operativo Ubuntu 14.04, este sistema operativo se encuentra en el repositorio de Docker (Docker hub), los comandos para la descarga de la imagen, la construcción y la ejecución de los contenedores se lo puede apreciar en el **Anexo 2,** en cada contenedor se actualizó los paquetes necesarios, y también se instaló las herramientas como CMAKE, MQTT y 4DIAC FORTE.

Arquitectura Docker

Docker usa una arquitectura cliente-servidor. El cliente de Docker habla con el demonio de Docker que hace el trabajo de crear, correr y distribuir los contenedores, ambos pueden ejecutarse en el mismo sistema, o se puede conectar un cliente a un demonio Docker remoto, el cliente Docker y el demonio se comunican vía sockets o a través de una RESTfull API. El demonio Docker levanta los contenedores haciendo uso de las imágenes, que pueden estar en local o en el Docker Registry. Cada contenedor se crea a partir de una imagen, que es un entorno aislado y seguro, este proyecto cuenta con algunos componentes principales como [14]:

- **Docker Hub:** Plataforma de Software como servicio (SaaS, Software-as-a-Service) sirve para compartir y administrar contenedores Docker.
- **Docker Engine:** Es el demonio que se ejecuta dentro del sistema operativo (Linux) y que expone una API, para la gestión de imágenes, contenedores, volúmenes o redes.
- **Docker Client:** Cualquier software o herramienta que hace uso de la API del demonio Docker, pero suele ser el comando docker, que es la herramienta de línea de comandos para gestionar Docker Engine. Éste cliente puede configurarse

para hablar con un Docker local o remoto, lo que permite administrar nuestro entorno de desarrollo local como nuestros servidores de producción.

- **Docker Images:** Son plantillas de sólo lectura que contienen el sistema operativo base, dónde correrá nuestra aplicación, además de las dependencias y software adicional instalado, necesario para que la aplicación funcione correctamente. Las plantillas son usadas por Docker Engine para crear los contenedores Docker.
- **Docker Registries:** Los registros de Docker guardan las imágenes. Pueden ser repositorios públicos o privados. El registro público lo provee el Hub de Docker.
- **Docker Containers:** El contenedor de Docker aloja todo lo necesario para ejecutar un servicio o aplicación. Cada contenedor es creado de una imagen base y es una plataforma aislada. Un contenedor es simplemente un proceso para el sistema operativo, que se aprovecha de él para ejecutar una aplicación, dicha aplicación sólo tiene visibilidad sobre el sistema de ficheros virtual del contenedor.

Docker utiliza una estructura, como se detalla en la figura 17, en donde se observa que cada contenedor esta virtualizado con su sistema Operativo, y se puede acceder a través de la shell de Raspbian en la que llamaremos "Docker Client".

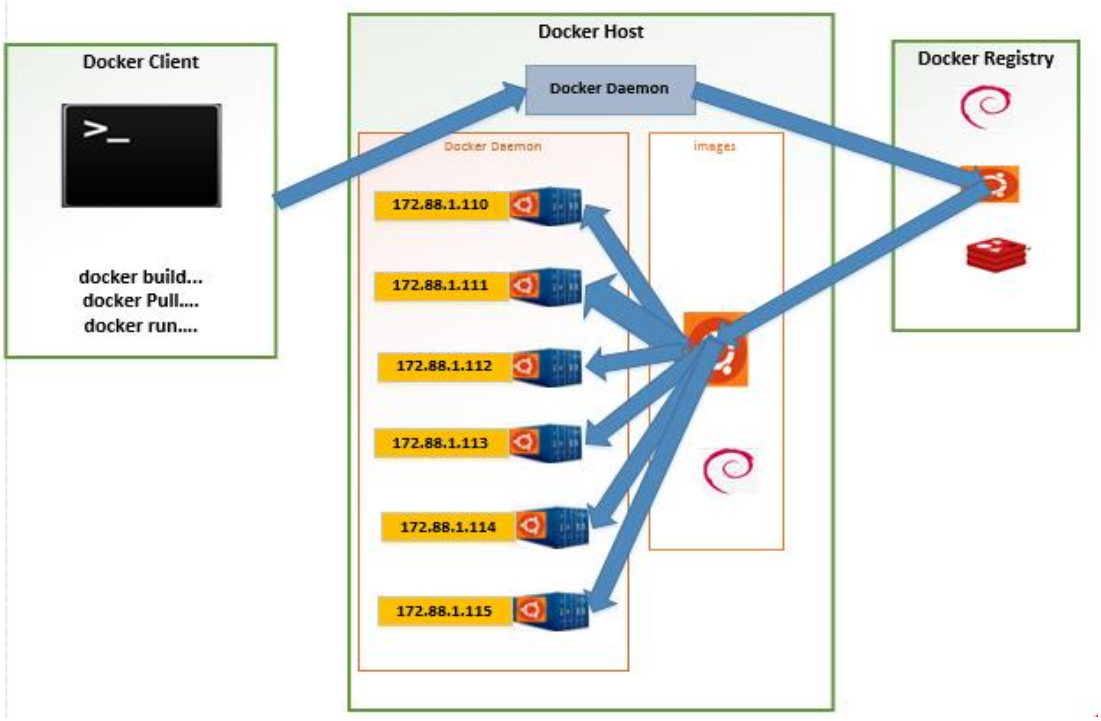


Figura17 Arquitectura de Docker
Elaborado por: El investigador

En el proyecto se tuvo que identificar a cada contenedor, para esto se utilizó un Router Tp-link, configurado previamente DHCP, en la que se habilita el rango de direcciones desde 172.88.1.100 hasta la 172.88.1.125, dentro de este rango están asignadas las direcciones a cada uno de los dispositivos, como Arduino, el ordenador con Unity 3D, la Raspberry Pi y los contenedores creados en la misma, las direcciones designadas para cada contenedor se observa en la figura 17.

Dentro de cada contendor se encuentra el directorio creado que contiene la aplicación forte, bajo la dirección "/home/", como se observa en la figura 18, en los contenedores se instaló cmake-gui, y se configuro y se generó los archivos cmake, para después compilarlos obteniendo la aplicación en el directorio como se desarrolló en un la etapa de creación de la aplicación forte.

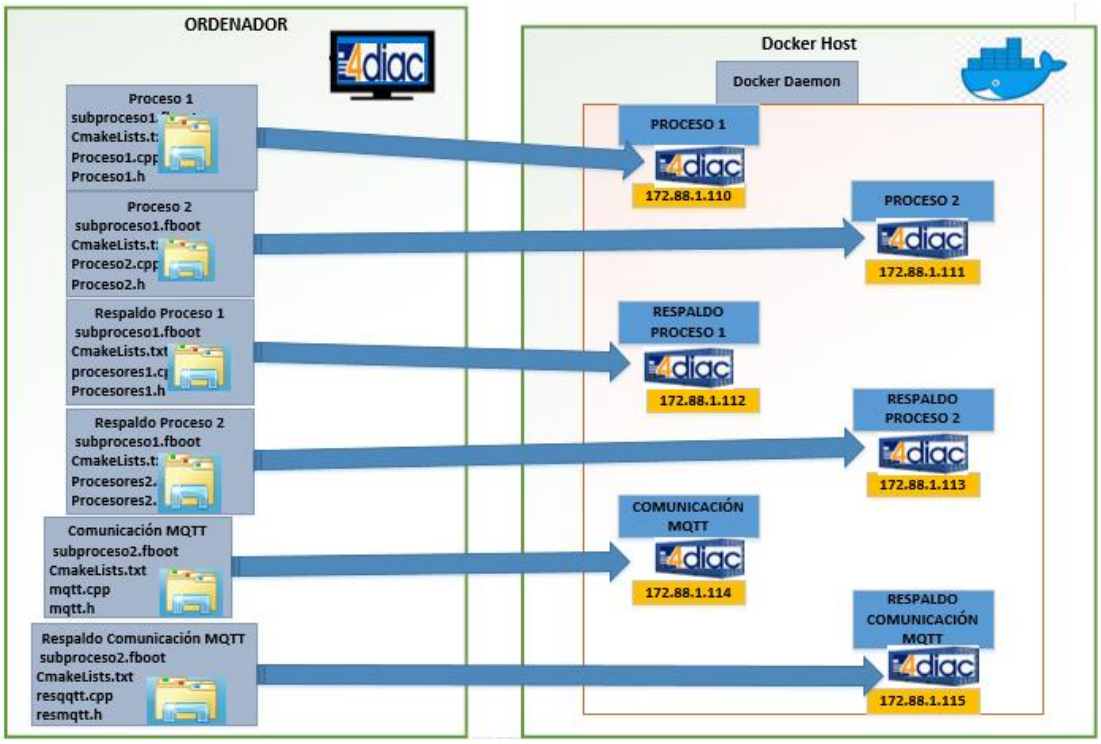


Figura 18 Exportación de los directorios a los contenedores

Elaborado por: El investigador

3.1.5 Diseño de la interfaz Web de Control

En la Raspberry Pi se encuentra los contenedores creados y listos para poder ejecutar la aplicación Forte con los procesos creados, para su ejecución se debe entrar por modo consola, o también conocida como "terminal", y ejecutar las líneas de

comandos en la shell para poder arrancar el contenedor, como también Forte, para esto se diseñó una página web, con el fin de facilitar manipulación del sistema, ya sea por el operador o la persona que requiera ejecutar el sistema, para la interfaz de la página se basó en la guía GEDIS, obteniendo un sistema de control supervisor industrial mediante criterios como: consistencia, visibilidad, perceptibilidad, informatividad, interactividad y tiempo de respuesta, para una adecuada interacción entre el usuario y el sistema.

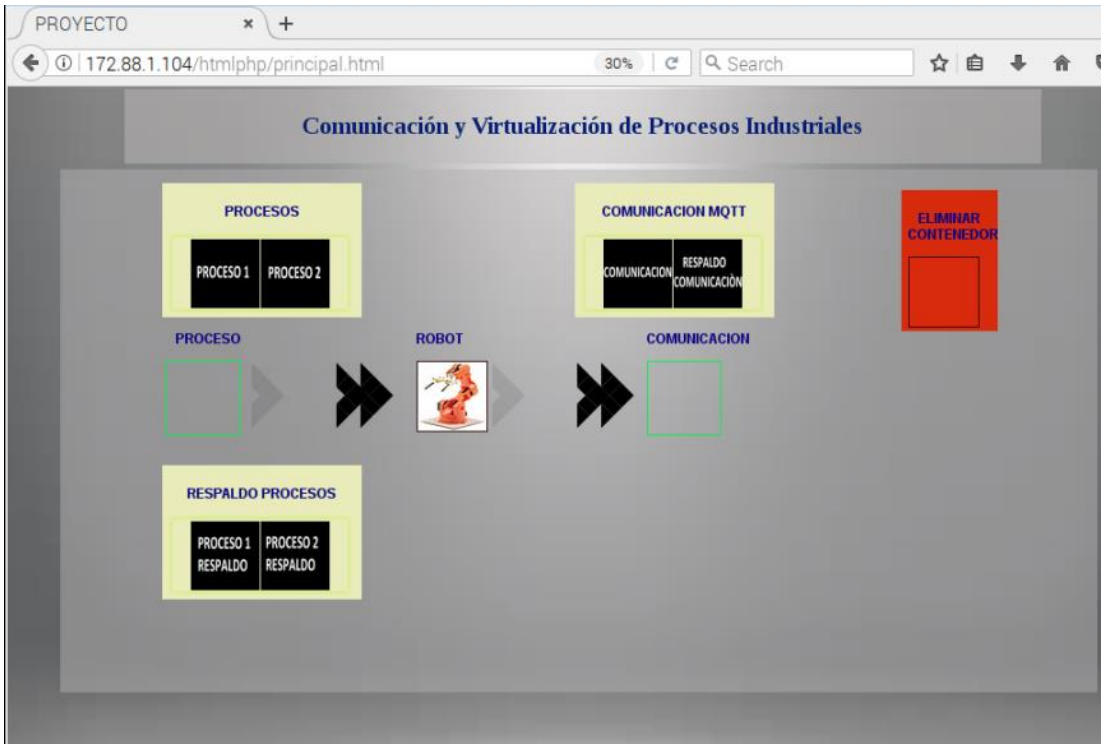


Figura 19 Diseño de la Pagina Web
Elaborado por: El investigador

Como se puede apreciar en la figura 19, se observa la interfaz de nuestra página, en donde se observa la creación de seis iconos negros, identificados como cada contenedor diferenciándolos por sus nombres, detallaremos su función a continuación:

- **PROCESO 1:** Este contenedor junto con su aplicación forte, establece una serie de datos que se envía en forma de caracteres numéricos del 1 al 16, hacia Arduino y este establece los movimientos previamente configurados para cada servomotor del prototipo de Brazo Robótico.

- **PROCESO 1 RESPALDO:** este contenedor está configurado igual que el anteriormente mencionando, y se tiene como respaldo, en caso de que el contenedor PROCESO 1 falle.
- **PROCESO 2:** Este contenedor tiene otro proceso muy diferente al PROCESO 1 al igual envía datos en forma de caracteres numéricos del 1 al 16, hacia Arduino y este establece los movimientos previamente configurados para cada servomotor del prototipo de Brazo Robótico.
- **PROCESO 2 RESPALDO:** este contenedor se tiene como respaldo del PROCESO 2.
- **COMUNICACIÓN MQTT**: Este contenedor tiene la función de receptar los datos enviados desde los contenedores PROCESO1 y PROCESO 2, y retrasmitirlos hacia al bróker utilizando protocolo MQTT.
- **COMUNICACIÓN MQTT RESPALDO:** Este contenedor es un respaldo del anteriormente mencionado.
- **PROCESOS:** Es el repositorio donde se encuentran los contenedores de los procesos.
- **RESPALDO PROCESOS:** Es el repositorio donde se encuentran los respaldos de los procesos.
- **COMUNICACIÓN MQTT:** Es el repositorio del contenedor de comunicación con el ordenador que tiene Unity 3D y su respectivo respaldo.
- **PROCESO:** En este espacio se ejecuta los comandos en los contenedores de Procesos, arrastrando hacia esta área el proceso seleccionado, arranca el contenedor y ejecuta la aplicación forte, mediante la programación realizada en php y html.
- **COMUNICACIÓN:** En este espacio se ejecuta los comandos en el contenedor de comunicación MQTT, arrastrando el proceso seleccionado hacia esta área, arranca el contenedor y ejecuta la aplicación forte, mediante la programación realizada en php y html.
- **ELIMINAR CONTENEDOR:** Arrastrando hacia esta área se elimina el contenedor.

Para arrancar el contenedor y ejecutar la aplicación Forte, se utiliza un archivo .sh que desde la configuración en Javascript, mediante la función “Ajax”, se puede

ejecutar un archivo configurado en php, y este se encarga de ejecutar los comandos del archivo .sh, mediante su nombre, para la activación se tiene que arrastrar los contendores creados representados por los nombres “PROCESO 1”, “PROCESO 2” y “COMUNICACIÓN MQTT”, hacia su área seleccionada con los nombres “PROCESO” Y “COMUNICACIÓN”, y automáticamente se ejecuta los contenedores como su aplicación Forte, al igual que va a suceder con sus respaldos, “PROCESO 1 RESPALDO”, “PROCESO 2 RESPALDO” y “COMUNICACIÓN MQTT RESPALDO”, como se puede apreciar gráficamente en la figura 20.

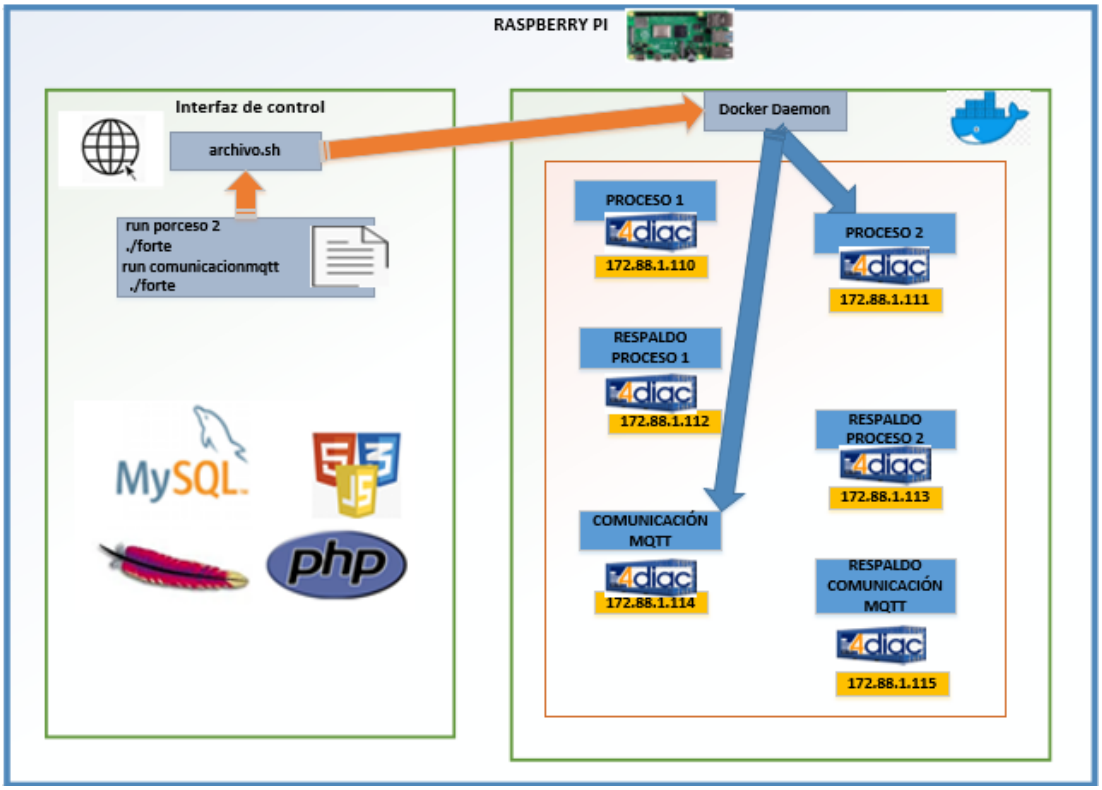


Figura 20 Estructura del funcionamiento de la interfaz de control con Docker

Elaborado por: El investigador

El acceso a la página web se lo puede realizar desde cualquier ordenador que tenga el navegador Firefox, y se conecté a la red creada, el router usa DHCP y le asigna una dirección en el rango configurado, se puede conectar inalámbricamente o por cable Ethernet, mediante la dirección de la Raspberry PI y la dirección donde se encuentre el archivo “principal.html”, una vez que se acceda a la interfaz se puede controlar el

sistema, a continuación en la figura 21, se presenta el diagrama de clases del diseño de la diseño de la página web.

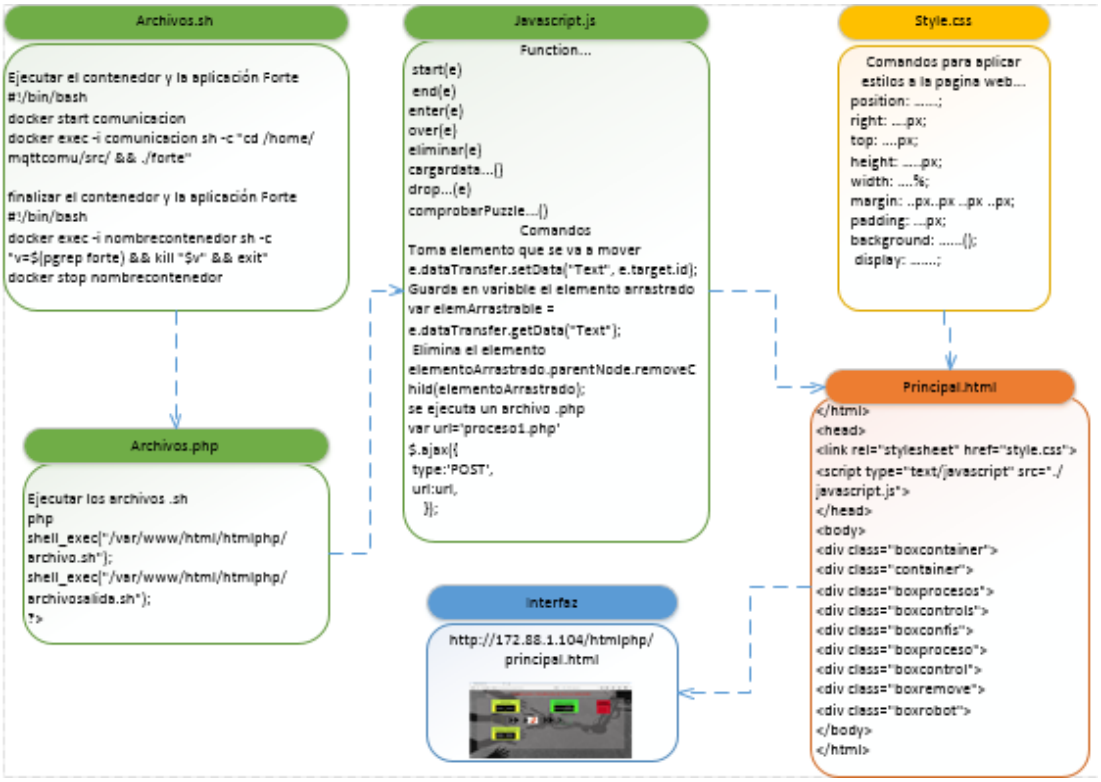


Figura 21 diagrama de clases de la configuración de la página Web

Elaborado por: El investigador

3.1.6 Comunicación entre Contenedores

Se creó una red que permite tener una dirección a cada dispositivo o contendedor para la comunicación, debido a que Docker tiene una arquitectura basado en Kubernetes (plataforma de contenedores y servicios), y cuenta con un dispositivo virtual “flannel” (crea una subred única para los contenedores), estos aspectos hacen que los contenedores solo acepten ciertos protocolos de comunicación como UDP, VxLAN, AWS, VPC y GCE. El protocolo de comunicación TCP/IP no se encuentra disponible para docker, ni los protocolos que ocupen esta conexión, en la figura 22 se detalla la forma de comunicación entre contenedores.

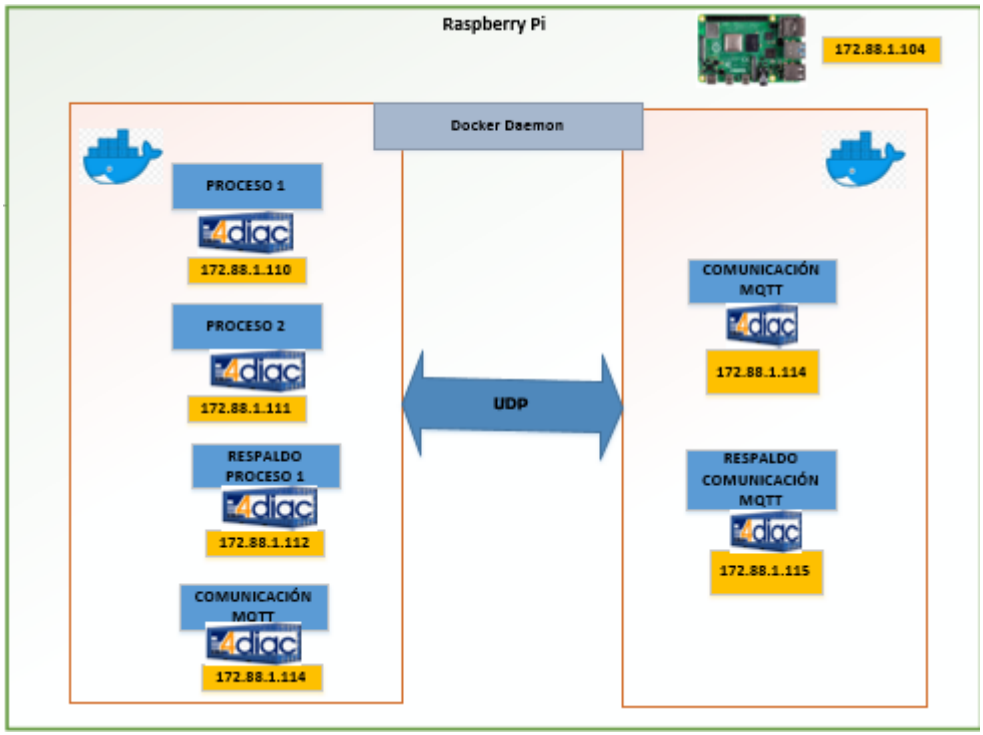


Figura 22 Comunicación entre contenedores

Elaborado por: El investigador

Existen dos grandes grupos de protocolos de comunicación en la industria, en su forma de conexión, en un capitulo anterior ya se los mencionó, hablamos del grupo de PUBLISH/SUBSCRIBE, en otras palabras un dispositivo se convierte en el publicador de la información a compartir, y el otro dispositivo es quien se conecta y recibe la información, muchos protocolos se basan en esta estructura, como MQTT, UDP, etc., para la comunicación entre contendores se utilizó una conexión UDP, mediante la ayuda de los bloques funcionales PUBLISH/SUBSCRIBE (ver figura 11) tratados anteriormente.

3.1.7 Comunicación Arduino y con los contenedores

En la conexión de los contenedores con Arduino, se utilizó la otra categoría de protocolos de comunicación como es CLIENT/SERVER, este requiere que los clientes se conecten al servidor y este administra la información, al proporcionar una dirección fija al Arduino, quien en este caso se configuro como servidor “172.88.1.118”y cumpliendo con el estándar IEC-61449, se utiliza el puerto 61505, para la comunicación, la configuración de esta placa se puede apreciar en el **anexo 5**.

El bloque funcional utilizado se puede observar en la figura 12, la configuración realizada en 4DIAC (ver Figura 15), envía los datos al servidor, mediante su dirección, y por el puerto que se comunica, el servidor está configurado en Arduino dependiendo del dato que recibe, este configura y envía la orden hacia los pines analógicos, los cuales accionaran los servomotores del Brazo Robótico, según la orden dispuesta desde el proceso seleccionado.

Según el proceso seleccionado cada tres segundos se ejecuta una acción en el prototipo, su conexión se puede ver en la figura 23 utilizando los pines 2, 4, 6 y 8, cada pin controla a un servomotor accionando según los datos que se obtuvo del proceso seleccionando.

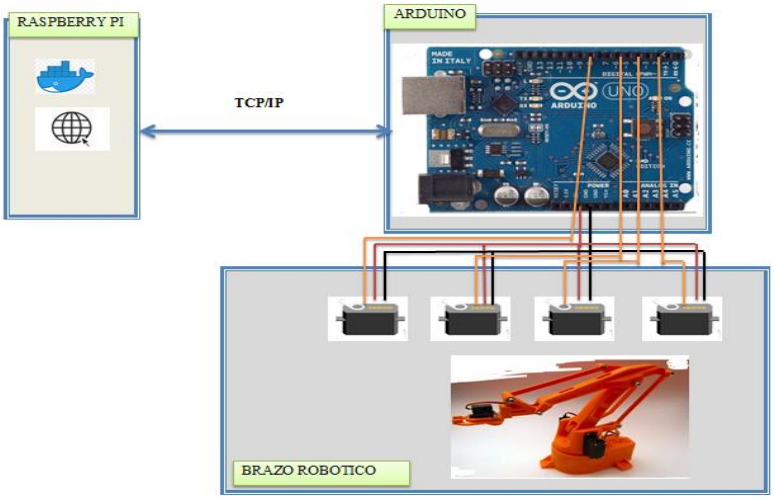


Figura 23 Comunicación Arduino con la Raspberry Pi
Elaborado por: El investigador

3.1.8 Comunicación de los contenedores con Unity

Se diseñó un modelo en 3D del prototipo del brazo Robótico, para la visualización de los movimientos que son realizados por el prototipo, para eso se instaló el software Unity 3D en un ordenador con sistema operativo Windows 10, el cual está conectado de igual manera a la red del sistema, el protocolo que utilizamos es MQTT, teniendo ventajas como seguridad y encriptación al enviar los datos, la configuración del brazo robótico y la habilitación del protocolo MQTT se puede apreciar en el **anexo 6**.

La Raspberry Pi es quien recibe la información mediante su dirección “172.88.1.104” actuando como bróker, para esto utilizamos el protocolo de comunicación Mosquitto, para la instalación y configuración de MQTT en la Raspberry Pi, primero actualizamos el sistema de la Raspberry Pi.

```
sudo apt-get update
```

Para instalar el software MQTT requerido, necesita instalar mosquitto.

```
sudo apt-get install –y mosquito mosquito-clients
```

Con las bibliotecas instaladas y la Raspberry Pi configurada. Podemos utilizar la Raspberry Pi como bróker, para obtener la información que los publicadores enviar al mismo, y este administra y reenvía la información a los subscriptores.
Para hacer que Mosquitto se inicie automáticamente al iniciar, se ingresa la siguiente línea de comandos:

```
sudo systemctl enable mosquitto.service
```

Una vez instalado y configurado MQTT en la Raspberry Pi, la información llega al bróker, cuando se habilite el contenedor de “COMUNICACIÓN”, o como su respaldo, dentro de la programación en 4DIAC encontramos el bloque funcional “PUBLISH”, configurado como se puede observar en la figura 13, esta configuración nos ayuda a enviar la información al bróker, donde está esperando que algún subscriptor requiera la información, mediante su tópico seleccionado “hello” y con identificación de la ID del cliente que es “forte” (ver Figura 16), a continuación en la figura 24, se observa el diagrama de clases de la configuración y diseño del brazo Robótico en 3D.

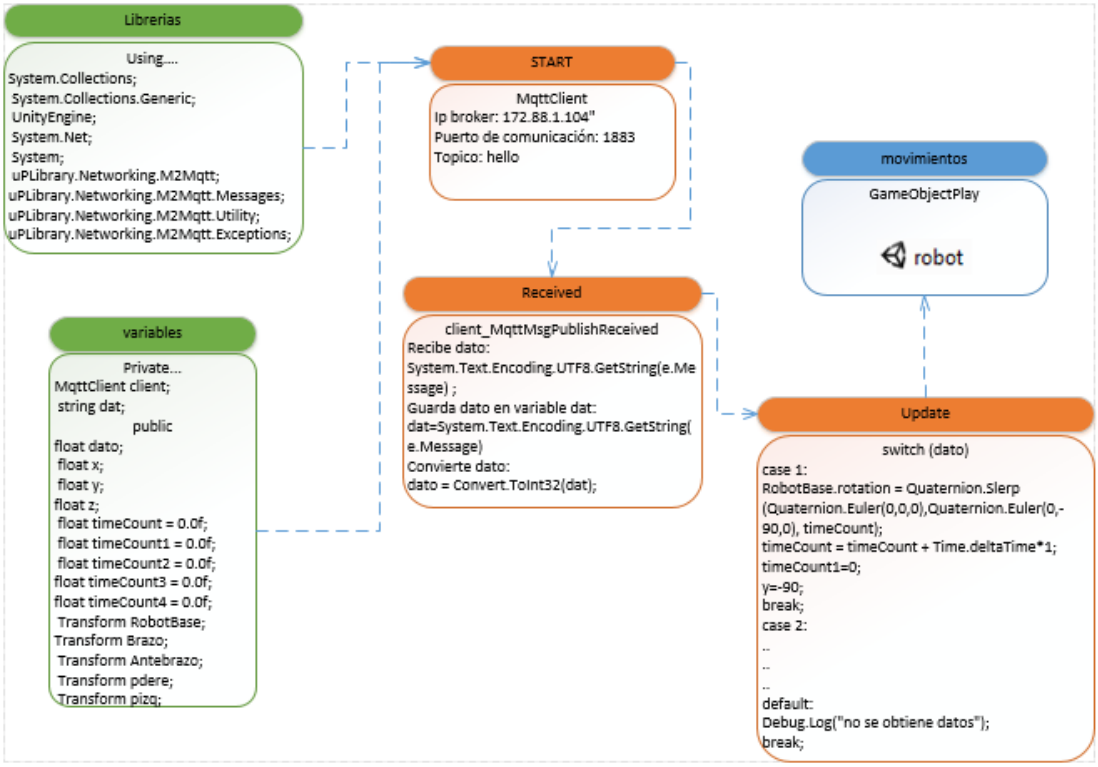


Figura 24 Diagrama de clases de MQTT configurado en Unity

Elaborado por: El investigador

Unity 3D ofrece la posibilidad de comunicarnos mediante el protocolo MQTT, al recibir los datos desde los contenedores al ordenador, este los configura y ejecuta las acciones, que el robot este realizando, así se obtiene la visualización del sistema, que se puede visualizar desde cualquier ordenador que esté conectado a la red mediante la herramienta Unity 3D, con su respectiva configuración.

Figura 25 Interfaz de visualización de Unity 3D

Elaborado por: El investigador

3.2 Presupuesto del Prototipo

Para determinar el costo total de la propuesta de investigación se deben considerar dos aspectos: el presupuesto de diseño y el presupuesto de ensamblaje del prototipo. Para el presupuesto de diseño se investigó el salario básico de un Ingeniero en Electrónica determinado por el Ministerio de Trabajo correspondiente a 424.15 dólares mensuales [32]. Considerando un promedio de 21 días laborables durante cada mes y aplicando la ecuación 1 se obtiene el salario diario.

$$Salario_{diario} = \frac{Salario_{mensual}}{Dias_{laborables}} \quad (1)$$

$$Salario_{diario} = \frac{424.15}{21}$$

$$Salario_{diario} = 20.20[dólares]$$

Es conocido que el día está constituido de 8 horas laborales, aplicando la ecuación 2 se obtiene la remuneración por hora de trabajo.

$$Salario_{hora} = \frac{Salario_{diario}}{Horas_{laborables}} \quad (2)$$

$$Salario_{hora} = \frac{20.20}{8}$$

$$Salario_{hora} = 2.53[dólares]$$

Se estiman 100 horas de investigación empleadas para el diseño, simulación y pruebas de funcionamiento; aplicando la ecuación 3 se obtiene el presupuesto de diseño del proyecto de investigación.

$$Presupuesto_{diseño} = Horas_{investigacion} * Salario_{hora} \quad (3)$$

$$Presupuesto_{diseño} = 100 * 2.53\ [dólares]$$

$$Presupuesto_{diseño} = 253[dólares]$$

El presupuesto para el ensamblaje del prototipo es importante, debido a que se utiliza equipos de hardware y software libre, y materiales de uso comercial, el costo final del diseño de la implementación del sistema es bajo, comparado con los sistemas que se encuentran en el mercado y que se utilizan en las empresas industriales. En la tabla siguiente se muestra los costos de los equipos y materiales empleados para el prototipo del proyecto desarrollado.

Tabla 7: Presupuesto total de ensamblaje del prototipo

Presupuesto					
Item	**Unidad**	**Descripción**	**Cantidad**	**valor unitario**	**Valor Total**
1	c/u	Raspberry Pi3 B+(Kit completo)	1	$ 120	$ 120
2	c/u	Router Tp-Link TL-WR741ND (Kit completo)	1	$ 30	$ 30
3	c/u	Arduino UNO R3 + Cable USB	1	$ 12	$ 12
4	c/u	Monitor Computador Monitor Pc Lg 19,5 Pantalla Hd Cpu Dvr	1	$ 90	$ 90
5	c/u	Kit de teclado y mouse Inalámbrico	1	$ 20	$ 20
6	c/u	Convertidor de HDMI a VGA	1	$ 20	$ 20
7	c/u	Memoria MicroSD	1	$ 25	$ 25
8	c/u	Cable de red	4	$ 3	$ 12
9	c/u	Prototipo del Brazo Robótico	1	$ 120	$ 120
10	Horas	Internet	200	$ 0,5	$ 100
11	c/u	Kit de equipamiento electrónico	1	$ 40	$ 40
12	c/u	Kit de herramientas de precisión	1	$ 20	$ 20
13	c/u	Otros(Materiales para la elaboración del prototipo)	1	$ 16,8	$ 16,8
Presupuesto total del proyecto					**$ 625,8**

Elaborado por: El Investigador

3.3 Análisis y Discusión de los Resultados

Este capítulo detalla los resultados obtenidos durante la investigación para el desarrollo del proyecto, se realizó un análisis de las muestras de los tiempos tanto del arranque del sistema de un contenedor como del arranque de una máquina virtual, permitiendo la comparación y evaluación de las mismas, también se analizó los resultados para el apagado de los contenedores. Además se muestra el desempeño que tiene la Raspberry Pi, con respecto al consumo de la CPU cuando el sistema se está ejecutando y los resultados finales del proyecto realizado.

3.3.1 Análisis del arranque de los contenedores

Se tomaron veinte muestras de los tiempos de arranque de los contenedores y conjuntamente con la ejecución de forte, se tomó el tiempo desde que se manda a ejecutar el proceso en la página web, hasta que el brazo robótico comienza a realizar los movimientos, el tiempo en enviar los datos para cada ejecución es de tres segundos, tiempo suficiente para el movimiento de cada articulación del brazo robótico.

En la figura 26 se detalla las muestras tomadas de los tiempos de arranque que se demoró en ejecutar el proceso.

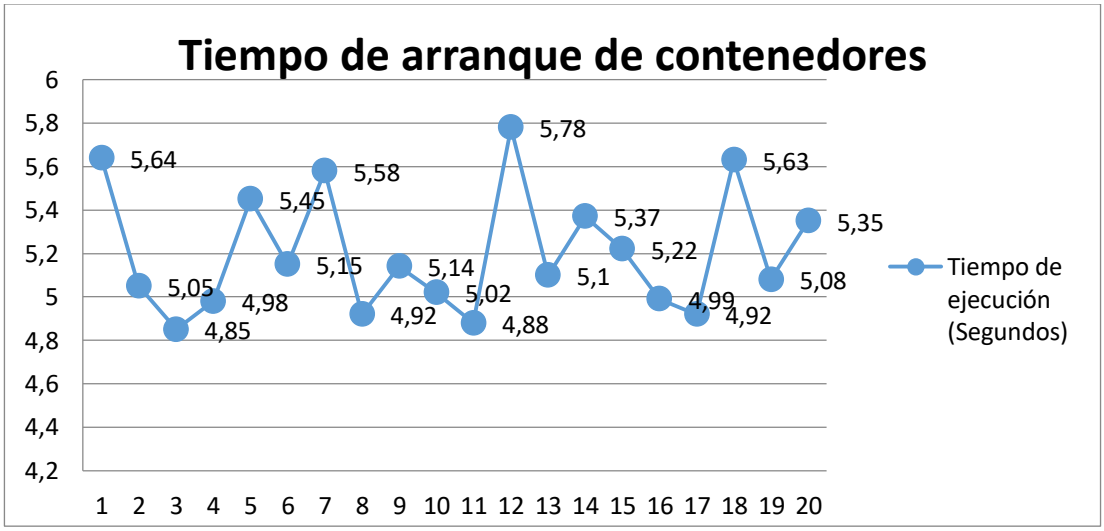


Figura 26: Tiempos obtenidos en la ejecución del proceso forte y el arranque del contenedor

Elaborado por: El investigador

Media aritmética

La media aritmética sirve para obtener una medida de posición central de un conjunto de valores con la siguiente formula.

$$Media(x) = \bar{x} = \frac{\sum_{i=1}^{N} \mathrm{X}\, i}{N} \qquad (4)$$

Dónde:

$\bar{x}$ = Media aritmética de la muestra

X i= valores de cada muestra tomada

N= Cantidad de muestras tomadas

Mediante la ecuación 4 se obtiene la media de los valores de la tabla 4, dándonos un valor:

$$\bar{x} = 5{,}0635 \text{ (Segundos)}$$

Este tiempo estimado que se requiere para el arranque del contenedor, con respecto al de una máquina virtual es más rápido, esto se debe a las diferentes tecnologías ocupadas como Docker y Virtual Box, respectivamente, pero también esta velocidad se obtuvo debido a las buenas características que cuenta el modelo de la Raspberry PI3 B+, teniendo en cuenta que tiene un procesador que corre a 1.4GHz, además la comunicación del sistema es por medio del puerto Ethernet que es de alta velocidad, puede alcanzar hasta 300 Mbps (megabits por segundo) en su capacidad de procesamiento teóricamente.

3.3.2 Análisis de protocolos de Comunicación

En el trascurso de la prueba realizada, la comunicación UDP se utilizó como protocolo en la comunicación entre contenedores, teniendo en cuenta que por la estructura de Docker no acepta una conexión TCP/IP, al tener los contenedores virtualizados el tráfico de datos resulto excelente sin perder ningún dato en la prueba realizada, mediante la herramienta “tcpdump” disponible para el sistema operativo UNIX, permitió realizar un análisis del tráfico que circula a través de nuestra red, permitiendo capturar en tiempo real los paquetes que circulen por ella, en la figura 27, se puede observar el tráfico de datos obtenidos del contenedor Proceso 1 en

donde se está ejecutando la aplicación Forte, y se envía los datos hacia el contenedor Comunicación, cada paquete es enviado cada tres segundos.

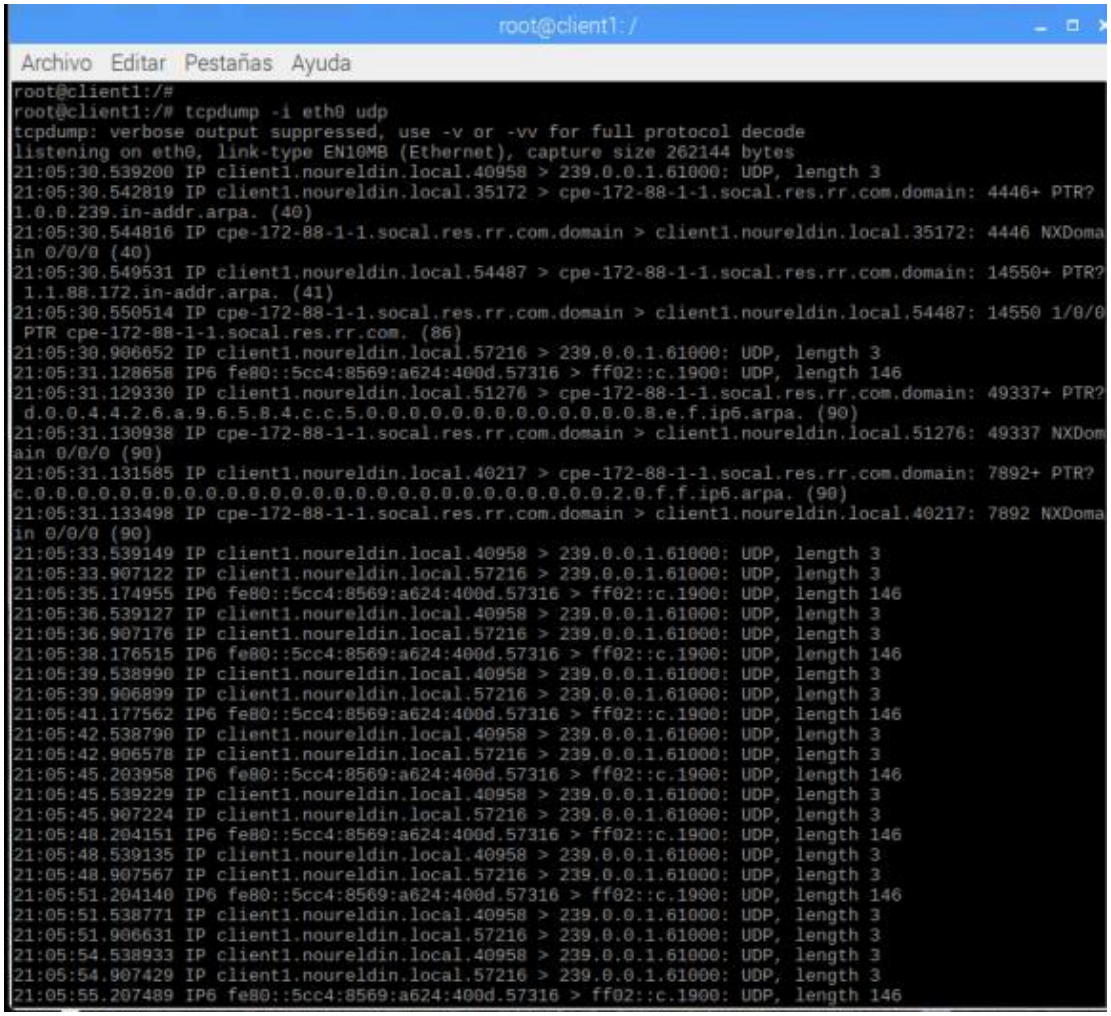


Figura 27 Tráfico de datos UDP (Contenedor Proceso 1)

Elaborado por: El investigador

También se realizó un análisis del tráfico que circula a través del contenedor Comunicación, en la figura 28 se puede observar la captura de los tiempos, en donde cada tres segundos se recibe un paquete que es enviado desde la aplicación Forte ejecutándose en el contenedor Proceso 1, al comparar los tiempos de envío y recepción de los paquetes que circula entre los contenedores, se observa un retardo de que varía entre los 40 a 60 microsegundos.

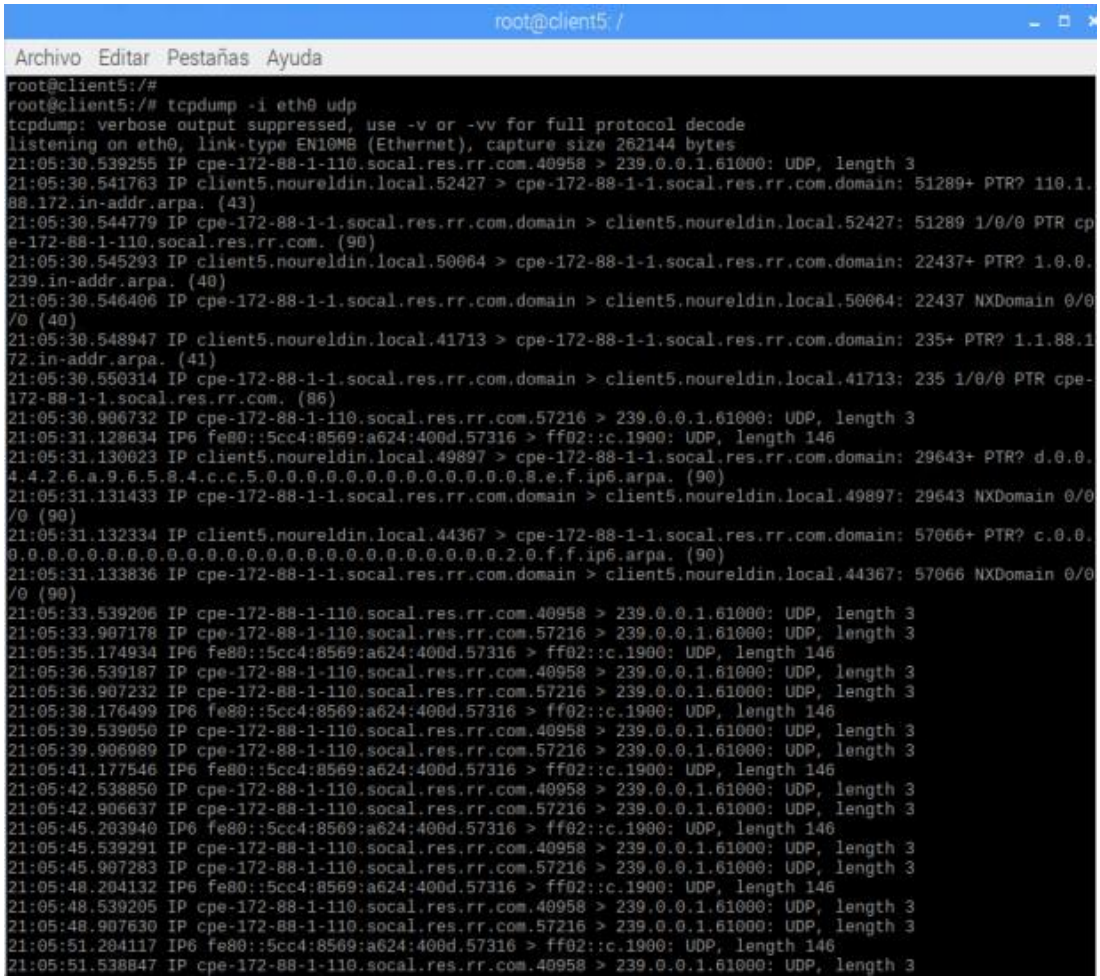


Figura 28 Tráfico de datos UDP (contenedor Comunicación)

Elaborado por: El investigador

En la comunicación del sistema con Arduino se utilizó el protocolo TCP/IP, aportando beneficios tanto en el envío y recepción de los paquetes que son trasmitidos desde la aplicación Forte de los contenedores Proceso 1, proceso 2, como sus respaldos, y estos son receptados por arduino, en la figura 29 se observa él envió de los paquetes cada tres segundos hacia el servidor configurado en Arduino mediante la dirección “172.88.1.118”, los datos son enviados desde la aplicación del contenedor Proceso 1, en este caso esta con el seudónimo de cliente1, envía los datos hacia Arduino por el puerto 61505, que es uno de los puertos de comunicación que aporta el software 4DIAC, según el estándar IEC-61499 .

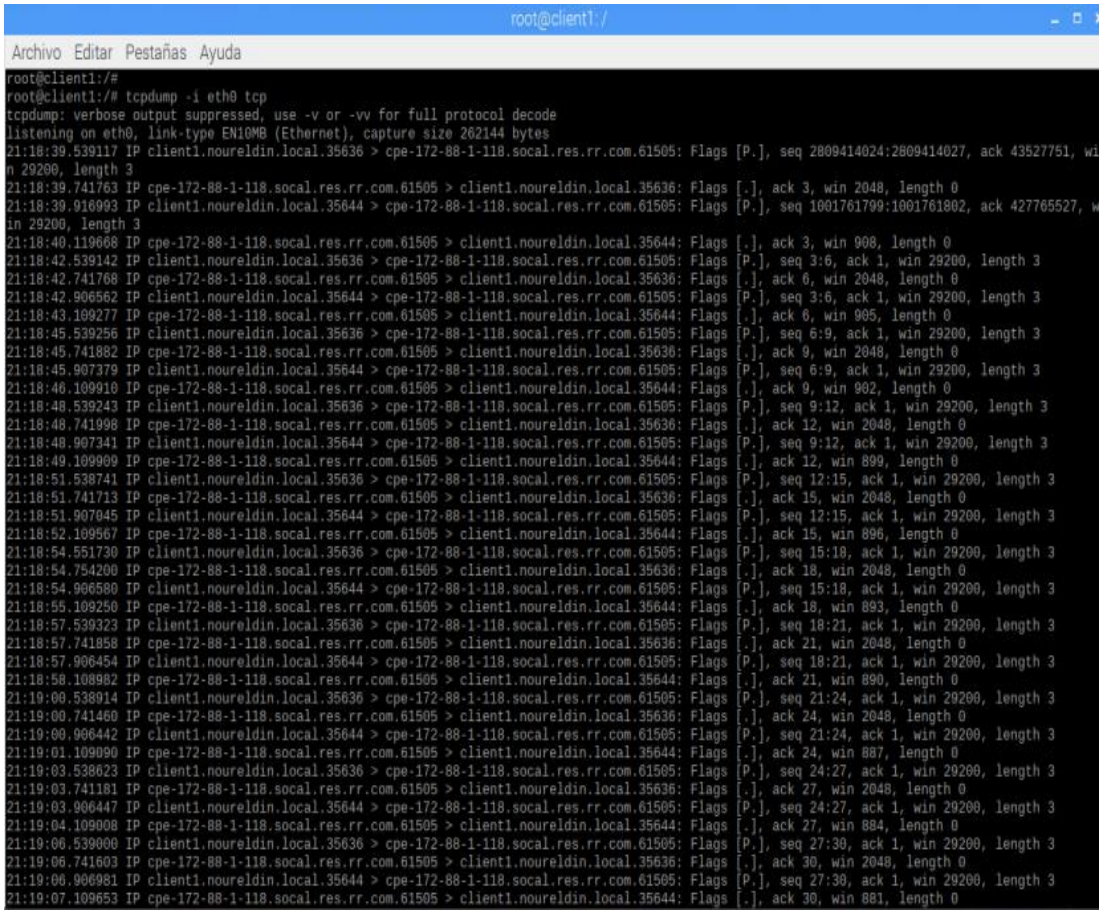


Figura 29 Tráfico de datos TCP/IP (Contenedor Proceso 1)

Elaborado por: El investigador

Para la envió de información entre la aplicación Forte del contenedor Comunicación como el de su respaldo, se escogió el protocolo MQTT, teniendo varias ventajas con respecto a otros protocolos, como la seguridad de los datos entre las más importantes, el receptor de los datos es un ordenador que previamente se instaló UNITY, y se creó un modelo en 3D de nuestro prototipo de brazo robótico, en la figura 30 se observa el tráfico de datos del contenedor Comunicación, en donde se muestra él envió de los paquetes cada 3 segundos hacia el bróker, que es este caso está configurado en la Raspberry Pi, con dirección 172.88.1.104, que se refiere al contenedor comunicación, para la conexión se utiliza el puerto 1883.

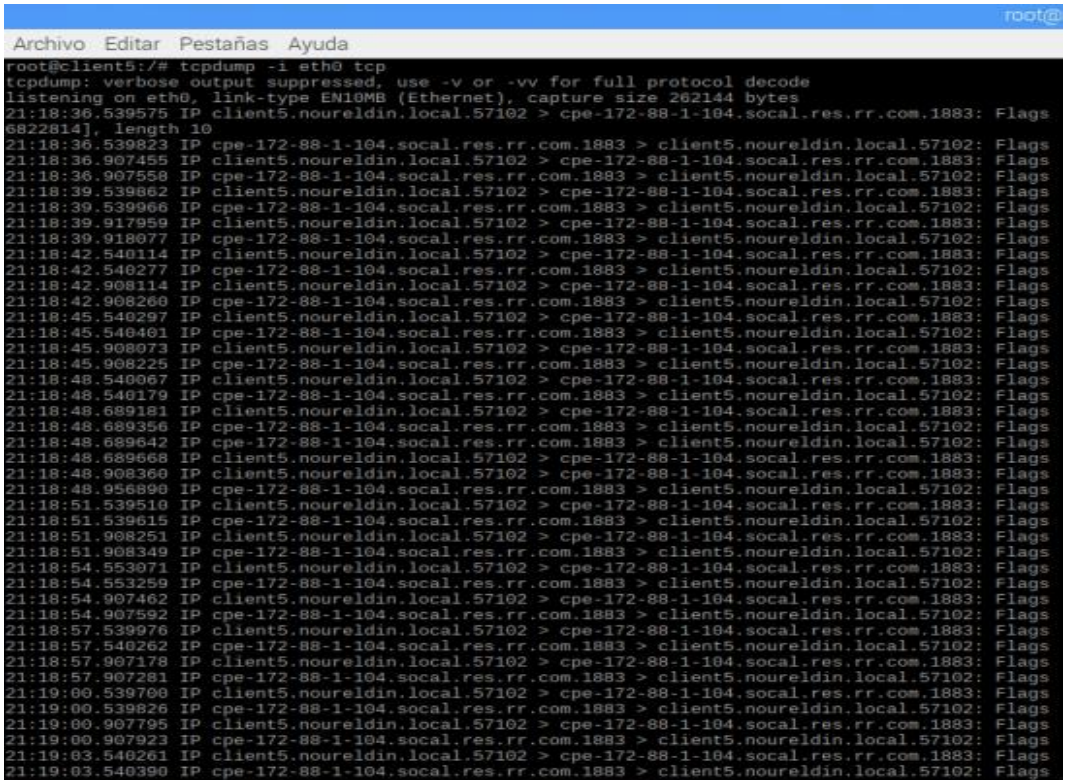


Figura 30 Tráfico de datos MQTT (Contenedor Comunicación)

Elaborado por: El investigador

En Unity se configuro el protocolo MQTT, como subscriptor para receptar los datos, que son publicados en bróker, en este caso será la dirección de la Raspberry Pi, en la figura 31 se observa la recepción de los datos en el software Unity, para configurarlos y enviar las acciones que debe hacer el modelo en 3D realizado.

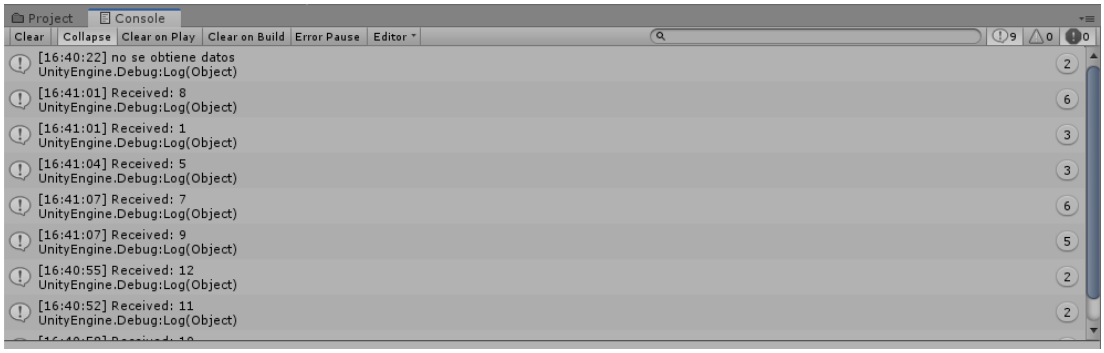


Figura 31 Recepción de datos MQTT (Unity 3D)

Elaborado por: El investigador

Los protocolos utilizados en cada caso resulto eficaz en la pruebas realizadas, se obtuvo una comunicación optima en el sistema, recurriendo a las fortalezas de cada protocolo según requiere el sistema, tanto en la velocidad de envió de información, como en la confiabilidad de entrega de los datos y la seguridad respectivamente.

3.3.3 Desempeño de la Raspberry PI con el sistema

La Raspberry PI puesta en marcha realiza la ejecución de la página web desde la cual se pone en marcha los dos contenedores tanto del proceso que envía los datos hacia el contenedor de comunicación como del que envía datos mediante el protocolo MQTT hacia el ordenador con Unity 3D, se puede observar en la figura 32 el rendimiento que tiene la CPU después 60 minutos de encender el sistema y estar ejecutándose un contenedor de proceso y el de comunicación MQTT.

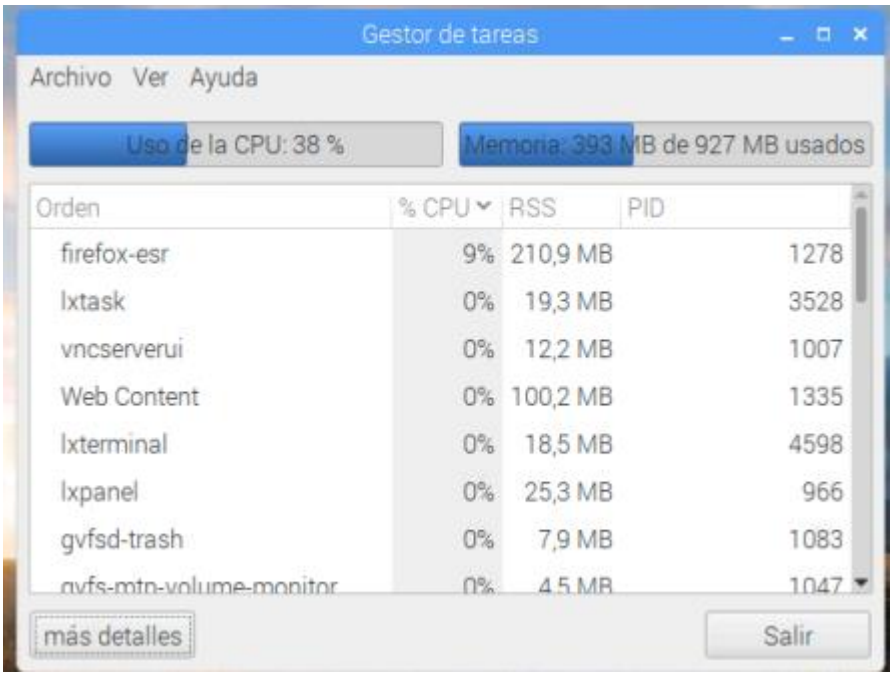


Figura 32 Gestor de tareas de la Raspberry PI3 B+.

Elaborado por: El investigador

También se puede observar en la figura 33, el consumo de la CPU de la Raspberry PI, al estar ejecutando el sistema, en el tiempo cero es donde se ejecuta el sistema las acciones razón por la cual existe un pico alcanzando el 61% de consumo de la CPU, pero después se estabiliza entre los porcentajes de 37 a 39%, ocupando un porcentaje moderado para el rendimiento del sistema.

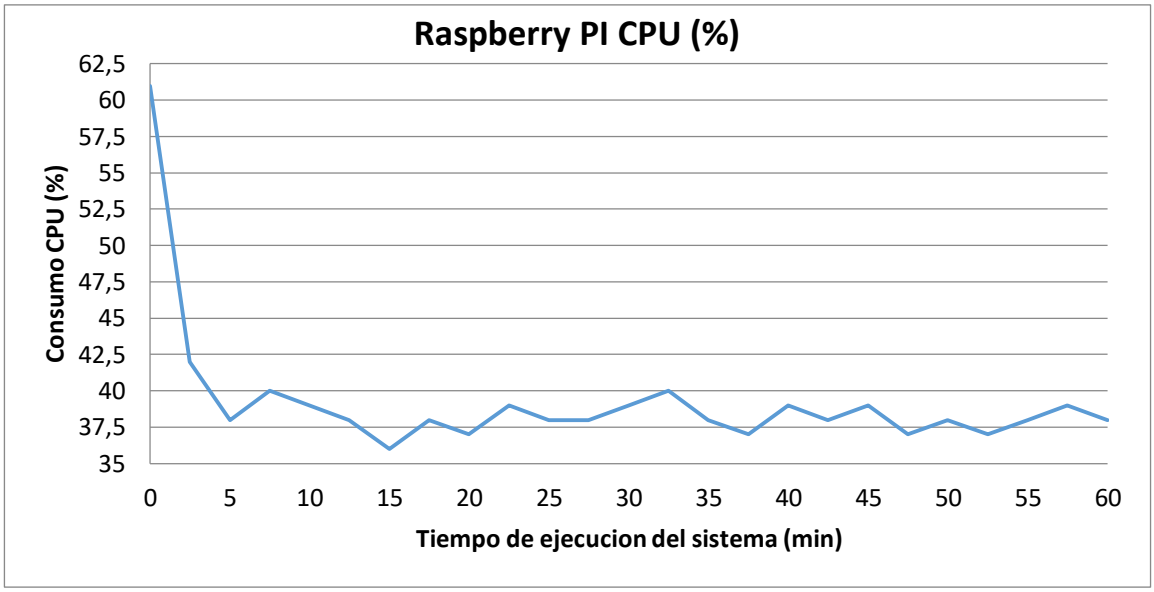


Figura 33 Grafica del consumo de la Raspberry PI3 B+

Elaborado por: El investigador

3.3.4 Resultado final del Proyecto

En las pruebas realizadas en el Prototipo de Brazo Robótico resultaron satisfactorias al momento de ejecutar todo el sistema, en la figura 34 se puede observar la etapa de control mediante la página web desarrollada, ejecutando el Proceso1 y habilitando el contenedor para una comunicación MQTT con el ordenador que tiene el diseño Robótico en 3D del prototipo.

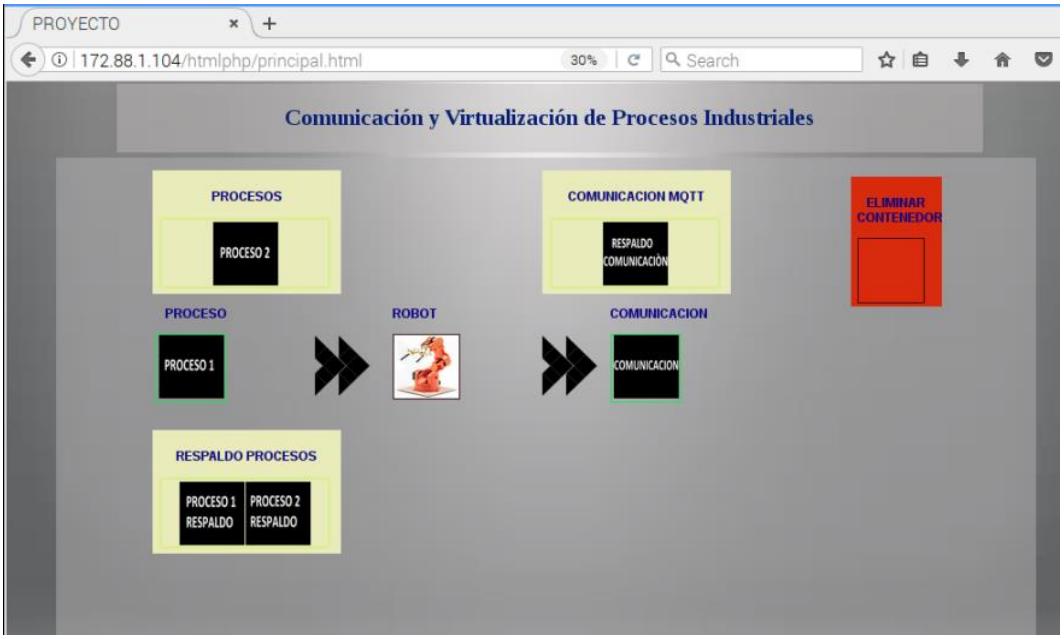


Figura 34 Ejecución de procesos

Elaborado por: El investigador

En la figura 35 se puede observar el diseño del brazo robótico en el software Unity 3D, manteniendo una posición inicial, hasta recibir los datos desde el contenedor de comunicación MQTT.

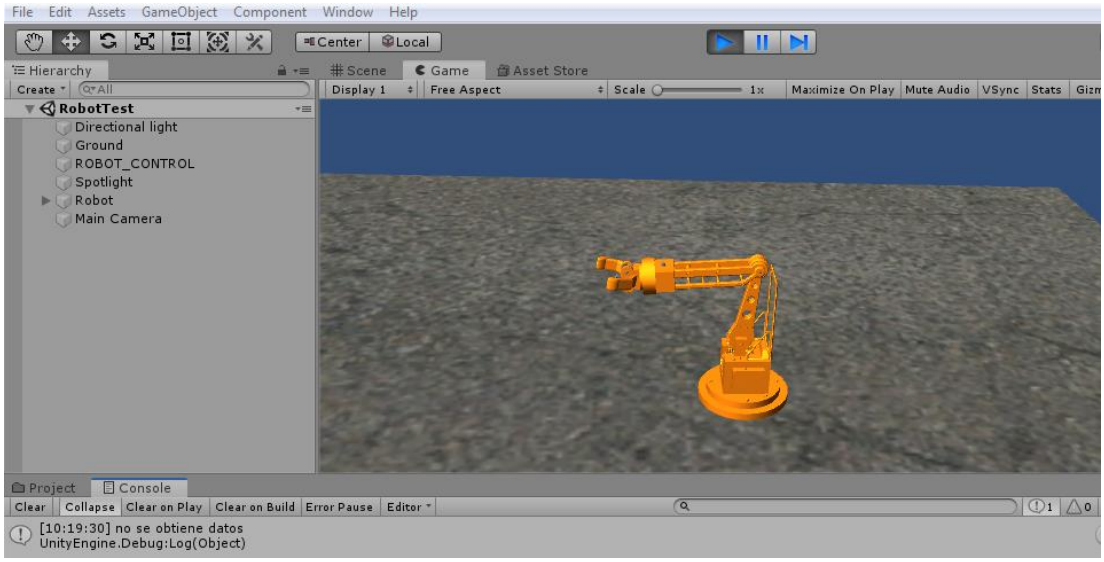


Figura 35 Posición inicial del Brazo Robótico en Unity 3D

Elaborado por: El investigador

En la siguiente figura se muestra los dispositivos Arduino y la Raspberry pi, conectados al router Tp-Link, mediante una conexión Ethernet.

Figura 36 Conexión de Dispositivos

Elaborado por: El investigador

En la siguiente figura se puede observar la implementación y la ejecución del sistema comprobando su eficaz rendimiento.

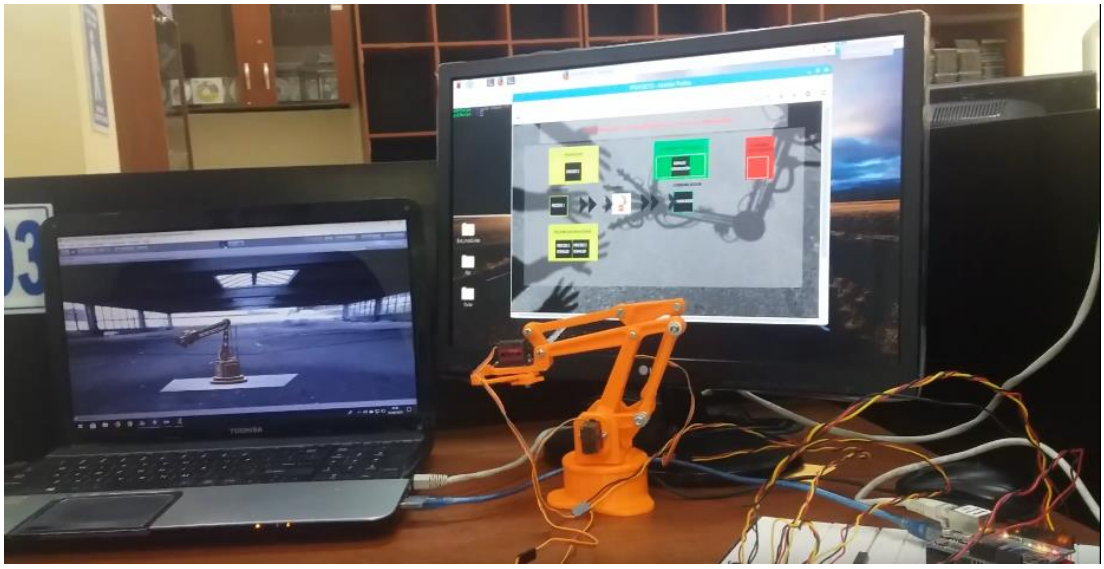


Figura 37 Implementación del sistema

Elaborado por investigador

El desarrollo del proyecto cumplió con los objetivos esperados, obteniendo un sistema que se basa en la virtualización de contenedores de software, migrando hacia un mundo digital alcanzando los objetivos de la era industrial 4.0, mediante la implementación del prototipo basado en el estándar IEC-61449, se pudo evidenciar el eficaz rendimiento del sistema propuesto, tanto en la ejecución de los procesos, como en la conectividad de los diferentes dispositivos. El presente proyecto puede convertirse como parte de nuevos proyectos o investigaciones para alcanzar nuevos horizontes hacia una digitalización de empresas industriales acarreando grandes beneficios a las mismas.

CAPÍTULO 4
CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El rendimiento de la tarjeta Raspberry Pi es eficiente en la implementación y la ejecución del sistema, teniendo varias tareas ejecutándose al mismo tiempo, como sus contenedores (Docker), obteniendo ventajas grandes a diferencia de otras tecnologías de virtualización de software como puede ser máquinas virtuales (Virtual Box), a partir de la estructura de Docker se obtuvo una optimización de la memoria de almacenamiento de la PI, mediante su forma de gestionar los recursos a todos los contenedores según su preferencia, optimizando la capacidad de procesamiento del dispositivo.

- En el área industrial se cuenta con varios protocolos de comunicación, de los cuales se escogió los óptimos para el presente proyecto, permitiendo tener una velocidad y confiabilidad en envió y recepción de los datos, mediante el análisis de sus fortalezas y debilidades de cada protocolo, se comprobó que el protocolo UDP escogido para la comunicación entre contenedores es eficiente, analizando los tiempos tanto al envió como en la recepción de los paquetes, se obtuvo un retraso de microsegundos en la transportación de los datos, al ser mínimo el tiempo de retraso se obtiene una rentabilidad en el sistema, puesto que el desarrollo de este trabajo está orientado al campo industrial, donde la velocidad de trasferencia de datos debe ser eficiente para los procesos de manufacturación en la empresa.

- El control del Sistema se realizó mediante el diseño de una página web, en donde se controló la ejecución de los contenedores de una forma automática, tanto para el encendido y la ejecución de la aplicación forte creada en cada contenedor,

como para el apagado y la finalización de la aplicación, creando un entorno de control dinámico y fácil para el operador o la persona que vaya a manipular el sistema, arrastrando simplemente el proceso hacia su área de ejecución, sin tener la necesidad de escribir algún código, ejecutándose automáticamente los procesos.

- La arquitectura establecida entre los contenedores, y los dispositivos de hardware del sistema, resulto eficaz, evidenciando los resultados al realizar diferentes movimientos de un prototipo de brazo Robótico que es controlado por Arduino, quien recibe los datos que son enviados desde la aplicación Forte del contenedor que se está ejecutando, creando así un sistema con una arquitectura única, orientada a la evolución de la forma de manufacturación en las empresas industriales, fusionando tecnologías para la obtención de fábricas inteligentes, entrando con auge en la competición ante las demás organizaciones migrando hacia una empresa digitalizada en la era de la industria 4.0.

4.2 Recomendaciones

- Las empresas dedicadas a la creación de nueva tecnología en el ámbito de micro-ordenadores como es la fundación Raspberry PI, no descansan en actualizar sus productos y lanzar nuevos modelos con mayores beneficios que los anteriores, en junio del 2019 la fundación lanza su nuevo modelo la Raspberry PI4, superando en varias características al Modelo PI3 B+, dispositivo que se utilizó en este proyecto, dejando como sugerencia la utilización del mismo, para obtención de sistemas más estables y brindando mejores beneficios en las aplicaciones o proyectos que deseen realizar.

- Para un mejor desempeño de la Raspberry PI, se recomienda utilizar disipadores de energía o ventiladores en el mejor de los casos, para cuidar el sobrecalentamiento del procesador de la placa, teniendo en cuenta que el sistema propuesto cuenta con varios procesos ejecutándose, si en caso de añadirse más procesos podría colapsar y dañar el dispositivo, igualmente se debe tener en cuenta la protección física de la Raspberry, optando por una armadura según

dependa el caso del campo de trabajo de la tarjeta, puesto que un daño en el hardware puede destruir la placa y terminar con el proceso que se está ejecutando.

- En el diseño de la página web se recomienda instalar en Raspbian el navegador "Mozilla Firefox" para la ejecución de la página y así como en sus pruebas, puesto que otros navegadores requieren más configuraciones y otros no ejecutan el código predispuesto, además de que todos los archivos html, css, javascripts y sh, estén bajo una misma carpeta para no tener complicaciones a la hora de dar permisos en la ejecución del sistema, como también darle una dirección fija al dispositivo, y trabajar con ella todo el tiempo según se requiera el caso.

Bibliografía

[1] M. C. Lucas-Estañ, T. P. Raptis, M. Sepulcre, A. Passarella, C. Regueiro, and O. Lazaro, "A software defined hierarchical communication and data management architecture for industry 4.0," in *2018 14th Annual Conference on Wireless On-Demand Network Systems and Services, WONS 2018 - Proceedings*, 2018, vol. 2018-Janua, pp. 37–44.

[2] C. Liu *et al.*, "Reconfigurable Smart Factory for Drug Packing in Healthcare Industry 4.0," *IEEE Trans. Ind. Informatics*, vol. 15, no. 1, pp. 507–516, 2018.

[3] T. Akiba, H. Matsukawa, H. Narimatsu, S. I. Eitoku, and K. Kitamura, "Device functions virtualization architecture," in *2017 IEEE 6th Global Conference on Consumer Electronics, GCCE 2017*, 2017, vol. 2017-January, pp. 1–2.

[4] Y. W. Ma, Y. C. Chen, and J. L. Chen, "SDN-enabled network virtualization for industry 4.0 based on IoTs and cloud computing," in *International Conference on Advanced Communication Technology, ICACT*, 2017, pp. 199–202.

[5] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *Proceedings - IEEE INFOCOM*, 2016, vol. 2016-Septe, pp. 999–1000.

[6] Y. Xu, V. Mahendran, and S. Radhakrishnan, "Towards SDN-based fog computing: MQTT broker virtualization for effective and reliable delivery," in *2016 8th International Conference on Communication Systems and Networks, COMSNETS 2016*, 2016.

[7] J. L. del Val Román, "La Digitalización y la Industria 4.0. Impacto industrial y laboral," in *XXVII Congreso de ACEDE*, 2017, pp. 1–93.

[8] "(PDF) Implementación de un ambiente de virtualización para el manejo de múltiples servidores de voip sobre una plataforma común de hardware." [Online]. Available: https://www.researchgate.net/publication/277878684_Implementacion_de_un_ambiente_de_virtualizacion_para_el_manejo_de_multiples_servidores_de_voip_sobre_una_plataforma_comun_de_hardware. [Accessed: 16-Dec-2019].

[9] J. Carro Suárez, F. Flores Salazar, I. Flores Nava, and R. Hernández Hernández, "Industry 4.0 and Digital Manufacturing: a design method

applying Reverse Engineering Industria 4.0 y Manufactura Digital: un Método de Diseño Aplicando Ingeniería Inversa," *Ingeniería*, vol. 24, no. 1, pp. 6–28, 2019.

[10] "Cómo aprovechar la industria 4.0 para mejorar la eficiencia de los equipos industriales." [Online]. Available: https://smart-lighting.es/industria-4-0-mejorar-eficiencia-equipos-industriales/. [Accessed: 17-Dec-2019].

[11] B. Mercedes, G. Cristian, G. Matias, S. Christopher, and T. Carlos, "Virtualización en la Educación: Laboratorio Portátil de Redes," Argentina, 2017.

[12] P. A. Pessolani, "Un Modelo de Arquitectura para un Sistema de Virtualización Distribuido," Universidad Nacional de la Plata, 2018.

[13] "Alternativas a los contenedores en Docker - 1&1 IONOS." [Online]. Available: https://www.ionos.es/digitalguide/servidores/know-how/alternativas-a-los-contenedores-en-docker/. [Accessed: 23-Sep-2019].

[14] Maria Cabrera Gomez de la Torre, "GESTIÓN DE CONTENEDORES DOCKER-KUBERNETES," 2016.

[15] Daniel Ortego Delgado, "¿Qué son los contenedores? Kubernetes, Mesos, Docker... | OpenWebinars," 2016. [Online]. Available: https://openwebinars.net/blog/kubernetes-mesos-docker-que-son-los-contenedores/. [Accessed: 23-Sep-2019].

[16] D. Britch, "Microservicios en contenedores - Xamarin | Microsoft Docs," 2017. [Online]. Available: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/containerized-microservices. [Accessed: 23-Sep-2019].

[17] C. Catalán Cantero, "Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil," Universidad de Zaragoza, 2015.

[18] 4DIAC Open Source Initiative, "4DIAC Documentation." [Online]. Available: https://www.eclipse.org/4diac/en_help.php?helppage=html/installation/install.html#4DIAC-IDE. [Accessed: 24-Sep-2019].

[19] S. P. B. Pulluquitin, "Desarrollo De Un Sistema De Control Industrial Basado En El Estándar Iec-61499," *Repo.Uta.Edu.Ec*, vol. 593, no. 03, pp. 1–73, 2017.

[20] G. J. Caiza and M. V. García, “Implementación de sistemas distribuidos de bajo costo bajo norma IEC-61499, en la estación de clasificación y manipulación del MPS 500,” Salesian Polytechnic University of Ecuador, 2017.

[21] “CMake.” [Online]. Available: https://cmake.org/. [Accessed: 04-Oct-2019].

[22] P. A. Semle, “Protocolos IIoT,” *AADECA Rev.*, pp. 32–35, 2016.

[23] G. Verdejo Alvarez, “SEGURIDAD EN REDES IP: Los protocolos TCP/IP.”

[24] P. U. Javeriana and L. F. Valencia, “Sistema de realidad virtual para el entrenamiento de médicos en la inserción de herramientas quirúrgicas en la piel,” 2018.

[25] “Powerful 2d - 3d software for modeling, animation, rendering, & simulation | App builder - Unity.” [Online]. Available: https://unity3d.com/es/unity?_ga=2.100198749.216323479.1569333681-1857932096.1569333681. [Accessed: 24-Sep-2019].

[26] “Comparativa de los mejores programas para Realidad Virtual en Arquitectura - 3D Business School.” [Online]. Available: https://3dbusinessschool.com/comparativa-de-los-mejores-programas-para-realidad-virtual-en-arquitectura/. [Accessed: 17-Dec-2019].

[27] Carlos Martinez Gamarra, “Microcomputadoras & Mini PC’s.” [Online]. Available: https://prezi.com/pfeto9hwtx2d/microcomputadoras-mini-pcs/. [Accessed: 04-Oct-2019].

[28] J. D. de Usera, “Alternativas a Raspberry Pi 3 B+: las mejores opciones para este 2019,” 2019. [Online]. Available: https://hardzone.es/2019/02/02/alternativas-raspberry-pi-3-b-opciones-2019/. [Accessed: 04-Oct-2019].

[29] Díaz María Fernanda and González José Luis, “Raspberry PI 3 y pcDUINO,” p. 52, 2018.

[30] O. Kuka Youbot and E. Santiago Barahona Guamani, “LÍNEA DE INVESTIGACIÓN: Sistemas de Control y Automatización.”

[31] J. Serrano Navarro, “DESARROLLO DE UN SISTEMA DE COMUNICACIONES USANDO COMPUTADORES DE PLACA REDUCIDA,” 2016.

[32] Ministerio del Trabajo, “Tabla de salarios mínimos sectoriales 2019.xlsx -

Google Drive.” [Online]. Available: https://drive.google.com/file/d/1mRqEvHx6U5yJZgvwa-8X8ErRrFqCmUKu/view. [Accessed: 04-Jan-2020].

ANEXOS

ANEXO 1: Instalación de Docker

Para la instalación de Docker se tuvo que requerir a buscar información sobre la nueva versión del sistema Operativo Raspbian Stretch, y dejando atrás la antigua versión Raspbian Jessie para esto se necesitó estar conectado a internet. Primero se necesitó instalar los paquetes necesarios para la certificación de docker, con la siguiente línea de comandos

```
sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common
```

A continuación se añade la clave de los paquetes instalados.

```
sudo curl-fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg | sudo apt-key add –
```

Una vez instalado todos los certificados necesarios para la instalación de docker accedemos a los repositorios de la página oficial de docker y descargamos el software para porcesadores Armhf (Rama estable), que tiene nuestro sistema Operativo Raspbian Stretch, con las siguientes líneas de código

```
sudo echo "deb [arch=armhf] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \$(lsb_release -cs) stable" | \ sudo tee /etc/apt/sources.list.d/docker.list
```

Hecho esto se procede actualizar los paquetes de los certificados de docker y del sistema operativo mediante los siguientes comandos.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Mediante la actualización completada se procedió a la instalación de Docker.

```
sudo apt install docker-ce
```

Para el acceso a docker sin privilegios, se procedió a configurar un grupo y añadiéndole un usuario mediante los siguientes comandos, esto ayudara a que se acceda a docker sin tener que dar permisos de superusuario.

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

Culminado la instalación de docker se procede a reiniciar el sistema, y verificar la versión que fue instalada, mediante el siguiente código.

docker -v

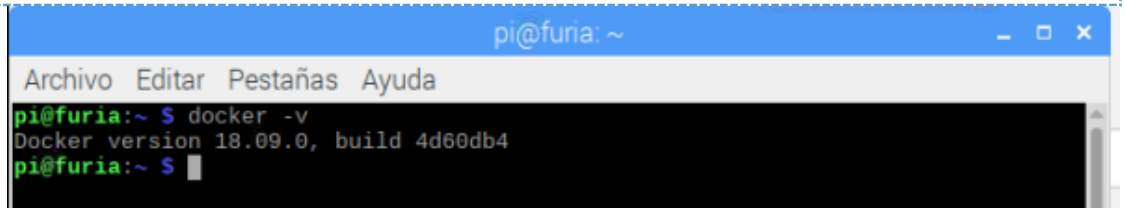


Ilustración 1 Verificación de la versión de Docker Instalada

ANEXO 2: Construcción y Ejecución de contenedores

Para la creación de un contenedor debemos encontrar y buscar si se encuentra la imagen requerida en el registro de docker, para eso se ejecuta el comando.

```
sudo docker search ubuntu
```

Ejecutado el comando saldrá una lista de imágenes disponibles y compatibles con es sistema operativo Raspbian Stretch, en el presente proyecto se seleccionó la imagen Ubuntu:14.04 y se procedió a descargar mediante la línea de condigo siguiente.

```
docker pull ubuntu:14.04
```

Descargada la imagen, se crea el contenedor añadiéndole un nombre y asignándole a la red que crearemos “Bridge”, mediante los siguiente comando.

```
docker run --name proceso1 -it --network bridge "client1.noureldin.local client1":172.88.1.110 ubuntu:14.04 /bin/bash -c; ip route add default via 172.88.1.1 dev eth0; /bin/bash"
```

Una vez creado el contenedor, se realizó el mismo procedimiento a partir de la imagen descargada, en este proyecto se asignó una ip y un nombre a cada contenedor, y se configuró dependiendo cada caso, los nombres para los contenedores son: proceso1, proceso2, proceso1re, proceso2res, comunicación y comunicacionres, si se creó correctamente los contenedores se puede observar mediante la siguiente línea de código, como en la ilustración 2.

```
docker ps -a
```

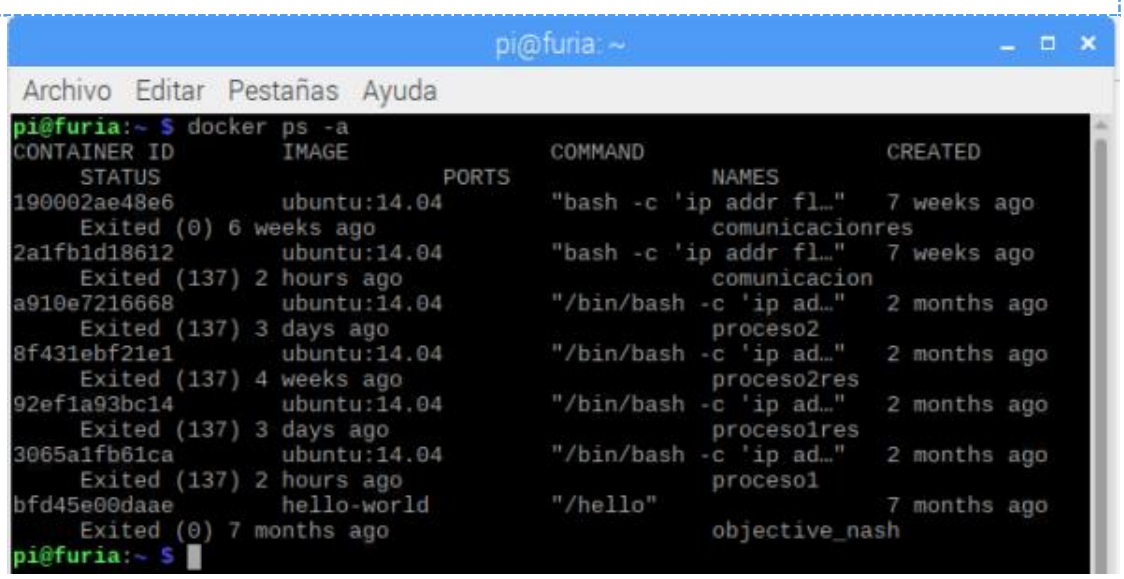


Ilustración 2 Lista de los contenedores Creados

Cuando se corre por primera vez el contenedor, se accede automáticamente dentro del mismo, se tiene en cuenta que cada contenedor se corre en modo superusuario, donde se puede comprobar la ip del contenedor, y el nombre del cliente a cual fue asignado en este ejemplo, es del contenedor proceso1 con ip 172.88.1.110, reconocido como cliente uno, esto se puede ver mediante el siguiente comando, y se puede apreciar en la ilustración 3.

```
ip add
```

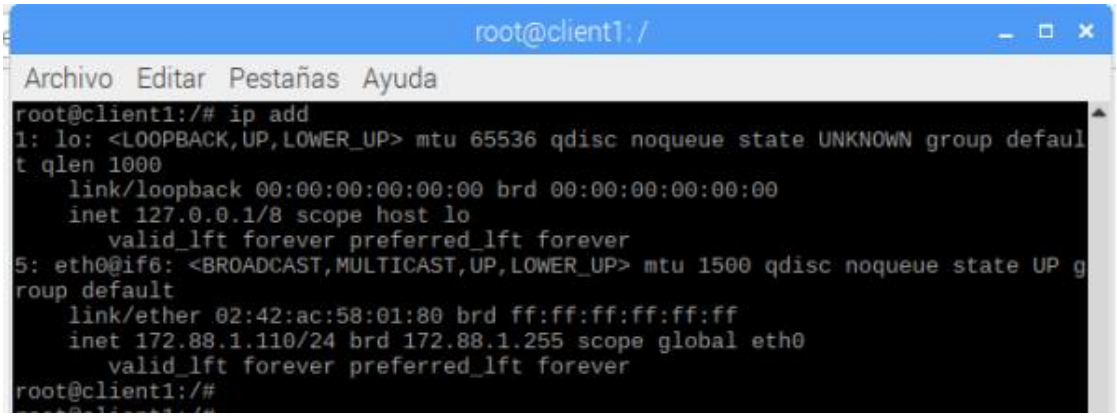


Ilustración 3 Dirección asignada al contenedor proceso1

Hecho esto se actualiza el sistema del contenedor con los siguientes comandos

```
apt-get update
```

```
apt-get upgrade
```

A continuación se procede a instalar el sistema de control de versiones distribuido "git" para poder acceder a los repositorios de GitHub y Bitbucket, mediante el siguiente comando.

```
apt-get install git
```

hecho esto se pudo acceder al repositorio GitHub, en donde se encuentran las herramientas de construcción de cmake, make, gcc y g++, que necesitaremos para poder ejecutar el archivo forte, para esto ejecutamos la siguiente línea de comandos.

```
apt-get install git cmake make gcc g++
```

Completado la instalación y actualizado nuevamente el sistema por los cambios hechos, se procede a apagar el contenedor mediante el siguiente comando

```
exit
```

Para el arranque de los contenedores y apagado de los mismos se utiliza los siguientes comandos y el proceso que se vaya a ejecutar ya sea proceso1, proceso2, proceso1re, proceso2res, comunicación y comunicacionres.

```
docker start ……....
docker stop …….
```

Para ejecutar la aplicación forte se ejecuta mediante el comando, direccionando en donde se ejecuta el archivo, aquí se ejecuta el archivo forte del proceso1

```
docker exec -i proceso1 sh -c "cd /home/mqttcomu/src/ && ./forte"
```

Para matar el proceso se ejecuta el siguiente comando

```
docker exec -i proceso1 sh -c "v=$(pgrep forte) && kill "$v" && exit"
```

ANEXO 3: Configuración del archivo proceso.cpp

A continuación se presenta la programación de configuración del proceso 1 en lenguaje c++, incluyendo sus librerías.

```
***  Librerias de FORTE
#include "proceso1.h"
#ifdef FORTE_ENABLE_GENERATED_SOURCE_CPP
#include "proceso_gen.cpp"
#endif
*** Este archive es generado por 4DIAC-FORTE
*** NOMBRE: proceso1
 *** Descripción: Programación de un Bloque funcional Básico
DEFINE_FIRMWARE_FB(FORTE_proceso, g_nStringIdproceso)
const CStringDictionary::TStringId FORTE_proceso::scm_anDataInputNames[] = {g_nStringIddato, g_nStringIdstop};
const CStringDictionary::TStringId FORTE_proceso::scm_anDataInputTypeIds[] = {g_nStringIdINT, g_nStringIdBOOL};
const CStringDictionary::TStringId FORTE_proceso::scm_anDataOutputNames[] = {g_nStringIdQO, g_nStringIdsalida};
const CStringDictionary::TStringId FORTE_proceso::scm_anDataOutputTypeIds[] = {g_nStringIdBOOL, g_nStringIdUINT};
const TForteInt16 FORTE_proceso::scm_anEIWithIndexes[] = {-1, 0};
const TDataIOID FORTE_proceso::scm_anEIWith[] = {0, 1, 255};
const CStringDictionary::TStringId FORTE_proceso::scm_anEventInputNames[] = {g_nStringIdINIT, g_nStringIdREQ};
const TDataIOID FORTE_proceso::scm_anEOWith[] = {0, 1, 255, 0, 1, 255};
const TForteInt16 FORTE_proceso::scm_anEOWithIndexes[] = {0, 3, -1};
const CStringDictionary::TStringId FORTE_proceso::scm_anEventOutputNames[] = {g_nStringIdINITO, g_nStringIdCNF};
const SFBInterfaceSpec FORTE_proceso::scm_stFBInterfaceSpec = {
  2, scm_anEventInputNames, scm_anEIWith, scm_anEIWithIndexes,
  2, scm_anEventOutputNames, scm_anEOWith, scm_anEOWithIndexes, 2, scm_anDataInputNames, scm_anDataInputTypeIds,
```

```
  2,  scm_anDataOutputNames, scm_anDataOutputTypeIds, 0, 0};

void FORTE_proceso::alg_RESET(void){
salida() = 0;}
void FORTE_proceso::alg_REQ(void){
if((dato() == 1)){
salida() = 1;};
if((dato() == 2)){
salida() = 7;};
if((dato() == 3)){
salida() = 9;};
if((dato() == 4)){
salida() = 12;};
if((dato() == 5)){
salida() = 10;};
if((dato() == 6)){
salida() = 8};
if((dato() == 7)){
salida() = 5;};
if((dato() == 8)){
salida() = 7;};
if((dato() == 9)){
salida() = 9;};
if((dato() == 10)){
salida() = 11;};
if((dato() == 11)){
salida() = 10};
if((dato() == 12)){
salida() = 8;};
if((stop() == false)){
salida() = 0;};
}
```

```
void FORTE_proceso::enterStateSTART(void){
m_nECCState = scm_nStateSTART;
}
void FORTE_proceso::enterStateINIT(void){
m_nECCState = scm_nStateINIT;
alg_RESET();
sendOutputEvent( scm_nEventINITOID);
}

void FORTE_proceso::enterStateREQ(void){
 m_nECCState = scm_nStateREQ;
 alg_REQ();
 sendOutputEvent( scm_nEventCNFID);
}
void FORTE_proceso::executeEvent(int pa_nEIID){
bool bTransitionCleared;
do{
 bTransitionCleared = true;
switch(m_nECCState){
case scm_nStateSTART:
if(scm_nEventINITID == pa_nEIID)
enterStateINIT();
else
 if(scm_nEventREQID == pa_nEIID)
enterStateREQ();
else
bTransitionCleared  = false; //no transition cleared
break;
case scm_nStateINIT
 if((1))
enterStateSTART();
else
bTransitionCleared  = false; //no transition cleared
```

```
 break;
 case scm_nStateREQ:
if((1))
enterStateSTART();
else
bTransitionCleared  = false; //no transition cleared
break;
default:
DEVLOG_ERROR("The state is not in the valid range! The state value is: %d. The max value can be: 2.", m_nECCState.operator TForteUInt16 ());
m_nECCState = 0; //0 is always the initial state
break;
}
pa_nEIID = cg_nInvalidEventID;  // we have to clear the event after the first check in order to ensure correct behavior
}while(bTransitionCleared);
}
```

ANEXO 4: Configuración del archivo proceso.h

A continuación se presenta la programación de configuración de las entradas y salida de bloque funcional creado en lenguaje c++.

```
***librerias
#ifndef _PROCESO_H_
#define _PROCESO_H_
#include <basicfb.h>
#include <forte_bool.h>
#include <forte_uint.h>
#include <forte_int.h>
class FORTE_proceso: public CBasicFB{
DECLARE_FIRMWARE_FB(FORTE_proceso)
private:
 static const CStringDictionary::TStringId scm_anDataInputNames[];
static const CStringDictionary::TStringId scm_anDataInputTypeIds[];
CIEC_INT &dato() {
return *static_cast<CIEC_INT*>(getDI(0));
  };
  CIEC_BOOL &stop() {
  return *static_cast<CIEC_BOOL*>(getDI(1));
  };


  static const CStringDictionary::TStringId scm_anDataOutputNames[];
  static const CStringDictionary::TStringId scm_anDataOutputTypeIds[];
  CIEC_BOOL &QO() {
    return *static_cast<CIEC_BOOL*>(getDO(0));
  };
  CIEC_UINT &salida() {
    return *static_cast<CIEC_UINT*>(getDO(1));
  };
  static const TEventID scm_nEventINITID = 0;
  static const TEventID scm_nEventREQID = 1;
```

```
  static const TForteInt16 scm_anEIWithIndexes[];
  static const TDataIOID scm_anEIWith[];
  static const CStringDictionary::TStringId scm_anEventInputNames[];
  static const TEventID scm_nEventINITOID = 0;
  static const TEventID scm_nEventCNFID = 1;
  static const TForteInt16 scm_anEOWithIndexes[];
  static const TDataIOID scm_anEOWith[];
  static const CStringDictionary::TStringId scm_anEventOutputNames[];
  static const SFBInterfaceSpec scm_stFBInterfaceSpec;
   FORTE_BASIC_FB_DATA_ARRAY(2, 2, 2, 0, 0);
  void alg_RESET(void);
  void alg_REQ(void);
  static const TForteInt16 scm_nStateSTART = 0;
  static const TForteInt16 scm_nStateINIT = 1;
  static const TForteInt16 scm_nStateREQ = 2;
  void enterStateSTART(void);
  void enterStateINIT(void);
  void enterStateREQ(void);
  virtual void executeEvent(int pa_nEIID);
public:
  FORTE_proceso(CStringDictionary::TStringId pa_nInstanceNameId, CResource
*pa_poSrcRes) :
      CBasicFB(pa_poSrcRes, &scm_stFBInterfaceSpec, pa_nInstanceNameId,
    0, m_anFBConnData, m_anFBVarsData){
  };
  virtual ~FORTE_proceso(){}
};
#endif
```

ANEXO 5: Programa compilado en la tarjeta Arduino

```
//librerias
#include <SPI.h>
#include <Ethernet.h>
#include <Servo.h>

// se añade una mac para arduino y la ip del servidor
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(172, 88, 1, 118);

//se inicializa el puerto por el cual vamos se va comunicar con los demás dispositivos
EthernetServer server(61505);

// se inicializa la identificación de los servomotores y su posición
Servo base;
Servo dere;
Servo izqui;
Servo pinza;
base.write(0);
dere.write(0);
izqui.write(180);
pinza.write(150);

void setup() {
 // Iniciamos los servomotores para que empiece a trabajar con los pines del Arduino
base.attach(2);
dere.attach(4);
izqui.attach(6);
pinza.attach(8);
Serial.begin(9600);
while (!Serial) {
```

```
;
}
Serial.println("Controlando Brazo Robótico");

// Se verifica la ip y la mac del Dispositivo
Ethernet.begin(mac, ip);

// arranca el servidor
server.begin();
Serial.print("servidor ");
Serial.println(Ethernet.localIP());
}
void loop() {
// escucha si hay clientes
EthernetClient client = server.available();
if (client) {
Serial.println("nuevo Cliente");
boolean currentLineIsBlank = true;
while (client.connected()) {
if (client.available()) {
int dato = client.read(); // lee y guarda el dato en la variable dato
Serial.write(dato);
switch (dato) {
case 1:
//varia la posición servomotor base
for (bas = 0; bas <= 90; bas += 1)
{
base.write(bas);
delay(15);
}
break;
case 2:
for (bas = 0; bas <= 180; bas += 1)
```

```
{
base.write(bas);
delay(15);
}
break;
case 3:
for (bas = 90; bas <= 180; bas += 1)
{
base.write(bas);
delay(15);
}
break;
case 4:
for (bas = 180; bas >= 0; bas -= 1)
{
base.write(bas);
delay(15);
}
break;
case 5:
for (bas = 90; bas >= 0; bas -= 1)
{
base.write(bas);
delay(15);
}
break;
case 6:
for (bas = 180; bas >= 90; bas -= 1)
{
base.write(bas);
delay(15);
}
break;
```

```
//varia la posición servomotor Brazo Base
case 7:
for (der = 0; der <= 60; der += 1)
{
dere.write(der);
delay(15);
}
break;
case 8:
for (der = 60; der >= 0; der -= 1)
{
base.write(der);
delay(15);
}
break;

//varia la posición servomotor Antebrazo
case 9:
for (izq = 180; izq >= 120; izq -= 1)
{
izqui.write(izq);
delay(15);
}
break;
case 10:
for (izq = 120; izq <= 180;izq += 1)
{
izqui.write(izq);
delay(15);
}
break;
```

```
//varia la posición servomotor Pinza
case 11:
for (pin = 150; pin >= 120; pin -= 1)
{
pinza.write(pin);
delay(15);
}
break;
case 12:
for (pin = 120; pin <= 150; pin += 1)
{
pinza.write(pin);
delay(15);
}
break;
//Si no encuentra datos mantiene la posicion inicial
default:
base.write(0);
dere.write(0);
izqui.write(180);
pinza.write(150);
break;
}
}
}
//retraso de 1 milisegundo hasta receptar otro dato
delay(1);
// Cierra la conexión:
client.stop();
Serial.println("Cliente desconectado");
}
}
```

ANEXO 7: Configuración del Protocolo MQTT en el software Unity 3D

```
//Librerias
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Net;
using System;
using uPLibrary.Networking.M2Mqtt;
using uPLibrary.Networking.M2Mqtt.Messages;
using uPLibrary.Networking.M2Mqtt.Utility;
using uPLibrary.Networking.M2Mqtt.Exceptions;
public class ROBOT : MonoBehaviour
{
// Declaramos variables privadas para el cliente MQTT
private MqttClient client;
private string dat;
void Start () {
// Creamos la instancia del cliente
client = new MqttClient(IPAddress.Parse("172.88.1.104"),1883,false);
// se registra el mensaje que se resibe
client.MqttMsgPublishReceived += client_MqttMsgPublishReceived;
string clientId = Guid.NewGuid().ToString();
client.Connect(clientId);
// Se subscribe al topico hello y se obtiene los datos
client.Subscribe(new string[] { "hello" }, new byte[] {
MqttMsgBase.QOS_LEVEL_EXACTLY_ONCE });
}
void client_MqttMsgPublishReceived(object sender, MqttMsgPublishEventArgs e)
{
Debug.Log("Received: " + System.Text.Encoding.UTF8.GetString(e.Message)  );
dat= System.Text.Encoding.UTF8.GetString(e.Message);
dato = Convert.ToInt32(dat);//se convierte en dato a variable int
}
```

```
// Se declara variables públicas para el movimiento del brazo Robótico
public float dato;
public float x;
public float y;
public float z;
public float timeCount = 0.0f;
public float timeCount1 = 0.0f;
public float timeCount2 = 0.0f;
public float timeCount3 = 0.0f;
public float timeCount4 = 0.0f;
public Transform RobotBase;
public Transform Brazo;
public Transform Antebrazo;
public Transform pdere;
public Transform pizq;
void Update(){
// Según el dato obtenido realizara los debidos movimientos en grados de cada parte del robot como corresponda
 switch (dato)
{
case 1:
RobotBase.rotation = Quaternion.Slerp (Quaternion.Euler(0,0,0),Quaternion.Euler(0,-90,0), timeCount);
timeCount = timeCount + Time.deltaTime;
timeCount1=0;
y=-90;
break;
case 2:
RobotBase.rotation = Quaternion.Slerp (Quaternion.Euler(0,0,0),Quaternion.Euler(0,180,0), timeCount);
timeCount = timeCount + Time.deltaTime;
timeCount1=0;
y=180;
```

```
break;
case 5:
RobotBase.rotation = Quaternion.Slerp (Quaternion.Euler(0,-90,0),Quaternion.Euler(0,0,0), timeCount);
timeCount = timeCount + Time.deltaTime;
timeCount1=0;
y=0;
break;
case 6:
RobotBase.rotation = Quaternion.Slerp (Quaternion.Euler(0,180,0),Quaternion.Euler(0,0,0), timeCount);
timeCount = timeCount + Time.deltaTime;
timeCount1=0;
y=0;
break;
case 7:
Brazo.rotation = Quaternion.Slerp (Quaternion.Euler(330,y,0),Quaternion.Euler(0,y,0), timeCount1);
timeCount1 = timeCount1 + Time.deltaTime;
timeCount=0;
break;
case 8:
Brazo.rotation = Quaternion.Slerp (Quaternion.Euler(0,y,0),Quaternion.Euler(330,y,0), timeCount1);
timeCount1 = timeCount1 + Time.deltaTime;
timeCount2=0;
break;
case 9:
Antebrazo.rotation = Quaternion.Slerp (Quaternion.Euler(20,y,0),Quaternion.Euler(40,y,0), timeCount2);
timeCount2 = timeCount2 + Time.deltaTime;
timeCount1=0;
break;
```

```
case 10:
Antebrazo.rotation = Quaternion.Slerp (Quaternion.Euler(40,y,0),Quaternion.Euler(20,y,0), timeCount2);
timeCount2 = timeCount2 + Time.deltaTime;
timeCount3=0;
break;
case 11:
pdere.rotation = Quaternion.Slerp (Quaternion.Euler(0,110,90),Quaternion.Euler(0,90,90), timeCount3);
pizq.rotation = Quaternion.Slerp (Quaternion.Euler(0,230,270),Quaternion.Euler(0,250,270), timeCount3);
timeCount3 = timeCount3 + Time.deltaTime;
timeCount=0;
break;
case 12:
pdere.rotation = Quaternion.Slerp (Quaternion.Euler(0,y+90,90),Quaternion.Euler(0,y+110,90), timeCount3);
pizq.rotation = Quaternion.Slerp (Quaternion.Euler(0,y+250,270),Quaternion.Euler(0,y+230,270), timeCount3);
timeCount3 = timeCount3 + Time.deltaTime;
timeCount2=0;
break;
default:
Debug.Log("no se obtiene datos");
break;
}
}
}
```