UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Carrera de Ingeniería en Sistemas Computacionales

Estudio de la plataforma Kubernetes para la administración automática de contenedores Docker en la facultad FICA de la Universidad Técnica del Norte

Correa Cuvi Wesly Jhalmarth

MSc. Rea Peñafiel Xavier Mauricio

2022

Dedicatoria

Para mi madre, mi bella dama y mi familia.

Resumen

El desarrollo web, al ser uno de los puntos de mayor importancia en el área informática, proporciona soluciones, las cuales son diversas, y dependiendo la necesidad otorga beneficios a los usuarios. En virtud de ello, entender la manera estable de como enviar aquellos desarrollos a producción genera un enfoque práctico de investigación. Por consiguiente, Kubernetes, la cual, al ser una herramienta de administración de servicios, enmarca con gran ímpetu la relevancia de conocer su funcionamiento, al mantener una relación estable con los contenedores, los cuales Kubernetes gestiona a la perfección, otorgando la capacidad de exponer servicios de manera declarativa mediante ficheros YAML.

Abstract

Web development, being one of the most important points in the computing area, provides diverse solutions, and depending on their needs, it also provides benefits to users. By virtue of this, understanding the stable way of sending those developments to production generates a practical research approach. Therefore, Kubernetes, which are a service management tool, deeply promotes the relevance of knowing how it works, by maintaining a stable relationship with containers, which Kubernetes manages perfectly, granting the ability to expose services declaratively using YAML files.

Agradecimiento

A todos los que me apoyaron, gracias.

Autorización de uso y publicación

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN
A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	**1501232696**		
APELLIDOS Y NOMBRES:	**Correa Cuvi Wesly Jhalmarth**		
DIRECCIÓN:	**Ibarra, El Olivo**		
EMAIL:	**wjcorreac@utn.edu.ec**		
TELÉFONO FIJO:	**2320-089**	**TELÉFONO MÓVIL:**	**0988082199**

DATOS DE LA OBRA	
TÍTULO:	**Estudio de la plataforma Kubernetes para la administración automática de contenedores Docker en la facultad FICA de la Universidad Técnica del Norte**
AUTOR (ES):	**Correa Cuvi Wesly Jhalmarth**
FECHA: DD/MM/AAAA	**30 de marzo de 2022**
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	**Ingeniería en Sistemas Computacionales**
ASESOR /DIRECTOR:	**Ing. Xavier Mauricio Rea Peñafiel, MSc**

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 30 días del mes de marzo de 2022

EL AUTOR:

(Firma)..... **Nombre:** Correa Cuvi Wesly Jhalmarth

Certificación del Director

CERTIFICADO DEL DIRECTOR DE TRABAJO DE GRADO

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN DEL DIRECTOR

En mi calidad de director del trabajo de grado presentado por el egresado Correa Cuvi Wesly Jhalmarth, portador de la cedula de ciudadanía 150123269-6 previo a obtener el título de ingeniería en sistemas computacionales cuyo tema es: Estudio de la plataforma Kubernetes para la administración automática de contenedores Docker en la facultad FICA de la Universidad Técnica del Norte, certifico que el presente trabajo cumple con los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del tribunal examinador que se designe.

Atentamente:

Firmado electrónicamente por:
XAVIER MAURICIO REA PEÑAFIEL

MSc. Mauricio Rea P.
DIRECTOR DE TRABAJO DE GRADO

Índice general

Índice de figuras

Índice de cuadros

CAPÍTULO 1

Introducción

1.1. Antecedente

Los modelos de desarrollo en el sector informático de la Carrera de Ingeniería en Sistemas Computacionales e Ingeniería de Software, según (Vizcaino Quiroz, 2021), mantienen un enfoque en torno a la arquitectura Java EE, llevando a constituir su base de conocimiento sobre la programación web, afianzando de esta manera el entendimiento de una estructura robusta de código y sobre todo mantenible. Pese a tal motivo el despliegue de aquellas soluciones elaboradas con propósito de estudio, se encuentran limitadas debido a la falta de disponibilidad por parte de la infraestructura tecnológica de la Facultad de Ingeniería en Ciencias Aplicadas, a la cual las carreras previamente mencionadas pertenecen.

1.2. Situación actual

Los estudiantes actualmente presentan restricciones al momento de acceder a la plataforma de virtualización debido a la creciente demanda. No obstante, se debe exteriorizar en base a los antecedentes que en la Facultad de Ingeniería en Ciencias Aplicadas (FICA) de la cual forma parte la Carrera de Software (CSOFT), en ella reside una plataforma de virtualización la misma que es usada en varias materias para la elaboración de prácticas por parte de los alumnos. Esto ha llevado a que el servidor de la facultad tenga una saturación en la disponibilidad de este, de tal modo que investigar nuevas formas de lanzar a producción aquellas soluciones genera un relevante nivel de interés.

Una problemática a nivel nacional según (Mejía Rodríguez & Barbecho Chimbo, 2021), es que en Ecuador el uso de las tecnologías en la nube, aún se encuentran emergentes y siguen en discusión en cuanto a emplearlas en entornos empresariales. Sin embargo, 3 países con un alto índice tecnológico, no dudan en operar con ellos.

1.3. Planteamiento del Problema

Cada semestre la carrera de Ingeniería en Sistemas Computacionales actualmente habilitada para titulación y la Carrera de Software, incorporan materias en las cuales el desarrollo de aplicaciones web es su entorno genérico de estudio, por lo tanto, la necesidad de llevar esos desarrollos a un ámbito de producción se ha vuelto casi indispensable. Sin embargo, la gran demanda de acceso a los recursos de virtualización de los servidores de la Facultad genera restricciones en el progreso colectivo de la actividad previamente mencionada.

1.4. Objetivos

1.4.1. General

Estudiar sobre el uso de la plataforma Kubernetes para la administración automática de contenedores Docker en la facultad FICA de la Universidad Técnica del Norte.

1.4.2. Específicos

- Definir una base teórica sobre el uso de contenedores Docker y kubernetes.
- Implementar el orquestador de kubernetes con minikube para realizar las prácticas en entornos locales.
- Validar los resultados, mediante la satisfacción del usuario empleando el cuestionario CSUQ.

1.5. Alcance

El presente trabajo de titulación delimita su área de estudio a la Carrera de Ingeniería de Software (CSOFT) de la Facultad de Ingeniería en Ciencias Aplicadas (FICA), de la Universidad Técnica del Norte (UTN), detallando la funcionalidad e implementación de kubernetes en un entorno de producción mediante el uso de contenedores Docker. A la vez, se pretende alcanzar una solvencia interactiva con la arquitectura Java EE, la cual es la plataforma de desarrollo con mayor demanda en el mercado Regional, sin embargo, los temas y los conceptos a tratar propondrán ideas, las cuales serán adaptables para el lenguaje con el cual se labore de preferencia, de igual forma, se dará uso a minikube para ejecutar una instancia de kubernetes en un nodo. Usando como sistema operativo CentOS 8, no obstante, al tomar partida los contenedores Docker directamente del kernel de Linux, las aplicaciones llegan a ser migrables. Sin embargo, para el proceso de instalación de Docker y kubernetes, es recomendable usar directamente las instrucciones que se encuentran en la página oficial de cada una de las plataformas anteriormente mencionadas, las cuales varían dependiendo el sistema operativo, o la distribución empleada.

Para ello el presente proyecto se subdividirá en 4 fases estratégicas:

1. Desarrollar un conocimiento general de kubernetes.
   a) Investigar el uso de los distintos elementos kubernetes mediante diversos documentos de fuentes bibliográficas verificadas.
2. Construir un entorno de despliegue óptimo con kubernetes.
   a) Creación de servicios para almacenar bases de datos y directamente las aplicaciones con la finalidad de colocarlas en producción.
3. Validación de los resultados.
   a) Los resultados serán verificados, mediante el cuestionario de Computer System Usability Questionnaire (CSUQ), empleando la tercera versión partiendo como base de la ejecución de (Ipiales Gubio, 2021). “Como se cita en” (Barajas-Bustillos y col., 2017).
   b) A la vez se empleará minikube para entornos locales y el proveedor de servicios web a continuación mencionado, con el motivo de manejar kubernetes en servicios externos:
      1) AWS

      Del mismo modo se dará uso a una Raspberry Pi 4 para la gestión de los contenedores creados y predispuestos a producción.
4. Implementar buenas prácticas para el uso óptimo de kubernetes.
   a) Crear archivos con extensión yaml siguiendo normas de optimización, levantamiento de servicios y despliegues para kubernetes.

1.6. Metodología

Para el cumplimiento del primer objetivo, se indagará en la documentación oficial de kubernetes y en investigaciones científicas previamente realizadas sobre el tema a ser tratado.

Para el cumplimiento del segundo objetivo, se instalará kubernetes en un proveedor Cloud externo el cual será AWS para exponer una aplicación de demostración, a la vez que se desarrollará pruebas

con minikube en entornos locales, implementando la solución tecnológica tanto en un servidor Linux personal como en los servidores de la Facultad FICA de la Universidad Técnica del Norte.

Para el cumplimiento del tercer objetivo se hará uso del cuestionario CSUQ, el cual se enfoca en evaluar 4 factores:

- Calidad del sistema
- Calidad de la información
- Calidad de la interfaz
- Satisfacción genera

1.7. Justificación

Para sustentar el enfoque del presente proyecto se hará uso de los Objetivos de Desarrollo Sostenible (ODS), específicamente el Nro. 9, relacionado a "Industria, innovación e infraestructura", planteado por la ONU y la UNESCO (La UNESCO y los Objetivos de Desarrollo Sostenible, 2021). Según la CEPAL esta ODS podría ser incluida en el eje VI del PEAS el cual busca "garantizar la integración productiva" (García y col., 2018):

Las metas del objetivo con las cuales la presente investigación cumple son:

9.5.- Incrementar la investigación y fomentar la innovación, mejorando las capacidades tecnológicas, de cierto modo hasta el 2030 el número de personas que laboren en investigación y desarrollo por millón de habitantes aumente considerablemente (Ibujes Villacís & Franco Crespo, 2019).

> 9.b.- Apoyar al desarrollo tecnológico, la innovación e investigación en países particularmente aquellos que se encuentran en desarrollo, incluyendo garantías de un entorno normativo propicio a la diversificación industrial y la adición de valor a los productos básicos, entre otras cosas (Ibujes Villacís & Franco Crespo, 2019).
>
> 9.c.- Aumentar significativamente el acceso a la tecnología de la información y las comunicaciones y esforzarse por proporcionar acceso universal y asequible a internet en los países menos adelantados, de aquí a 2020 (Ibujes Villacís & Franco Crespo, 2019).

Actualmente al hablar de servicios Cloud, muchas empresas, e instituciones no las emplean, sea por el desconocimiento del funcionamiento de estos servicios, o por miedo al fracaso. Por ende, se entiende que su competencia y alcance de mercado se encuentra limitado a una región en específico, debido a que los costos de mantenimiento de una aplicación web son solventables, y no requieren de herramientas que aporten a mitigar costos de infraestructura tecnológica.

Justificación Tecnológica. - Su importancia radica viendo desde el enfoque empresarial en que, al emplear servicios como lo son los contenedores Docker y kubernetes, se optimizan los recursos y permite a las organizaciones economizar costos de operación, de tal modo que genera beneficios.

Justificación Ambiental. - Al emplear los servicios Cloud se ahorra en costos de infraestructura, los cuales por lo general son elevados, a la vez que su mantenimiento incluye precios adicionales y un flujo de operación que incentiva a la contaminación ambiental.

Justificación Teórica. - El despliegue de aplicaciones y el mantenimiento de las mismas del modo tradicional genera una mayor complejidad que emplear kubernetes con microservicios, por el motivo

de que al operar mediante contenedores los cuales ya se encuentran preconfigurados, el tiempo de realizar una configuración manual es evitado.

1.8. Riesgos

R1.- Caída del servicio de internet: En tal caso al encontrarse kubernetes siendo manejado en un entorno local, el servicio, se encontraría con dificultades operacionales, pero por lo general se lo emplea con proveedores externos, los cuales nunca dejan de operar.

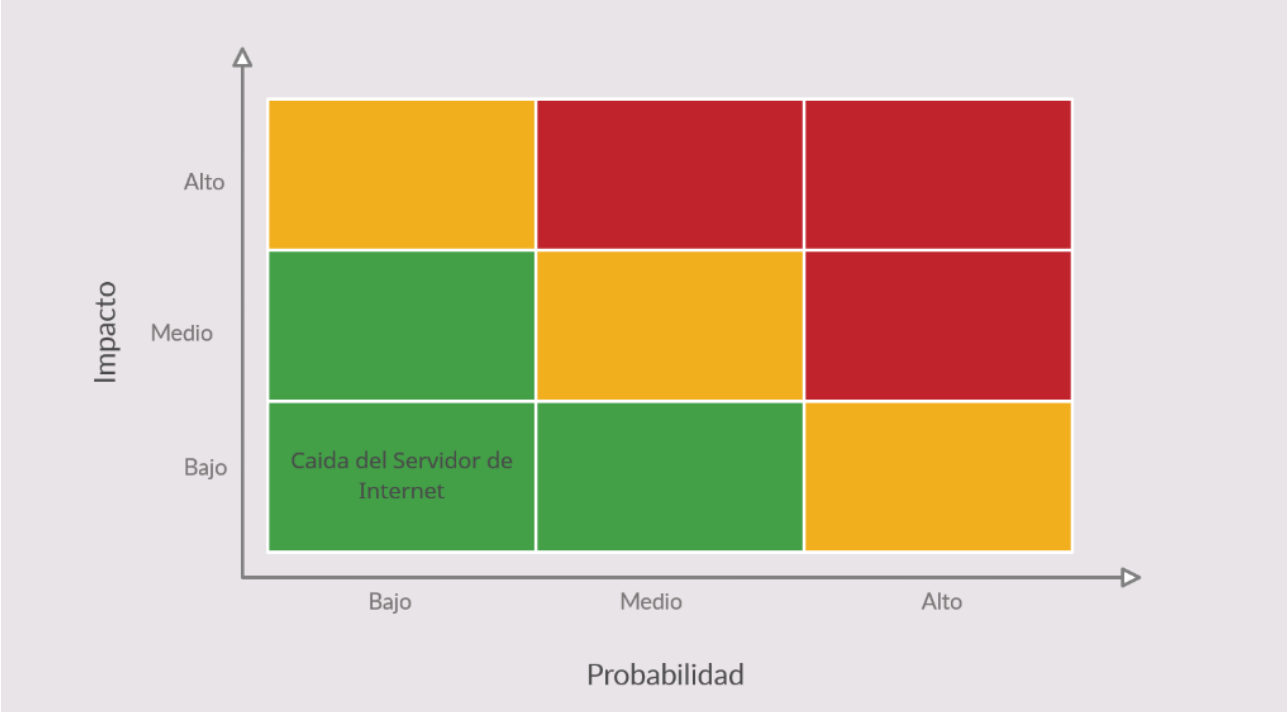


Figura 1.1: Riesgo Nro 1
Fuente: Elaboración Propia

R2.- Desplegar un POD a producción: Los PODs al ser desechables, k8s, en caso de algún problema con la aplicación, inmediatamente desechara ese POD y creara otro, pero si ese POD obtenía acceso a una base de datos, puede que falle, por ende, al desplegar una aplicación se emplean los DEPLOY, los cuales enlazan diversos servicios para colocarlos en producción. En tal caso únicamente se debe cambiar el tipo de servicio y emplear las configuraciones para el DEPLOY y funcionando la aplicación normalmente, cabe destacar que esto en un entorno de producción no se va a generar, a excepción de que se encuentre realizando pruebas del servicio.

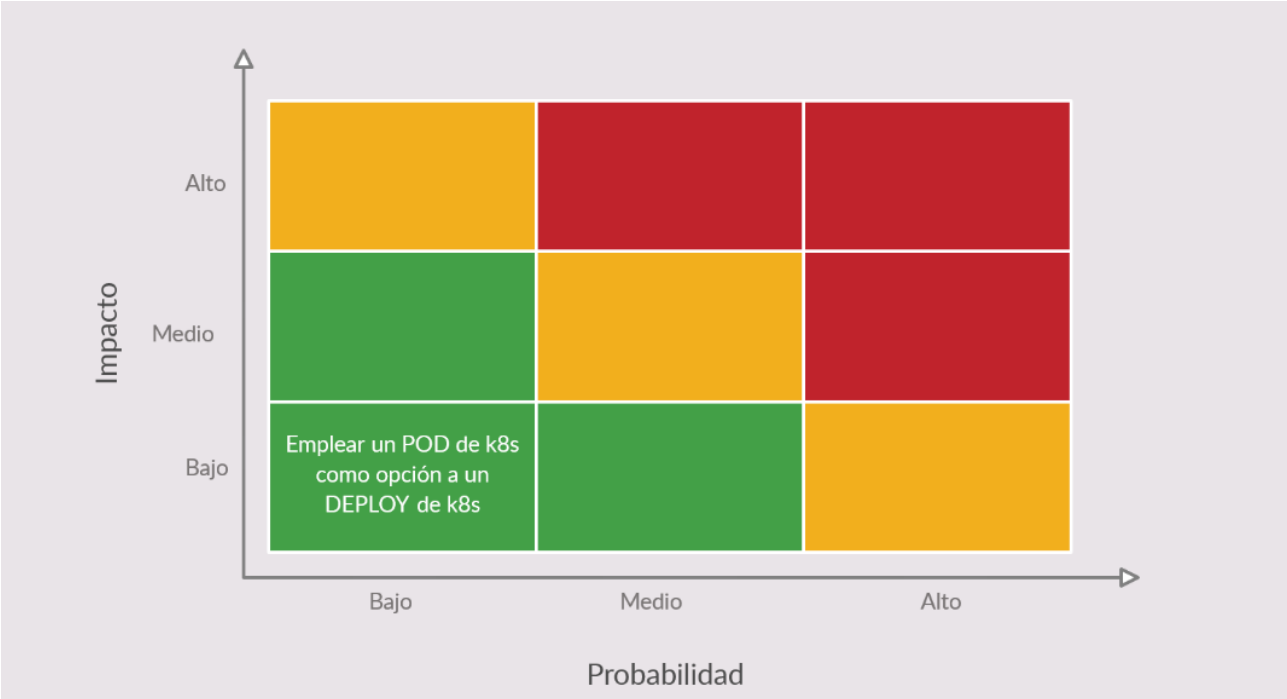


Figura 1.2: Riesgo Nro 2
Fuente: Elaboración Propia

R3.- Que el desarrollador no tengo acceso a internet para realizar el despliegue de una actualización y por ende no pueda actualizar una tarea concisa. Aun así, dependiendo la importancia de esta el impacto podría ser bajo o alto. Aunque la probabilidad de que suceda tal riesgo es baja.

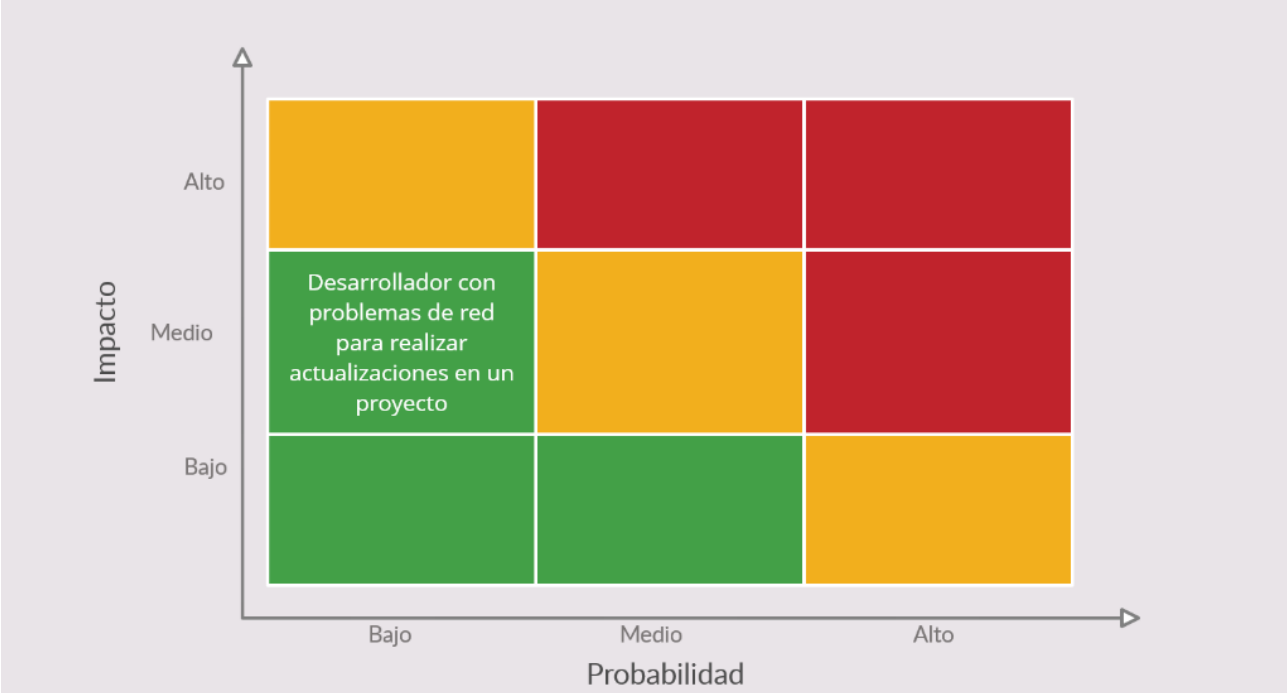


Figura 1.3: Riesgo Nro 3
Fuente: Elaboración Propia

CAPÍTULO 2

Marco Teórico

2.1. Arquitecturas Monolíticas.

Es un tipo de arquitectura de arquitectura de software, la cual mantiene los componentes necesarios para que una aplicación establezca un régimen de funcionalidad adecuado, creando una base centrada de código, como lo supo expresar (Mendonça y col., 2021). De tal modo que se da por entender que tanto las funcionalidades del Frontend como lo sería:

- Html
- Css
- Sass
- JavaScript

Como las del Backend, las cuales corresponden a la lógica de la aplicación. Engloban sus componentes, haciendo del mantenimiento de la aplicación una tarea de suma complejidad. De este modo en caso surja un fallo en la aplicación, los procesos necesarios para su corrección serian una fase extensa de tiempo, en la cual el servicio quedaría fuera de operación, y en caso de no haber mantenido una copia de seguridad del mismo el tiempo para la restauración del sistema seria aun mayor dependiendo sin lugar a duda de la extensión del sistema.

2.2. Virtualización

La idea principal de la virtualización es separar los servicios en máquinas virtuales diferentes, es decir el servidor web, el servidor proxy u otros, en sistemas diferentes, evitando la necesidad de adquirir nuevos equipos ahorrando de preferencia recursos económicos. Permite crear de manera eficiente variados entornos simulados.

Cuando se tiene un sistema operativo, y en el mismo se encuentra instalado un software de virtualización.

2.2.1. Razones para usar máquinas virtuales:

- Ahorro de recursos económicos
- Realizar pruebas de funcionalidad con diversos servicios y sistemas.
- Separar servicios.

2.2.2. Host

Máquina física en donde serán instaladas las máquinas virtuales.

2.2.3. Hipervisor

Es el sistema de virtualización encargado de crear y ejecutar las máquinas virtuales.

Tipo 1 (Bare Metal)

También mencionado como *"metal desnudo"*, otorga soporte físico para almacenar los datos, o se instalan diferentes tipos de servicios, como lo sería un sistema operativo.

Términos	**Conceptos**	**Ejemplos**
Máquinas Virtuales	Es el huésped instalado por medio del hipervisor, se encuentra aislado de la máquina física.	Linux Windows Mac OS otros
Hipervisor	Es el software de virtualización, el cual se emplea para la creación de las máquinas virtuales, en este caso al ser de tipo 1, parten directamente del hardware.	Proxmox Citrix ESXI VMware Hyper-V KVM
Hardware	Elemento físico en el cual se instala el sistema operativo.	Computadora Servidor

Cuadro 2.1: Hipervisor Tipo 1
Fuente: Elaboración Propia

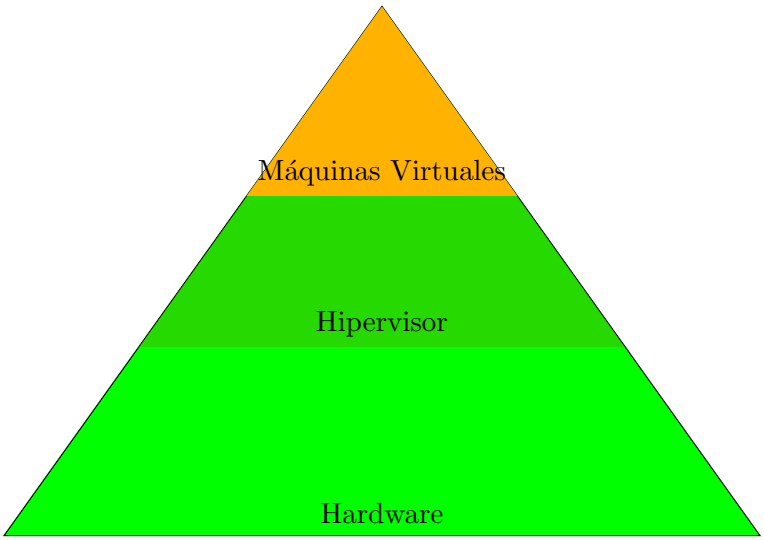


Figura 2.1: Hipervisor Tipo 1
Fuente: Elaboración Propia

Tipo 2 (Hosted)

Mantiene una estructura del siguiente tipo:

Términos	**Conceptos**	**Ejemplos**
Máquinas Virtuales	Es el huésped instalado por medio del hipervisor, se encuentra aislado de la máquina física.	Linux Windows Mac OS otros
Hipervisor	Es el software de virtualización, el cual se emplea para la creación de las máquinas virtuales,	VirtualBox VMware Parallels
Sistema Operativo	Software que permite la interacción directa con el hardware del computador.	Linux Windows Mac
Hardware	Elemento físico en el cual se instala el sistema operativo.	Computador

Cuadro 2.2: Hipervisor Tipo 2
Fuente: Elaboración Propia

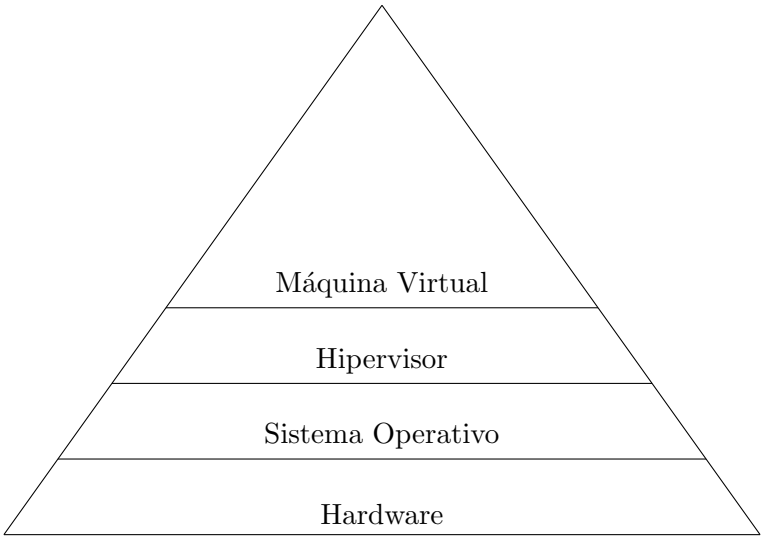


Figura 2.2: Hipervisor Tipo 2
Fuente: Elaboración Propia

2.3. Virtualización con qemu kvm

Según (Red Hat, s.f.-d), kvm (Kernel-based Virtual Machine), es una tecnología open source empleada en los sistemas Linux, que en gran escala permite a un máquina host ejecutar entornos de virtualización múltiples, transformando a Linux en un hipervisor altamente funcional.

Fue lanzado en el año 2006, y al siguiente se integró de manera nativa con el kernel de Linux otorgando la oportunidad de aprovechar por completo todas los recursos, soluciones y progresos del Sistema Operativo.

KVM transforma al Sistema Operativo Linux en un completo hipervisor de tipo 1, entregando herramientas para la gestión de los componentes de los sistemas huéspedes. QEMU según (QEMU, s.f.-a), es un emulador de máquinas virtuales genérico y de código abierto. Puede ser empleado de diversas formas, aunque la más común es como "user mode emulation", lanzando procesos compilados de un CPU a otro CPU. QEMU provee de un sin número de utilidades por la línea de comandos para facilitar la administración de este servicio.

2.3.1. Arquitecturas de host soportadas

CPU Architecture	Accelerators
Arm	kvm (64 bit only), tcg, xen
MIPS	kvm, tcg
PPC	kvm, tcg
RISC-V	tcg
s390x	kvm, tcg
SPARC	tcg
x86	hax, hvf (64 bit only), kvm, nvmm, tcg, whpx (64 bit only), xen

Cuadro 2.3: Arquitecturas de host kvm
Fuente: (QEMU, s.f.-b)

2.4. Contenedores

Como lo supo explicar (Red Hat, 2018), los contenedores son un conjunto de procesos aislados del sistema, los cuales, empleando imágenes diferentes, haciendo de estos, portables y manejables, unificando las etapas de: desarrollo, prueba y producción.

Agilizando los medios de desarrollo en comparación al tradicional, el cual requería replicar entornos de prueba.

Los contenedores no son iguales a la virtualización, ya que los mismos parten directamente del kernel del sistema Operativo. En otras palabras, los sistemas Linux, ejecutan contenedores Linux, y los sistemas Windows, ejecutan contenedores Windows.

Figura 2.3: Virtualización
Fuente: Elaboración Propia

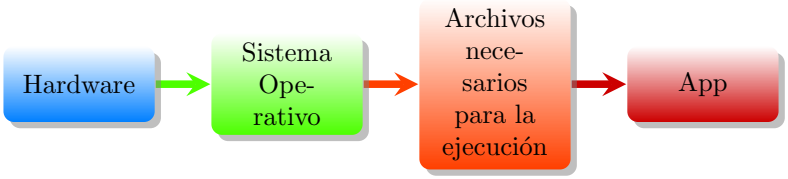


Figura 2.4: Contenedores
Fuente: Elaboración Propia

2.4.1. Docker

Según (Red Hat, s.f.-a), es una tecnología que permite la creación de contenedores Linux, constantemente apoyada por la comunidad open source de Docker, al igual que de Docker Inc., la cual es una empresa que hace más seguros los aportes de la comunidad de Docker, y los comparte con la misma. A demás que proporciona soluciones para entes empresariales. A menudo son confundidas, pero una es la comunidad y otra es la empresa con el mismo nombre. En un principio Docker se desarrolló a partir de los Linux Containers (LXC), sin embargo, esta al no ofrecer una buena experiencia al desarrollador, ni al usuario, se fue separando con el tiempo, mejorando los procesos de creación de contenedores, manejo de versiones, entre otros. Actualmente es posible ejecutarlo a parte de Linux, en Mac y en Windows.

2.4.2. Podman

Creado por Red Hat, Podman, por lo general la comunidad crea un alias en su sistema para que en vez de digitar podman en la línea de comandos, digiten docker, ya que funciona exactamente igual. Funciona en Linux como base nativa, en Windows con WSL (Windows Subsystem for Linux), o en Mac.

2.5. Kubernetes (k8s)

Kubernetes o también llamado k8s según (Red Hat, s.f.-c), es una plataforma de código abierto, la cual automatiza los procesos y operaciones de los contenedores Linux, los gestiona y monitorea constantemente. Crea clusters de hosts, los cuales, lo ayudan en el proceso de autogestion de los servicios. Mantiene una gran afinidad con diversos proveedores de servicios cloud, por lo que es la plataforma de preferencia para alojar aplicaciones nativas en los mismos.

Fue creado por Google, los cuales lo usaban para mantener sus servicios en producción, su precursora fue Borg, desarrollada igualmente por Google, la cual antes de la aparición de kubernetes fue la encargada de mantener los servicios de Google en un constante auge. Posteriormente la empresa la donó a la Cloud Native Computing Foundation, desde la cual el código fue liberado y constantemente recibe soporte por parte de la comunidad. Su API mantiene un sin número de opciones y acciones que se pueden llevar a cabo para automatizar tareas. Su unidad mínima es el Pod, el cual es el conjunto de uno o varios contenedores linux. Su base de datos es el etcd, el cual según expresa (Red Hat, s.f.-b), almacena y replica el estado de los clusters de kubernetes.

CAPÍTULO 3

Desarrollo

3.1. Conocimientos generales de kubernetes en orden secuencial

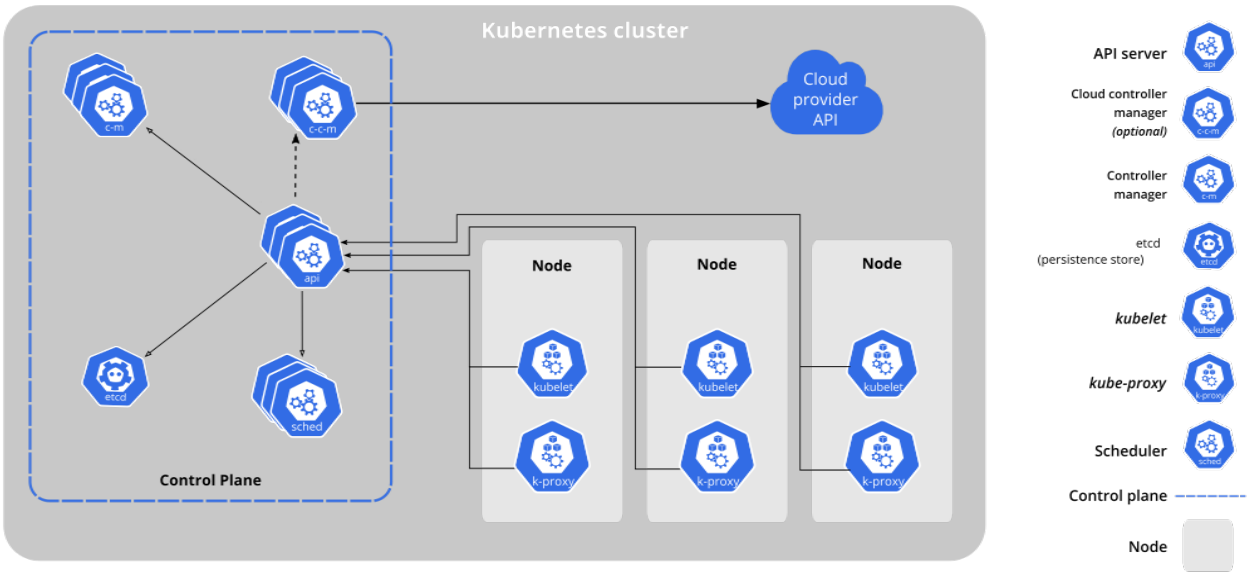


Figura 3.1: Componentes de Kubernetes
Fuente: (kubernetes.io, s.f.-b)

3.1.1. Kubernetes

Componentes del Control Plane

kube-apiserver Según (Nguyen & Kim, 2020), es el encargado de proveer un punto de entrada al plano de control de Kubernetes. Se encuentra preparado para escalar horizontalmente con la finalidad de balancear la carga entre diversas instancias.

etcd De acuerdo a (Larsson y col., 2020), la base de datos etcd es un distribuido almacén de clave-valor, el cual emplea el algoritmo Raft para consensos distribuidos.

kube-scheduler Como lo explica (kubernetes.io, s.f.-a), el kube-scheduler es un proceso del plano de control el cual asigna Pods a Nodos. Por ello como lo expresa (Wojciechowski y col., 2021), kube-scheduler considera que los Pods son independientes e iguala su distribución entre los diversos nodos del clúster, a excepción de que se especifique lo contrario en las reglas de afinidad y antiafinidad.

kube-controller-manager De acuerdo a (Martin, 2020), es el cual integra todos los controladores, incluido:

- Controlador de Nodo
- Controlador de Replicación
- Controlador de endpoints
- Controladores de tokens y cuentas de servicio

cloud-controller-manager Según (Buchanan y col., 2019), el cloud-controller-manager es similar al kube-controller-manager a excepción de que este maneja el respectivo controlador de procesos, el cual depende a su vez de un proveedor cloud subyacente. Es decir, que si Kubernetes se encuentra ejecutando en azure, u otra plataforma cloud, el cloud-controller-manager se encarga de que los balanceadores de carga de Kubernetes se encuentren operando sin problema.

Componentes de los Worker Nodes

kubelet De acuerdo a (Martin, 2020), este componente es el encargado de asegurarse de que los Pods pertenecientes al nodo se encuentra ejecutando correctamente.

kube-proxy Como lo afirma (Martin, 2020) Es el encargado de mantener las reglas de la red en los respectivos nodos para satisfacer las demandas del Servicio.

Container runtime De acuerdo a (kubernetes.io, s.f.-b), es el software encargado de ejecutar los contenedores. Kubernetes admite varios container runtimes, entre ellos:

- Docker
- containerd
- CRI-O

3.1.2. Kubectl

Es el cliente por línea de comandos que emplea kubernetes, para comunicarse con el clúster de kubernetes.

3.1.3. Pod

Unidad mínima de kubernetes. No pueden existir pods con el mismo nombre en un mismo namespace.

Estructura de un pod básico

Algoritmo 3.1: podprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Pod
metadata:
  name: podprueba
spec:
  containers:
  - name: podprueba
    image: nginx
    ports:
    - containerPort: 80
      name: podprueba
      protocol: TCP
```

3.1.4. Namespace

Es un espacio de trabajo donde se aplican sentencias de kubernetes.

Estructura de un namespace básico

Algoritmo 3.2: namespaceprueba.yaml
Fuente: Elaboración Propia

```
kind: Namespace
apiVersion: v1
metadata:
  name: namespaceprueba
```

3.1.5. Replicaset

Objeto de kubernetes empleado para escalar de manera horizontal, es decir, agrega mayor número de instancias de un pod, en vez de incrementar la capacidad de cómputo. Generalmente se emplean directamente desde los deployments, y no se aplica en un fichero independiente, sin embargo, es muy importante conocer su funcionamiento. Los replicaset tienen dos puntos importantes incluidos en ellos:

labels

Son necesarios para seleccionar y organizar un grupo de recursos. Empleados como pares de clave-valor.

selectors

Usan las etiquetas para identificar y agrupar un conjunto de objetos.

Estructura de un Replicaset básico

Algoritmo 3.3: replicasprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replicaprueba
  labels:
    app: prueba
spec:
  replicas: 3
  selector:
    matchLabels:
      app: prueba
  template:
    metadata:
      labels:
        app: prueba
    spec:
      containers:
      - name: nginxprueba
        image: nginx
        ports:
        - containerPort: 80
          name: podprueba
          protocol: TCP
```

3.1.6. Deployments

Son similares a los replicaset, que a su vez son de los objetos de kubernetes más usados. Proporcionan actualizaciones a las cargas de trabajo.

Estructura de un Deployments básico

Algoritmo 3.4: deploymentprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentprueba
  labels:
    app: prueba
spec:
  replicas: 3
  selector:
    matchLabels:
      app: prueba
  template:
    metadata:
      labels:
        app: prueba
    spec:
      containers:
      - name: nginxprueba
        image: nginx
        ports:
        - containerPort: 80
```

3.1.7. Servicios en kubernetes

Son los encargados de exponer una aplicación que se encuentra desplegada en kubernetes hacia los usuarios finales. Especifican cómo se accede a una aplicación. A los Pods, kubernetes les asignada una dirección IP propia, y balancea la carga entre ellos.

ClusterIP

Es la forma por defecto con la cual se expone una aplicación en kubernetes. Asigna una dirección IP interna al servicio en el clúster. Es útil para mantener servicios internos que no se desea que tengan visibilidad desde internet.

Estructura de un servicio tipo ClusterIP básico

Algoritmo 3.5: ClusterIPprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
 name: prueba2
 labels:
   app: prueba2
spec:
 ports:
  - protocol: TCP
    port: 80
 type: ClusterIP
 selector:
   app: prueba2
```

NodePort

Expone el servicio en un puerto estático en cada IP del nodo. Crea por defecto un Servicio ClusterIP, el cual se emplea para enrutar el tráfico y hacerlo accesible a los pods, por defecto para asignar un puerto toma un valor entre 30000-32767.

Estructura de un servicio tipo NodePort básico

Algoritmo 3.6: Nodeportprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
 name: prueba2
 labels:
   app: prueba2
spec:
 ports:
  - protocol: TCP
    port: 80
 type: NodePort
 selector:
   app: prueba2
```

LoadBalancer

Es usado en producción para exponer las aplicaciones, ya que emplea el cloud controller manager para establecer una conexión con el proveedor cloud, y de ese modo levantar automáticamente este tipo de servicio. Para implementaciones bare metal se recomienda emplear MetalLB que generalmente es utilizado en entornos locales.

Estructura de un servicio tipo LoadBalancer básico

Algoritmo 3.7: LoadBalancerprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
 name: prueba2
 labels:
   app: prueba2
spec:
 ports:
  - protocol: TCP
    port: 80
 type: LoadBalancer
 selector:
   app: prueba2
```

ExternalName

Asocia un DNS con un servicio. Se define con el parámetro spec.externalName. No es utilizado con frecuencia.

Estructura de un servicio tipo ExternalName básico

Algoritmo 3.8: ExternalNameprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
 name: prueba2
 labels:
   app: prueba2
spec:
 type: ExternalName
 externalName: db.example.prueba.com
```

3.1.8. ConfigMaps

De acuerdo a la documentación, es un objeto de kubernetes que se utiliza para almacenar datos no confidenciales en un formato clave-valor. Permite almacenar la configuración de otros objetos utilizados. Algunos objetos de kubernetes tienen la palabra “spec” para especificar ciertas sentencias, en tal caso el ConfigMap tiene una sección llamada “data”, para almacenar los ítems identificándolos como clave-valor.

Puede ser empleado para configurar un contenedor dentro de un Pod, las 4 maneras de utilizarlo de acuerdo a la documentación son:

- Argumento en la línea de comandos como entrypoint de un contenedor.
- Variable de entorno de un contenedor.
- Como fichero en un volumen de solo lectura para que lo lea la aplicación.
- Escribir el código para ejecutar dentro de un Pod que utiliza la API para leer el ConfigMap.

Estructura de un servicio tipo ConfigMap básico

Algoritmo 3.9: ConfigMapprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-prueba3
data:
  PORT: "80"
  SERVICE_HOST: "db.example.com"
  environment.properties:  |
    PORT=80
    SERVICE_HOST=db.example.com
```

3.1.9. Volúmenes

Son los encargados de mantener la persistencia de los datos especialmente cuando un contenedor se elimina o termina su ciclo, con dicha finalidad estos se mantienen y vuelven a ser utilizados cuando se reasigna otro contenedor o se reinicia el mismo. Docker igualmente soporta la persistencia de datos mediante volúmenes, aunque es menos controlado y con funcionalidades limitadas. Hay que aclarar que kubernetes soporta varios tipos de volúmenes. Un punto que se debe descartar es que no se puede montar un volumen dentro de otro.

Tipos de volúmenes

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- downwardAPI
- emptyDir
- fc (canal de fibra)
- flocker (deprecado)
- gcePersistentDisk
- glusterfs
- hostPath
- iscsi
- local
- nfs

- persistentVolumeClaim
- portworxVolume
- projected
- quobyte
- rbd
- secret
- storageOS
- vsphereVolume

StorageClasses

Permite a los administradores de kubernetes describir la clase de almacenamiento que se provee.

PersistentVolumes (PV)

Mantiene la persistencia de los datos, es decir si el recurso que empleaba este tipo de volumen es terminado, como por ejemplo la eliminación de un pod, este recurso se mantiene, y no desaparece, a excepción que sea eliminado manualmente, permitiendo la reutilización del mismo y su preservación. Es definido por parte de los administradores.

Ciclo de vida de un volumen y claim

- Provisioning
- Binding
- Using
- Storage Object in Use Protection
- Reclaiming
- Retain
- Reserving a PersistentVolume
- Expanding Persistent Volumes Claims
- Delete

Tipos de volúmenes persistentes

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- csi
- fc
- gcePersistentDisk

- glusterfs
- hostPath (no funciona en un cluster multinodo)
- iscsi
- local
- nfs
- portworxVolume
- rbd
- vsphereVolume

Estructura de un volumen tipo pv básico

Algoritmo 3.10: pvprueba.yaml
Fuente: (kubernetes.io, s.f.-d)

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pvprueba
spec:
 capacity:
  storage: 5Gi
 volumeMode: Filesystem
 accessModes:
  - ReadWriteOnce
 persistentVolumeReclaimPolicy: Recycle
 storageClassName: slow
 mountOptions:
  - hard
  - nfsvers=4.1
 nfs:
  path: /direccion/del/nfs/server
  server: direccion ip del servidor nfs
```

PersistentVolumeClaim (PVC)

Se emplea para mantener la persistencia de los datos declarando los atributos a preservar. Es definido por parte de los administradores, y es el objeto de kubernetes encargado de los volúmenes más usados.

Estructura de un volumen tipo pvc básico

Algoritmo 3.11: pvcprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvcprueba
spec:
  storageClassName: volumenpvc
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
```

3.1.10. Secretos

A diferencia del ConfigMap, este objeto de kubernetes sirve para almacenar secretos, es decir, información con cierto nivel de confidencialidad. De tal modo que existe reducción del riesgo de exposición accidental de este tipo de información. Es recomendable encriptar aquellas claves para poderlos exportar. Este tipo de recurso se puede crear desde la línea de comandos con kubectl:

Desde ficheros que tengan la información requerida

Algoritmo 3.12: Sentencias para crear un secreto desde ficheros
Fuente: Elaboración Propia

```
$ echo -n 'administrador' > ./username.txt
$ echo -n 'contrasenia' > ./password.txt
$ kubectl create secret generic nombresecret --from-file=./username.txt --from-file=
    ./password.txt
```

Directamente desde la línea de comando

Algoritmo 3.13: Sentencias para crear un secreto desde la terminal
Fuente: Elaboración Propia

```
$ kubectl create secret generic nombresecret --from-literal=username='usuario'
    --from-literal=password='contrasenia'
```

Por conciderar, la creación de este tipo de elementos con mayor frecuencia es mediante el empleo de archivo yaml.

Estructura de un secret básico

1. Encriptar datos en base64
2. Creación del archivo que contenga el secreto de kubernetes

Algoritmo 3.14: Encriptar datos en base64
Fuente: Elaboración Propia

```
$ echo -n 'administrador' | base64
YWRtaW5pc3RyYWRvcg==
$ echo -n 'contrasenia' | base64
Y29udHJhc2VuaWE=
```

.

Algoritmo 3.15: secretprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Secret
metadata:
 name: secretprueba
type: Opaque
data:
 username: YWRtaW5pc3RyYWRvcg==
 password: Y29udHJhc2VuaWE=
```

3.1.11. INGRESS

Es un objeto de kubernetes que permiten acceder a los recursos del cluster, exponiendo las rutas HTTP Y HTTPS desde fuera del mismo. El enrutamiento del enrutamiento es controlado por las reglas definidas en los recursos del Ingress. Pueden proveer:

- LoadBalancer
- Terminación SSL
- Virtual hosting basado en nombres

Los Ingress class y los Ingress Controllers son definidos por los administradores del cluster de kubernetes, con el fin de otorgar mayor disponibilidad a los usuarios.

Algoritmo 3.16: ingresssprueba.yaml
Fuente: Elaboración Propia

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingresssprueba
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /pruebaingress
        pathType: Prefix
        backend:
          service:
            name: wildflyprueba
            port:
              number: 80
```

Ingress class

Pueden ser implementados por diferentes controladores, a menudo con caracteristicas diferentes, para poder exponer las cargas de trabajo hacia el exterior.

Ingress Controllers

Para que los recursos del Ingress funcionen, el cluster requiere emplear un Ingress Controller para en ejecución. A diferencia de otros controladores que se ejecutan como parte del binario del kube-controller-manager, es controlador no inicia automáticamente con el cluster. Se puede tener uno o varios Ingress Controllers en el mismo cluster, y se los distinguirá mediante el ingress class.

Kubernetes de acuerdo a su documentación, soporta y mantiene los ingress controller de:

- AWS
- GCE
- NGINX

Ingress Controllers adicionales (Provienen de terceros)

- AKS Application Gateway Ingress Controller
- VAmbassador
- VApache APISIX ingress controller
- VAvi Kubernetes Operator
- VBFE Ingress Controller
- VCitrix ingress controlle
- VContour
- VEnRoute
- VEasegress IngressController
- VF5 BIG-IP Container Ingress Services for Kubernetes
- VGloo
- VHAProxy Ingress
- VHAProxy Ingress Controller
- VIstio Ingress
- VKong Ingress Controller
- VNGINX Ingress Controller
- VTraefik Kubernetes Ingress provider
- VTyk Operator
- Voyager

3.1.12. Addons

De acuerdo a la documentación respectiva (kubernetes.io, s.f.-b), son los encargados de incrementar la funcionalidad de kubernetes, a parte de de otorgar mayores beneficios en la gestión de los servicios del antes mencionado. Por lo general son proporcionados por parte de terceros. Pueden ser gestionados mediante Deployments, ReplicationControllers u otros.

DNS

Si bien otros addons no son estrictamente requeridos, todos los clusters de kubernetes deberían tener un cluster DNS, ya que constantemente es requerido.

Web UI (Dashboard)

Es la herramienta de kubernetes que otorga un entorno gráfico en la web para administrar los clusters de kubernetes y gestionar sus servicios de forma visible, aunque comúnmente, se emplea la linea de comandos, ya que el Dashboard únicamente es de propósito general.

3.1.13. Comando para aplicar ficheros yaml en kubernetes

Algoritmo 3.17: Código

```
$ kubectl apply -f <fichero>.yaml
```

Algoritmo 3.18: Ejemplo

```
$ kubectl apply -f namespaceprueba.yaml
```

3.1.14. Comando para eliminar elementos creados a partir de ficheros yaml en kubernetes

Algoritmo 3.19: Código

```
$ kubectl delete -f <fichero>.yaml
```

Algoritmo 3.20: Ejemplo

```
$ kubectl delete -f namespaceprueba.yaml
```

3.2. Prerrequisitos Generales

3.2.1. Máquinas virtuales

Como corresponde de manera adecuada por demostración existe la necesidad de crear máquinas virtuales, las cuales serán empleadas como nodos del clúster de kubernetes, sin embargo, la forma correcta de crear este tipo de solución tecnológica es mediante la incorporación de servidores locales Linux enlazados por una red LAN, la cual en los hogares es el tipo de red por defecto, no obstante para ejemplificar su funcionamiento bastará con máquinas virtuales funcionando en una computadora personal, la misma que contiene el sistema operativo Ubuntu 20.04.

Por preferencia para el levantamiento de este tipo de instancias, se hará uso de vagrant, el cual crea máquinas virtuales de manera declarativa, mediante el archivo llamado Vagrantfile, de tal modo que se omite el proceso de formar una máquina virtual paso por paso (véase anexo 7).

Del mismo modo, si por elección se desea emplear otro método de virtualización, el proceso funcionara debidamente, bien sea levantando una máquina virtual con:

- VirtualBox
- VMware
- KVM (Recomendado para Linux)

3.2.2. Tecnología de virtualización

Con el efecto de otorgar una funcionalidad adecuada a KVM se requiere activar previamente la tecnología de vitalización (véase anexo 4.1).

3.2.3. Levantar las máquinas virtuales con vagrant empleando como proveedor a VirtualBox

Vagrant requiere necesariamente de un proveedor para virtualizar, en tal caso se dará uso de VirtualBox (véase anexo 6), posteriormente se enlaza el virtualizador con vagrant (véase anexo 7)

Para no mezclar archivos, es preferible crear un directorio aparte en la dirección que se desee guardar la configuración de vagrant. Se entiende que el sistema operativo Linux con el cual se labora es Linux, por tal motivo se deben emplear los comandos:

Algoritmo 3.21: Crea un directorio

```
$ mkdir prueba
```

Algoritmo 3.22: Accede al directorio

```
$ cd prueba
```

Al ubicarse en el directorio deseado, con el comando:

```
$ vagrant init
```

Se crea automáticamente un archivo llamado Vagrantfile, el cual contiene las sentencias necesarias para dar inicio a una máquina virtual con los valores por defecto.

Para mayor información de las diversas máquinas que se pueden virtualizar empleando vagrant, es recomendado en el buscador realizar la siguiente consulta:

```
$ vagrant boxes
```

La misma que como resultado retornará la página de “Vagrant Cloud”, la cual contendrá un sin fin de vagrant boxes, las anteriormente mencionadas son imágenes que la comunidad ha aportado y están preconfiguradas.

Debido a que la ruta url correspondiente a “vagrant boxes” suele cambiar, es recomendable acceder a ella realizando la consulta antes mencionada.

Para levantar una máquina virtual en vagrant con un “box” por defecto, se puede emplear el comando:

Algoritmo 3.23: Levantar una máquina virtual en vagrant con datos por defecto

```
$ vagrant init <box-a-emplear>
```

Algoritmo 3.24: Ejemplo

```
$ vagrant init generic/centos8
```

De tal manera que Crea un archivo Vagrantfile con los valores por defecto para dar funcionalidad a una máquina virtual de Centos 8.

Continuamente con el comando:

```
$ vagrant up
```

Se levanta una máquina virtual de acuerdo a las sentencias del Vagrantfile.

Por consiguiente, con el comando:

```
$ vagrant ssh
```

Se permite el ingreso a la máquina iniciada.

Si la máquina virtual requiere configuraciones adicionales desde el archivo Vagrantfile, se las puede realizar.

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "generic/centos8"
  # config.vm.network :forwarded_port, guest: 80, host: 8089
  config.vm.network :private_network, ip: "10.0.4.3"
  config.vm.hostname = "minikubelocal"
  config.vm.provider "virtualbox" do |vb|
    vb.name = "centos8minikube"
    vb.cpus=3
    vb.memory=3072
  end
  # Instalacion de ufw para el tratamiento del firewall
  # ufw habilita los puertos requeridos de k3s y el puerto 80 para nginx o apache
     para comprobar que las reglas se encuentren levantads.
  config.vm.provision "shell", inline: <<-SHELL
    echo "Cambiando de Centos8 a Centos8 Stream"
    dnf --disablerepo '*' --enablerepo=extras swap centos-linux-repos centos-stream-
        repos -y
    dnf distro-sync -y

    echo "Limpiar cache"
    yum clean all
    yum clean metadata
    yum clean packages

    echo "------ Instalacion de ufw para el firewall y nginx para exponer un servicio
         por defecto-------"

    yum -y install nginx

    echo "------------Iniciar y habilitar ufw, e iniciar nginx---------------"
    systemctl start nginx

    echo "----------------- Habilitar puertos de kubernetes -----------------"
    firewall-cmd --add-port=80/tcp
```

```
    firewall-cmd --add-port=8080/tcp --permanent

    echo "Control plane"
    firewall-cmd --add-port=6443/tcp --permanent
    firewall-cmd --add-port=2379/tcp --permanent
    firewall-cmd --add-port=2380/tcp --permanent
    firewall-cmd --add-port=10250/tcp --permanent
    firewall-cmd --add-port=10259/tcp --permanent
    firewall-cmd --add-port=10257/tcp --permanent

    echo "---------------Worker Nodes----------------------"
    firewall-cmd --add-port=30000-32767/tcp --permanent
    echo "--------------Version de centos-------------------"
    cat /etc/os-release
  SHELL
end
```

Dependiendo la situación, se puede hacer uso de un sin número de funcionalidades de vagrant, por ello los comandos previamente mensionados y los que a continuación se muestran, tienen un nivel de importancia, debido a que son requeridos para realizar una gestión básica de esta herramienta.

Algoritmo 3.25: Lista los boxes de vagrant en la máquina

```
$ vagrant box list
```

Algoritmo 3.26: Detiene la máquina de vagrant ejecutándose

```
$ vagrant halt
```

Algoritmo 3.27: Detruye la máquina de vagrant establecida

```
$ vagrant destroy
```

Con lo referente al comando previo, de preferencia se debe tomar en cuenta si no se encuentran archivos en la máquina virtual a eliminar, ya que se destruirán en conjunto con la misma.

Es relevante mencionar que, por defecto vagrant se integra automáticamente con VirtualBox, sin embargo, es posible dar uso a otros virtualizadores, los cuales requieren de configuraciones adicionales para su funcionamiento.

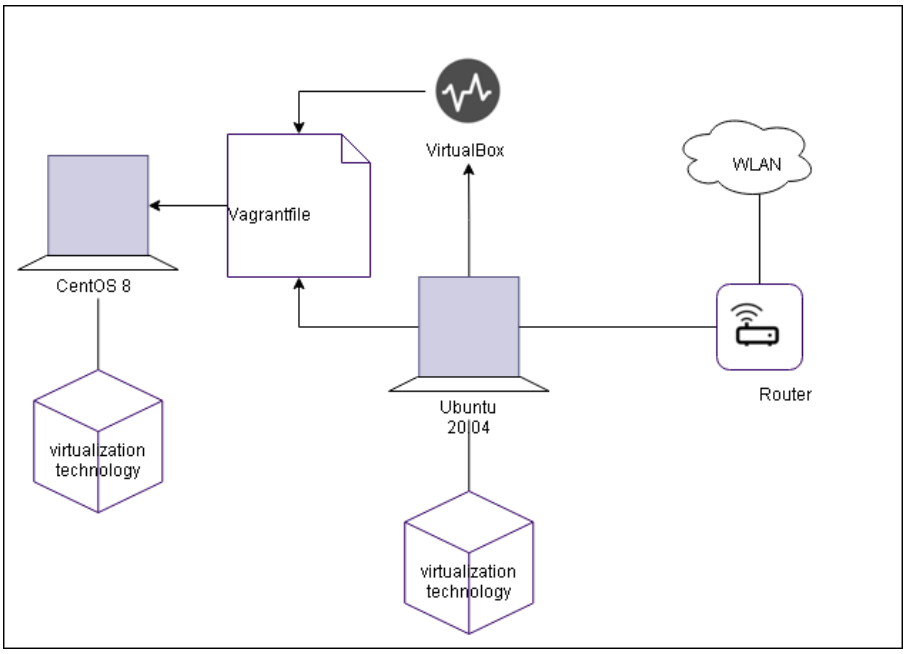


Figura 3.2: Virtualización anidada con VirtualBox
Fuente: Elaboración Propia

3.2.4. Prerrequisitos para la instalación de minikube en CentOS 8 integrado a kvm

Si el sistema operativo CentOS a emplear se encuentra instalado directamente en una máquina física, se requiere únicamente activar la tecnología de virtualización la cual fue previamente mencionada.

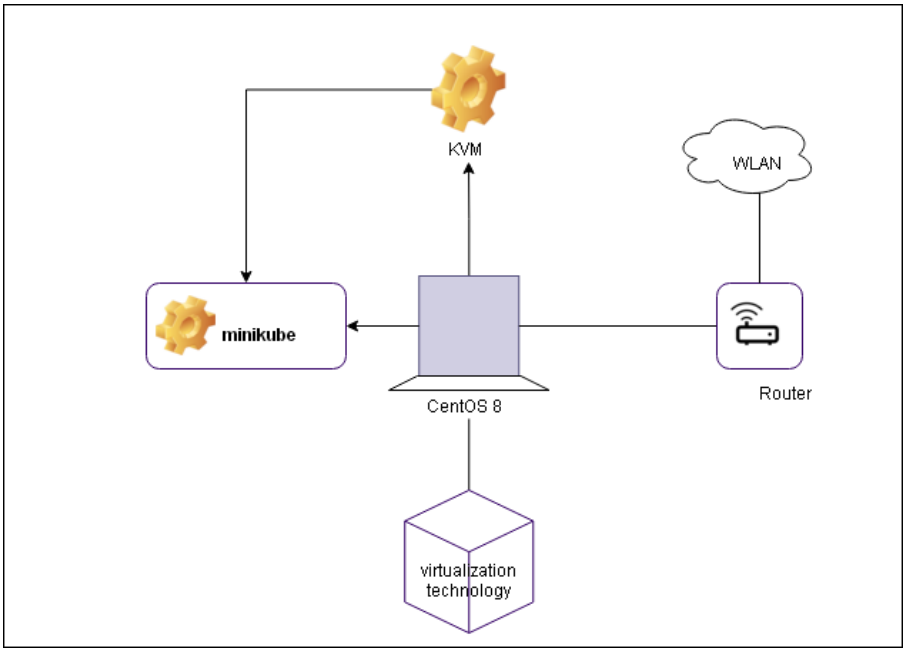


Figura 3.3: Minikube en una máquina física de CentOS con KVM
Fuente: Elaboración Propia

Si el sistema operativo se encuentra en una máquina virtual, la tecnología de virtualización se debe activar accediendo directamente a la BIOS de esa máquina virtual.

Si la máquina virtual a emplear se la procede a crear con KVM, la tecnología de virtualización se hereda automáticamente.

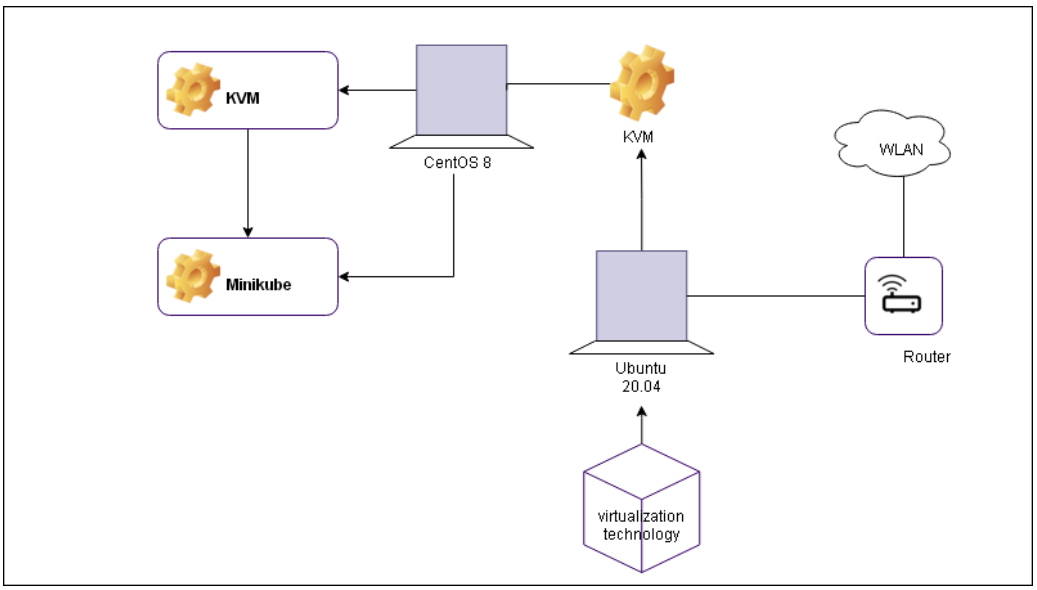


Figura 3.4: Minikube en una máquina virtual CentOS levantada con kvm
Fuente: Elaboración Propia

De conformidad con la documentación oficial de minikube (kubernetes.io, s.f.-c), los requisitos mininos con los cuales debe contar el sistema operativo son:

- 2 CPUs o más.
- 2GB de memoria libre.
- 20GB de espacio en el disco libre.
- Conexión a internet
- Contenedor o administrador de máquinas virtuales, tales como:
  - ➢ Docker
  - ➢ Hyperkit
  - ➢ Hyper-V
  - ➢ KVM
  - ➢ Parallels
  - ➢ Podman
  - ➢ VirtualBox
  - ➢ VMware Fusion/Workstation

3.2.5. Virtualización anidada de vagrant con VirtualBox

En la sección, en la cual se explicó como levantar una máquina virtual de VirtualBox empleando vagrant, se dio a conocer un archivo llamado Vagrantfile, el mismo que contenía las configuraciones necesarias para el funcionamiento del cliente. Sin embargo, para que la tecnología de virtualización se incorpore a este cliente es necesario realizar modificaciones mediante la línea de comandos.

Por accesibilidad, los comandos posteriormente presentados, otorgarán una secuencia exacta para habilitar el servicio necesario.

Algoritmo 3.28: Levantar la máquina con vagrant normalmente

```
$ vagrant up
```

Algoritmo 3.29: Ingresar a la máquina

```
$ vagrant ssh
```

Algoritmo 3.30: Apagar la máquina

```
$ sudo poweroff
```

Algoritmo 3.31: Comprobar el estado de la máquina (El estado deseado es 'apagado')

```
$ vagrant status
```

Algoritmo 3.32: Obtener el nombre de las máquinas virtuales existentes (Extraer el nombre de la máquina de vagrant)

```
$ VBoxManage list vms
```

Algoritmo 3.33: Habilitar la tecnología de virtualización en la máquina de VirtualBox deseada

```
$ VBoxManage modifyvm centos8minikube --nested-hw-virt on
```

Para verificar que la tecnología de virtualización se encuentre funcionando es neceario levantar la máquina de vagrant, ingresar a la misma y posteriormente digitar el comando:

Algoritmo 3.34: Verificar la habilitación de la tecnología de virtualización

```
$ cat /proc/cpuinfo | egrep -c "(vmx|vtx)"
3
```

Es obligatorio levantar por primera vez máquina con las configuraciones por defecto, debido a que se requiere de la presencia de la instancia del cliente, es decir, la máquina virtual en la lista de VirtualBox, caso contrario no es posible habilitar la tecnología de virtualización por el motivo de que el cliente deseado aún no existe.

3.2.6. Instalación de minikube en CentOS integrado a kvm

Esta sección redacta de forma ordenada los pasos para la incorporación de minikube a CentOS según lo específicado (véase anexo 9).

3.2.7. Comandos básicos de minikube

Iniciar

Algoritmo 3.35: Iniciar el servicio de minikube

```
$ minikube start
```

Algoritmo 3.36: Iniciar el servicio de minikube con el driver de kvm

```
$ minikube start --driver=kvm2
```

Algoritmo 3.37: Iniciar el servicio de minikube con diferentes proporciones de memoria y cpu

```
$ minikube start --memory 2400 --cpus=2
```

Dashboard

Algoritmo 3.38: Obtener acceso al dashboard de minikube

```
$ minikube dashboard
```

Addons

Algoritmo 3.39: Lista de addons en minikube

```
$ minikube addons list
```

Algoritmo 3.40: Habilitar el ingress en minikube

```
$ minikube addons enable ingress
```

Red

Algoritmo 3.41: Conexión ssh a minikube

```
$ minikube ssh
```

Algoritmo 3.42: Obtener la IP de minikube

```
$ minikube ip
```

Algoritmo 3.43: Habilitar el LoadBalancer en minikube hacia el localhost de la máquina

```
$ minikube tunnel
```

Eliminar

Algoritmo 3.44: Eliminar el cluster de kubernetes levantado con minikube

```
$ minikube delete
```

Estado

Algoritmo 3.45: Estado del cluster de kubernetes levantado con minikube

```
$ minikube status
```

Detener

Algoritmo 3.46: Detener el cluster levantado con minikube

```
$ minikube stop
```

Obtener

Algoritmo 3.47: Obtener información de un servicio

```
$ minikube service <serviciorequerido>
```

Algoritmo 3.48: Verificar si existen actualizaciones para minikube

```
$ minikube update-check
```

Algoritmo 3.49: Obtener la versión de minikube

```
$ minikube version
```

3.2.8. Requisitos de las imagenes a emplear

Para publicar las imagenes creadas se da la necesidad de previamente disponer de una cuenta de Docker Hub.

Wildfly

Instrucciones previas

- Del archivo wildfly instalado localmente copiar la carpeta "modules", la misma que contiene las configuraciones del datasource, al directorio principal en el cual se encuentra el Dockerfile.
- Copiar el archivo jar de postgres, y colocarlo en el directorio del Dockerfile.
- Copiar del directorio local de wildfly '/wildfly/standalone/configuration/' el archivo 'standalone.xml', y colocarlo en el directorio del Dockerfile.
- **Nota)** Si en el interior del archivo 'standalone.xml' al final se encuentran:

Algoritmo 3.50: Líneas a eliminar
Fuente: Elaboración Propia

```
<deployments> <deployment name="postgresql-42.2.23.jar" runtime-name="
    postgresql-42.2.23.jar"> <content sha1="9
    cb217a3d5b640567ed7c6e8c11f389613c81c4d"/> </deployment> </deployments>
```

Es necesario **eliminar estas líneas** ya que genera errores por duplicidad en la creación de la imagen

Dockerfile

Algoritmo 3.51: Contenido del archivo Dockerfile de wildfly
Fuente: Elaboración Propia

```
FROM jboss/wildfly:22.0.0.Final
RUN pwd
RUN /opt/jboss/wildfly/bin/add-user.sh wesly wesly --silent
RUN rm -rf /opt/jboss/wildfly/modules/*
COPY ./modules/ /opt/jboss/wildfly/modules/
RUN rm /opt/jboss/wildfly/standalone/configuration/standalone.xml
COPY ./standalone.xml /opt/jboss/wildfly/standalone/configuration/
COPY ./postgresql-42.2.23.jar /opt/jboss/wildfly/standalone/deployments/
COPY ./PracticaEAR.ear  /opt/jboss/wildfly/standalone/deployments/
RUN rm -rf /opt/jboss/wildfly/standalone/configuration/standalone_xml_history/*
CMD ["/opt/jboss/wildfly/bin/standalone.sh", "-b", "0.0.0.0", "-bmanagement", "
    0.0.0.0"]
```

Despliegue al docker hub

Algoritmo 3.52: Construir la imagen

```
$ docker build . -t <userid_del_dockerhub>/<nombre_de_la_imagen>:<version>
```

Algoritmo 3.53: Ejemplo

```
$ docker build . -t cuentawj/wildflypostgres:2.0
```

Algoritmo 3.54: Iniciar session en docker desde la terminal

```
$ docker login
```

Algoritmo 3.55: Publicar la imagen

```
$ docker push cuentawj/wildflypostgres:2.0
```

Adminer

Esta imagen se implementará en kubernetes directamente desde los repositorios, por ende no requiere customización.

Presentación

Instrucciones previas

Se requiere levantar un proyecto de javascript creado en base a:

- nodejs
- vitejs
- revealjs

Dockerfile

Algoritmo 3.56: Contenido del archivo Dockerfile de la presentación
Fuente: Elaboración Propia

```
FROM node:17-alpine3.12
WORKDIR /usr/src/presentacion
COPY package*.json ./
COPY . .
RUN yarn install
RUN yarn run build
RUN mv tesis.pdf dist/
RUN npm install -g http-server
CMD [ "http-server", "dist" ]
```

Despliegue a docker hub

Algoritmo 3.57: Construir la imagen

```
$ docker build . -t <userid_del_dockerhub>/<nombre_de_la_imagen>:<version>
```

Algoritmo 3.58: Ejemplo

```
$ docker build . -t cuentawj/presentacion:v1
```

Algoritmo 3.59: Iniciar session en docker desde la terminal

```
$ docker login
```

Algoritmo 3.60: Publicar la imagen

```
$ docker push cuentawj/presentacion:v1
```

3.3. Creación de ficheros yaml

En base a la sección previa, de acuerdo al segmento denominado **requisitos de las imagenes a emplear**. Para la gestión de aquellas imagenes publicadas en Docker Hub, con el fin de mantenerlas en producción, se hara uso de kubernetes.

Los comandos aportados a continuación, son parte de aquellos que se posicionan en la sección de **Implementación**, cabe destacar que debido a su importante en la creación de archivos Yaml se dara un uso adecuado en este segmento, con los cuales se facilita el proceso de creación y despliegue de servicios en kubernetes.

3.3.1. Wildfly

Creación de servicio

Algoritmo 3.61: Crear despliegue del wildfly customizado
Fuente: Elaboración Propia

```
$ kubectl create deployment wildfly-prueba --image=cuentawj/wildflypostgres:2.0
    --replicas=3 -o yaml --dry-run=client >> deployWildfly.yaml
```

Algoritmo 3.62: deployWildfly.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: wildfly-prueba
  name: wildfly-prueba
spec:
  replicas: 3
  selector:
    matchLabels:
      app: wildfly-prueba
  strategy: {}
  template:
    metadata:
      labels:
        app: wildfly-prueba
    spec:
      containers:
      - image: cuentawj/wildflypostgres:2.0
        name: wildfly
```

Exposición del servicio

Algoritmo 3.63: Exponer el despliegue del wildfly customizado
Fuente: Elaboración Propia

```
$ kubectl expose deployment wildfly-prueba --type=NodePort --port=8080 -o yaml
    --dry-run=client >> exposeWildfly.yaml
```

Algoritmo 3.64: exposeWildfly.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
  name: wildfly-prueba
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: wildfly-prueba
  type: NodePort
```

3.3.2. Adminer

Creación de servicio

Algoritmo 3.65: Crear despliegue de adminer
Fuente: Elaboración Propia

```
$ kubectl create deploy servicioadminer --image=adminer --replicas=2  -o yaml
    --dry-run=client >> deployAdminer.yaml
```

Algoritmo 3.66: deployAdminer.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: servicioadminer
  name: servicioadminer
spec:
  replicas: 2
  selector:
    matchLabels:
      app: servicioadminer
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: servicioadminer
    spec:
      containers:
      - image: adminer
        name: adminer
```

Exposición del servicio

Algoritmo 3.67: Exponer el despliegue de adminer
Fuente: Elaboración Propia

```
$ kubectl expose deploy servicioadminer --type=NodePort --port=8080  -o yaml
    --dry-run=client >> exposeAdminer.yaml
```

Algoritmo 3.68: exposeAdminer.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: servicioadminer
  name: servicioadminer
spec:
  ports:
  - port: 8080
```

```
    protocol: TCP
    targetPort: 8080
  selector:
    app: servicioadminer
  type: NodePort
status:
  loadBalancer: {}
```

3.3.3. Presentación

Creación de servicio

Algoritmo 3.69: Crear despliegue de la presentacion elaborada
Fuente: Elaboración Propia

```
$ kubectl create deploy presentacion --image=cuentawj/presentacion:v1 --replicas=2
    -o yaml --dry-run=client >> deployPresentacion.yaml
```

Algoritmo 3.70: deployPresentacion.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: presentacion
  name: presentacion
spec:
  replicas: 1
  selector:
    matchLabels:
      app: presentacion
  template:
    metadata:
      labels:
        app: presentacion
    spec:
      containers:
      - image: wescor/presentacion:v4
        name: presentacion
```

Exposición del servicio

Algoritmo 3.71: Exponer el despliegue de la presentacion
Fuente: Elaboración Propia

```
$ kubectl expose deploy presentacion --type=NodePort --port=8080  -o yaml --dry-run=
    client >> exposePresentacion.yaml
```

Algoritmo 3.72: exposePresentacion.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: presentacion
  name: presentacion
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: presentacion
  type: NodePort
```

3.3.4. Base de datos PostgreSQL

Creación del ConfigMap

Algoritmo 3.73: configmap.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: prueba
  name: postgres-configmap
  labels:
    app: wildflyapp
data:
  POSTGRES_DB: postgresdb
  POSTGRES_USER: postgres
```

Creación de los Secret

Algoritmo 3.74: secret.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Secret
metadata:
  name: secretos-postgres
  namespace: prueba
data:
  contrasenia_root: YWRtaW5pc3RyYWRvcgo=
  contrasenia_usuario: dXN1YXJpbwo=
```

Creación del PersistentVolume

Algoritmo 3.75: pvbasededatos.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: PersistentVolume
metadata:
  namespace: prueba
  name: postgres-persistentvolume
  labels:
    type: local
    app: wildflyapp
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/mnt/data"
```

Creación del PersistentVolumeClaim

Algoritmo 3.76: pvcpostgres.yaml
Fuente: Elaboración Propia

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  namespace: prueba
  name: postgres-persistentvolumeclaim
  labels:
    app: wildflyapp
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Exposición del servicio

Algoritmo 3.77: servicepostgres.yaml
Fuente: Elaboración Propia

```
apiVersion: v1
kind: Service
metadata:
  namespace: prueba
  name: postgresdb
  labels:
    app: wildflyapp
spec:
  ports:
    - name: postgres
```

```
      port: 5432
      nodePort: 30432
  type: NodePort
  selector:
    app: wildflyapp
```

Creación del servicio

Algoritmo 3.78: postgresdeploy.yaml
Fuente: Elaboración Propia

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: prueba
  name: postgresdb
spec:
  selector:
    matchLabels:
      app: wildflyapp
  replicas: 1
  template:
    metadata:
      labels:
        app: wildflyapp
    spec:
      containers:
        - name: postgres
          image: postgres:latest
          imagePullPolicy: "IfNotPresent"
          env:
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: secretos-postgres
                  key: contrasenia_usuario
            - name: POSTGRES_USER
              valueFrom:
                configMapKeyRef:
                  name: postgres-configmap
                  key: POSTGRES_USER
            - name: POSTGRES_DB
              valueFrom:
                configMapKeyRef:
                  name: postgres-configmap
                  key: POSTGRES_DB
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgrespvcenlace
      volumes:
        - name: postgrespvcenlace
          persistentVolumeClaim:
            claimName: postgres-persistentvolumeclaim
```

3.3.5. dry-run

Este parámetro, sirve para simular la creación de una sentencia de Kubernetes sin aplicarlo al clúster.

3.4. Implementación.

Tomando como partida los archivos creados en la sección **Creación de ficheros yaml**, para que las sentencias se apliquen al clúster de Kubernetes es necesario emplear la linea de comandos.

El orden correcto para la implementación de estos servicios es:

1. Aplicar las declaraciones del fichero encargado de la creación del servicio.
2. Aplicar las declaraciones del fichero encargado de exponer el servicio.

3.4.1. Wildfly

Algoritmo 3.79: Aplicar las declaraciones del fichero encargado de la creación del servicio

```
$ kubectl apply -f deployWildfly.yaml
```

Algoritmo 3.80: Aplicar las declaraciones del fichero encargado de exponer el servicio

```
$ kubectl apply -f  exposeWildfly.yaml
```

3.4.2. Adminer

Algoritmo 3.81: Aplicar las declaraciones del fichero encargado de la creación del servicio

```
$ kubectl apply -f deployAdminer.yaml
```

Algoritmo 3.82: Aplicar las declaraciones del fichero encargado de exponer el servicio

```
$ kubectl apply -f exposeAdminer.yaml
```

3.4.3. Presentación

Algoritmo 3.83: Aplicar las declaraciones del fichero encargado de la creación del servicio

```
$ kubectl apply -f deployPresentacion.yaml
```

Algoritmo 3.84: Aplicar las declaraciones del fichero encargado de exponer el servicio

```
$ kubectl apply -f exposePresentacion.yaml
```

3.4.4. Base de datos PostgreSQL

Algoritmo 3.85: Crear el namespace prueba

```
$ kubectl create ns prueba
```

Algoritmo 3.86: Aplicar las declaraciones de todos los ficheros encargados del servicio

```
$ kubectl apply -f .
```

3.4.5. Ejecutar todos los ficheros que se encuentran en un directorio

Algoritmo 3.87: Aplicar todas las sentencias de los ficheros

```
$ kubectl apply -f .
```

3.4.6. Eliminar todos los servicios que se crean en base a los ficheros en un directorio

Algoritmo 3.88: Eliminar todo en base a los ficheros

```
$ kubectl delete -f .
```

3.5. Empleo de la API de kubernetes (k8s).

Si bien Kubernetes es un orquestador con una gran variedad de funcionalidades, lo cual a nivel empresarial proporciona beneficios inimaginables, conocer su funcionamiento requiere de un elevado tiempo de estudio, no obstante, obtener una base centrada de conocimientos es posible, por ello ulteriormente se presentan una serie de comandos para efectuar una administración básica del clúster, cabe declarar que las sentencias a continuación presentadas, no son las únicas existentes, pero aporta información para un conocimiento esencial de Kubernetes.

3.5.1. Informar

Algoritmo 3.89: Mostrar la información del cluster de kubernetes

```
$ kubectl cluster-info
```

3.5.2. Obtener

Algoritmo 3.90: Listar los nodos del respectivo cluster

```
$ kubectl get nodes
```

Algoritmo 3.91: Listar los servicios

```
$ kubectl get service
```

Algoritmo 3.92: Listar los servicios

```
$ kubectl get svc
```

Algoritmo 3.93: Listar los pods

```
$ kubectl get pods
```

Algoritmo 3.94: Listar los deployments

```
$ kubectl get deployments
```

Algoritmo 3.95: Listar los namespaces

```
$ kubectl get namespaces
```

Algoritmo 3.96: Listar los namespaces

```
$ kubectl get ns
```

Algoritmo 3.97: Listar los pods de un determinado namespace en este caso llamado 'tesis'

```
$ kubectl get pods -n tesis
```

Algoritmo 3.98: Listar todo el contenido del cluster de kubernetes

```
$ kubectl get  all -o wide
```

Algoritmo 3.99: Listar los secretos almacenados en kubernetes

```
$ kubectl get secrets
```

Algoritmo 3.100: Listar los replicaset

```
$ kubectl get rs
```

3.5.3. Exponer

Algoritmo 3.101: Exponer un deployment en este caso uno llamado 'wildfly-prueba'

```
$ kubectl expose deployment wildfly-prueba --port=8080  --type=NodePort
```

3.5.4. Eliminar

Algoritmo 3.102: Eliminar un servicio en este caso uno llamado primer-servicio

```
$ kubectl delete service primer-servicio
```

Algoritmo 3.103: Eliminar un deployment en este caso uno llamado wildfly-prueba

```
$ kubectl delete deployment wildfly-prueba
```

Algoritmo 3.104: Eliminar un namespace

```
$ kubectl delete ns  <nombre-del-namespace>
```

Algoritmo 3.105: Ejemplo de la eliminación de un namespace

```
$ kubectl delete ns prueba
```

3.5.5. Crear

Algoritmo 3.106: Crear un namespace

```
$ kubectl create namespace prueba
```

Algoritmo 3.107: Crear un secreto

```
$ kubectl create secret generic bdd-pass --from-literal=password=miclave
```

Algoritmo 3.108: Crear pod en un namespace llamado 'prueba' partiendo de un fichero yaml

```
$ kubectl  apply -f podprueba.yaml -n prueba
```

3.5.6. Ejecutar

Algoritmo 3.109: Acceder a un Pod de kubernetes

```
$ kubectl exec -it wildfly-01 -- bash
```

Algoritmo 3.110: Obtener el DNS por pod

```
$ kubectl exec <nombre-del-pod>  -- cat /etc/resolv.conf
```

Algoritmo 3.111: Ejecutar un comando sin ingresar al servicio desplegado con kubernetes

```
$ kubectl exec servicioadminer-8689ab -- curl http://servicioadminer-8689ab:8080
```

3.5.7. Describir

Algoritmo 3.112: Mostrar información detallada de un pod en este caso llamado wildfly-01

```
$ kubectl describe pod wildfly-01
```

Algoritmo 3.113: Ver la ip de la primera concurrencia en los pods

```
$ kubectl describe pod <nombre-del-pod> | grep "IP"| head -n 1
```

3.5.8. Escalar

Algoritmo 3.114: Escalar un deployment a 4 replicas

```
$ kubectl scale --replicas=4 deployment wildfly-prueba
```

Algoritmo 3.115: Escalar un deployment a 2 replicas

```
$ kubectl scale --replicas=2 deploy/servicioadminer
```

3.5.9. Forwardear puertos (Expone un pod o servicio para su visualización)

Algoritmo 3.116: Forwardear el Puerto

```
$ kubectl port-forward <nombre-del-pod> <puerto-externo-deseado>:<puerto-del-pod>
```

Algoritmo 3.117: Ejemplo de forwardeo de un Puerto

```
$ kubectl port-forward prueba-5575fff496-k89xp 8080:80
```

3.5.10. Logs (Salida de lo que ejecuto un pod)

Algoritmo 3.118: Obtener una salida normal

```
$ kubectl logs <nombre-del-pod>
```

Algoritmo 3.119: Obtener la salida y mantenerlo en escucha para observar los cambios

```
$ kubectl logs <nombre-del-pod> -f
```

3.5.11. Actualizar

Algoritmo 3.120: Actualizar

```
$ kubectl set image deployment/<nombre-deployment> <containers-name>=<imagen>:version
```

Algoritmo 3.121: Ejemplo de actualizar

```
$ kubectl set image deployment/prueba  pruebaimagen=wescor/nginxmodificado:v2
```

3.5.12. Historial de un deployment

Algoritmo 3.122: Incluye el número de revisión de ese deployment

```
$ kubectl rollout history deployment/<nombre-deployment>
```

Algoritmo 3.123: Ejemplo

```
$ kubectl rollout history deployment/deployprueba
```

3.5.13. Regresar el deployment a un estado

Algoritmo 3.124: De acuerdo a la revisión obtenida en el historial se puede seleccionar a cuál se desea regresar

```
$ kubectl rollout undo deployment/deployprueba --to-revision=3
```

3.5.14. Estado

Algoritmo 3.125: Ver el estado de la actualización de un deployment

```
$ kubectl rollout status -w deployment/<nombre-deployment>
```

Algoritmo 3.126: Ejemplo

```
$ kubectl rollout status -w deployment/deployprueba
```

3.6. Clúster de kubernetes con k3s

Para su instalación se requiere de la documentación oficial de k3s (k3s.io, s.f.), la cual se encuentra en su página web oficial https://k3s.io/

3.6.1. Componentes requeridos

- Un Raspberry Pi 4, con un sistema operativo Linux instalado, específicamente raspbian os (véase anexo 8), para simular un servidor externo.
- Una máquina virtual de linux levantada con vagrant (véase el anexo 7.2), para la creación de un nodo de kubernetes k3s, la distribución puede ser la de preferencia. No requiere de la tecnología de virtualización habilitada en la máquina virtual.

3.6.2. Vagrantfile a emplear

En caso de que se emplee a vagrant, el contenido del archivo correspondiente será el siguiente:

```
# -*- mode: ruby -*- ""
# vi: set ft=ruby :
Vagrant.configure("2") do |config|
  config.vm.box = "generic/centos7"

  config.vm.define "master" do |nodosconfig|
    nodosconfig.vm.hostname= "master"
    nodosconfig.vm.network :private_network, ip: "10.0.0.3"
    #nodosconfig.vm.network :private_network, type: "dhcp"
    nodosconfig.vm.provider "virtualbox" do |vb|
      vb.name="master"
      vb.cpus=1
      vb.memory=1024
    end
  end

  (1..NUMERODENODOS).each do |numbernode|
    config.vm.define "nodo#{numbernode}" do |nodosconfig|
      nodosconfig.vm.hostname= "node#{numbernode}"
      nodosconfig.vm.network :private_network, ip: "10.0.0.#{numbernode+3}"
      #nodosconfig.vm.network :private_network, type: "dhcp"
      nodosconfig.vm.provider "virtualbox" do |vb|
        vb.name="nodo#{numbernode}"
        vb.cpus=1
        vb.memory=1024
      end
    end
```

```
  end
  # Instalacion de ufw para el tratamiento del firewall
  # ufw habilita los puertos requeridos de k3s y el puerto 80 para nginx o apache
  # para comprobar que las reglas se encuentren levantads.
  config.vm.provision "shell", inline: <<-SHELL
    yum -y install ufw
    yum -y install nginx

    systemctl start nginx
    systemctl enable ufw
    systemctl start ufw

    echo "reglas de firewall de k3s"
    ufw allow 80
    ufw allow 8472/udp
    ufw allow 6443,10250/tcp
    ufw allow 2379:2380/tcp
    ufw allow  30000:32767/tcp
  SHELL
end
```

3.6.3. Proceso

Desde el Raspberry Pi 4

Descargar e instalar automáticamente el binario de k3s.

```
$ curl -sfL https://get.k3s.io | sh -
```

Para comprobar que el nodo se encuentre listo se debe esperar alrededor de 30 segundos.

```
$ k3s kubectl get node
```

Desde una máquina conectada a la red obtenemos

Obtener el token de acceso a k3s empleando ssh:

```
$ ssh <ip-del-servidor-con-k3s-instalado> -l <usuario> "
    commando-a-ejecutar-en-el-servidor"
```

Aplicar el comando de la siguiente manera:

```
$ ssh 192.168.1.12 -l pi "sudo cat /var/lib/rancher/k3s/server/node-token"
```

Desde la máquina virtual levantada con vagrant

Se procede a digitar el comando, para enlazar la máquina de vagrant con el servidor externo:

```
$ curl -sfL https://get.k3s.io | K3S_URL=https://<
    ip-del-servidor-con-k3s-instalado>:6443 K3S_TOKEN=
    <token-obtenido-del-servidor-con-k3s-instalado> sh -
```

El cual funciona de la siguiente manera:

```
$ curl -sfL https://get.k3s.io | K3S_URL=https://192.168.1.12:6443K3S_TOKEN=
    K105f62bd95a7770a99def6448ab7b2::server:532b26a4 sh -
```

3.6.4. Diagrama

Muestra la estructura del clúster (véase el anexo 11).

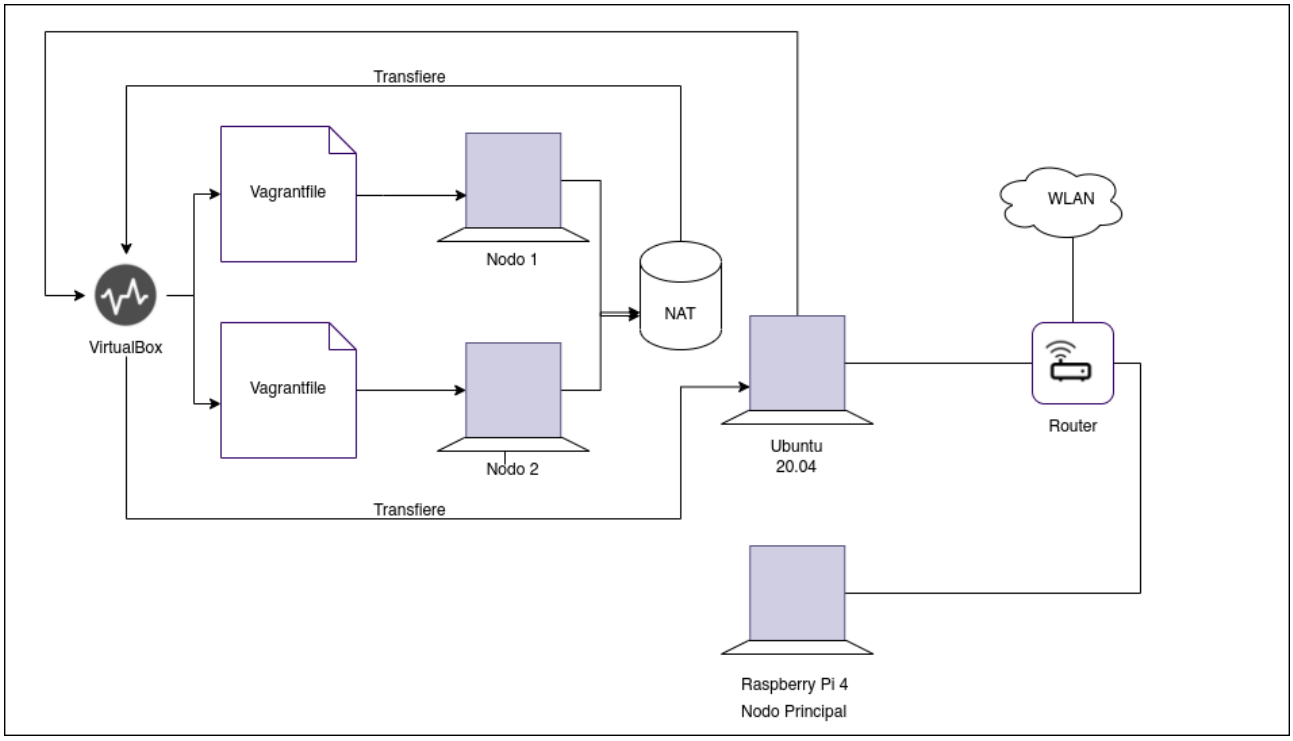


Figura 3.5: Clúster de Kubernetes con k3s
Fuente: Elaboración Propia

3.7. Controlar el clúster de kubernetes k3s externo desde la máquina local con Ubuntu 20.04

3.7.1. Requisitos

- Instalar kubectl en la máquina local.
- Crear en la máquina local el directorio $\sim/.kube$ de no existir.

3.7.2. Desde el Raspberry Pi 4

Copiar el archivo k3s.yaml al directorio actual con el punto agregado al final del comando.

```
$ cp /etc/rancher/k3s/k3s.yaml .
```

Poner el puerto 2222 a la escucha empleando netcat, y compartir el archivo k3s.yaml.

```
$ nc -lvp 2222 < k3s.yaml
```

Importante

Si existe fallos en uno de los pasos anteriormente mostrados, puede ser necesario cambiar los permisos del archivo k3s.yaml, y volver a ejecutar el pertinente proceso.

```
$ sudo chmod 644  /etc/rancher/k3s/k3s.yaml
```

3.7.3. Desde la máquina local con Linux

Copiar desde el servidor externo el archivo k3s.yaml con el comando estructurado del siguiente modo:

```
$ nc <ip_rasberry> <puerto> > archivo
```

Por ende, la sentencia quedaría definida de la siguiente manera:

```
$ nc 192.168.1.12 2222 > k3s.yaml
```

Posteriormente se debe copiar el archivo k3s.yaml a la dirección $\sim/.kube/config$, la cual corresponde a la localización del archivo de configuración de kubernetes, por ello previamente se requiere tener instalado kubectl en la máquina local.

```
$ cp k3s.yaml ~/.kube/config
```

Se procede a editar el archivo config empleando el editor de preferencia, en este caso en particular se emplea vim.

```
$ vim ~/.kube/config
```

En la sección en la cual se especifica el server en el archivo config se debe cambiar la dirección que se encuentra especificada por la dirección del servidor externo:

```
server: https://<ip_rasberry>:6443
```

De forma que la sentencia se emplea de la siguiente manera:

```
server: https://192.168.1.12:6443
```

Se guarda las modificaciones hechas en el archivo y se procede a salir del mismo:

```
:wq
```

Finalmente se ejecuta el comando para obtener los nodos del clúster de kubernetes, y con ello se puede verificar el funcionamiento del servicio.

```
$ kubectl get nodes
```

3.7.4. Diagrama

Muestra la estructura del clúster controlado desde una máquina local(véase el anexo 12).

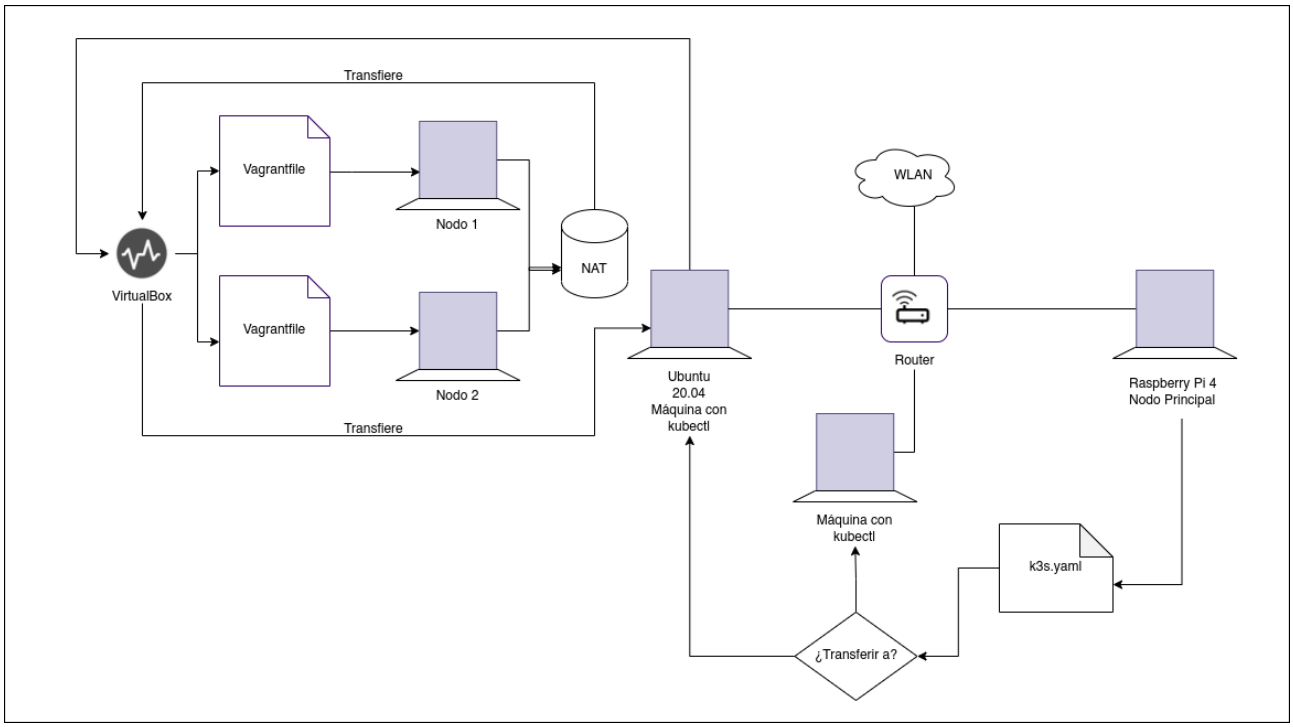


Figura 3.6: Control externo del Clúster de Kubernetes con k3s
Fuente: Elaboración Propia

3.8. Pruebas empleando Kubernetes desde un proveedor cloud

3.8.1. Requisitos

- kubectl instalado en una máquina local.
- cuenta de okteto cloud (Se requiere de una cuenta de github).

3.8.2. Beneficios

- No requiere de una tarjeta de credito.
- DNS automático.
- Archivo de configuración descargable.
- Entorno con mayor cercania a un ambiente de productos.
- Provee una interfaz gráfica.

3.8.3. Desventajas

- La disponibilidad de los servicios se ven afectados, si no se emplea, caso contrario mantendra el servicio en constante ejecución.
- Para la versión publica unicamente expone servicios ClusterIP, aunque en el manifiesto del fichero Yaml se lo denomine con NodePort u otro.

3.8.4. Configuración

Se incorpora a kubectl como herramienta encargada de interactuar con los clusters de Kubernetes(véase anexo 9.2), al proveedor okteto cloud(véase anexo 10).

3.8.5. Validar

Algoritmo 3.127: Describir los contextos y marcar el actual

```
$ kubectl config get-contexts
```

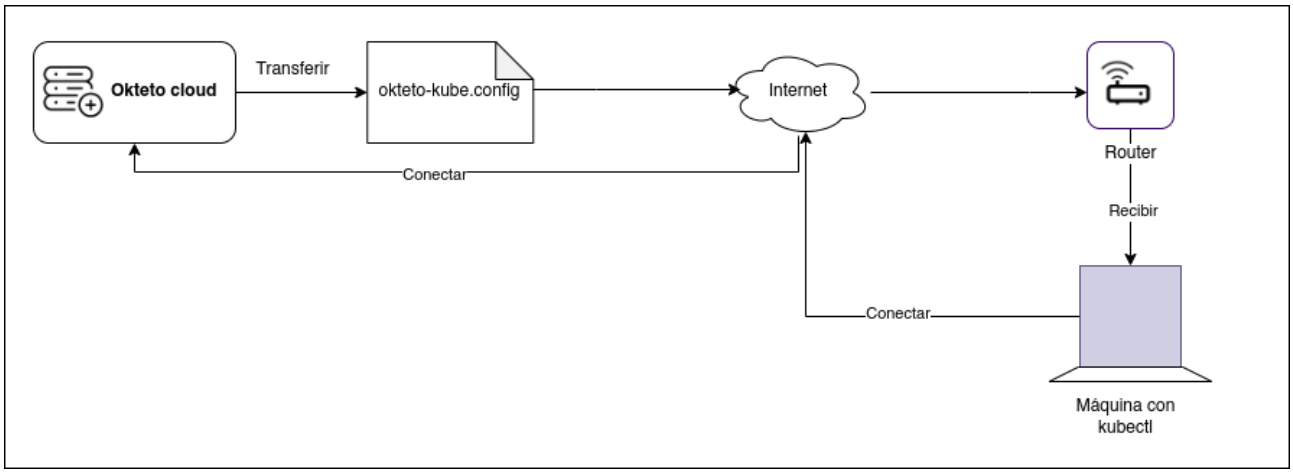


Figura 3.7: Kubernetes empleando a okteto cloud
Fuente: Elaboración Propia

CAPÍTULO 4

Resultados

4.1. Validación de resultados

Para la aplicación del cuestionario de Usabilidad de Sistemas Informáticos CSUQ, se tomó como base la adaptación al español del mismo (Hedlefs y col., 2016). Con el fin de obtener los resultados, se diseñaron **16** preguntas representativas, puntuadas en un rango del 1 al 7, donde 1 corresponde a *"Totalmente en desacuerdo"* y el valor de **7** corresponde a *"Totalmente de acuerdo"*.

N°	***Preguntas***	***1***	***2***	***3***	***4***	***5***	***6***	***7***
1	*En general, estoy satisfecho con lo fácil que es utilizar Kubernetes.*							
2	*Fue simple usar esta herramienta.*							
3	*Soy capaz de completar mi trabajo rápidamente utilizando esta herramienta.*							
4	*Me siento cómodo utilizando Kubernetes.*							
5	*Fue fácil aprender a utilizar Kubernetes.*							
6	*Creo que me volví experto rápidamente utilizando Kubernetes.*							
7	*La herramienta muestra mensajes de error que me dicen claramente cómo resolver los problemas.*							
8	*Cada vez que cometo un error utilizando Kubernetes, lo resuelvo fácil y rápidamente.*							
9	*La información (como ayuda en línea, mensajes en pantalla o en la terminal y otra documentación) que provee Kubernetes es clara.*							
10	*Es fácil encontrar en Kubernetes la información que necesito.*							
11	*La información que proporciona Kubernetes fue efectiva ayudándome a completar las tareas.*							
12	*La organización de la información de Kubernetes en la terminal o por pantalla fue clara.*							
13	*La interfaz de Kubernetes fue placentera.*							
14	*Me gustó utilizar Kubernetes.*							
15	*Kubernetes tuvo todas las herramientas que esperaba que tuviera.*							
16	*En general, estuve satisfecho con la herramienta.*							

Cuadro 4.1: Cuestionario CSUQ
Fuente: Elaboración Propia

4.1.1. Elementos empleados para la validación

- Google Forms para elaborar el Cuestionario CSUQ.
- Previo a la aplicación del cuestionario, se procedió a realizar un taller demostrativo a un grupo de 29 estudiantes de los últimos niveles de la carrera de Software, de la Facultad de Ingeniería en Ciencias Aplicadas (FICA) de la Universidad Técnica del Norte. La agenda del taller fue la siguiente:
  - ➢ Empleo de Kubernetes desde el servidor de virtualización de la Facultad de Ingeniería en Ciencias Aplicadas con minikube.
  - ➢ Empleo de Kubernetes en un entorno de producción con okteto cloud.
  - ➢ Diseño de un script de Python para mostrar mensajes explicativos por la terminal.

4.1.2. Script de python diseñado

Algoritmo 4.1: Script de Python para mostrar los mensajes por la terminal
Fuente: Elaboración Propia

```
import sys


#print(sys)

def concepto():
    print("\033[1;91m Conceptos \033[1;00m \n")

listasec2=["Levantar una Base de datos postgres con kubernetes",
        "Levantar adminer para conectarse a la base de datos",
        "Instalar psql para acceder a la base de datos desde la terminal",
        "Instalar wildfly",
        "Integrar a wildfly con postgres",
        "Crear y desplegar una imagen de wildfly configurada",
        "Consumir la imagen desde kubernetes usando a minikube y okteto cloud"
        ]

listanombresec2 = ["base",
        "adminer",
        "psql",
        "wildfly",
        "integrar",
        "desplegar",
        "consumir"
        ]

listanombreconsumir=["minikube","okteto"]
listaconsumir=["Desde minikube","Desde okteto cloud"]

listarecordar=["Toca destacar que para desplegar la solucion, la base de datos no se
    debe encontrar en un entorno local por ello, se emplea a ElephantSQL.",
        "Previamente se requiere de una cuenta en Docker Hub para publicar la imagen
            que se genere."
        ]

listadesplegartomarencuenta=[
        "recordatorio",
```

```
        "dockerfile"
        ]


#print(sys.argv)
#print(listasec2.index("Se integrara a wildfly con postgres"))
if sys.argv[1]=="gracias":
    print("""Gracias""")

if (sys.argv[1]=="inicio"):
    print("\033[1;91m En esta seccion se procedera a:\033[1;00m \n")
#    print(len(sys.argv))
    if len(sys.argv)==2:
        for value in listasec2:
            print("\033[1;91m "+str(listasec2.index(value)+1)+".- \033[1;32m"+value+"
                \033[1;00m\n")
    elif len(sys.argv)>=3:
        for nombrehacer in listanombresec2:
            if sys.argv[2]==nombrehacer:
                posicion= listanombresec2.index(nombrehacer)
                print("\033[1;91m "+str(listanombresec2.index(nombrehacer)+1)+".-
                    \033[1;32m"+listasec2[posicion]+"\033[1;00m\n")

                if sys.argv[2]=="consumir":
                    for consumodesde in listanombreconsumir:
                        if consumodesde in sys.argv:
                            if sys.argv[3]==consumodesde:
                                posicionelementoconsumir=listanombreconsumir.index(
                                    consumodesde)
                                print("\033[0;49;34m {} \033[1;00m\n".format(
                                    listaconsumir[posicionelementoconsumir]))
                elif sys.argv[2]=="desplegar":
                    for despuesdedesplegar in listadesplegartomarencuenta:
                        if despuesdedesplegar in sys.argv:
                            if (despuesdedesplegar=="recordatorio") & (sys.argv[3]=="
                                recordatorio"):
                                print("\033[1;91m {} \033[1;00m \n".format(
                                    despuesdedesplegar.upper()))
                                for datosrecordar in range(0,len(listarecordar)):
                                    print("\033[0;49;34m "+"{}.-".format(
                                        datosrecordar+1)+"\033[0;49;34m {}".format(
                                        listarecordar[datosrecordar])+"\033[1;00m\n")
                            elif (despuesdedesplegar=="dockerfile") & (sys.argv[3]=="
                                dockerfile"):
                                print("\033[1;91m {} \033[1;00m \n".format(
                                    despuesdedesplegar.upper()))
                                print("\033[0;49;34m "+"->"+"\033[0;49;34m Contenido
                                    del Dockerfile"+"\033[1;00m\n")



listaobjetoskubernetes=["configmap",
        "secret",
        "pv",
        "pvc",
        "deploy",
        "service"
        ]
listaconceptosobjetosdeKubernetes=["Guarda informacion que no es confidencial",
        "Guarda informacion confidencial",
```

```
        "Monta un volumen en el cluster",
        "Persiste la informacion para que al terminar el servicio los datos se
            preserven",
        "Configura el servicio",
        "Expone el servicio"
        ]
listanombreobjetoskubernetes=["ConfigMap",
        "Secret",
        "PersistentVolume",
        "PersistentVolumeClaim",
        "Deployment",
        "Service"
        ]


if sys.argv[1]=="concepto":
    for nombresob in listaobjetoskubernetes:
        if sys.argv[2]==nombresob:
            concepto()
            objetokube= listaobjetoskubernetes.index(nombresob)
            print("\033[1;91m {} ->  \033[1;32m {} \033[1;00m \n".format(
                listanombreobjetoskubernetes[objetokube],
                listaconceptosobjetosdeKubernetes[objetokube] ))
```

4.1.3. Diagrama del script creado

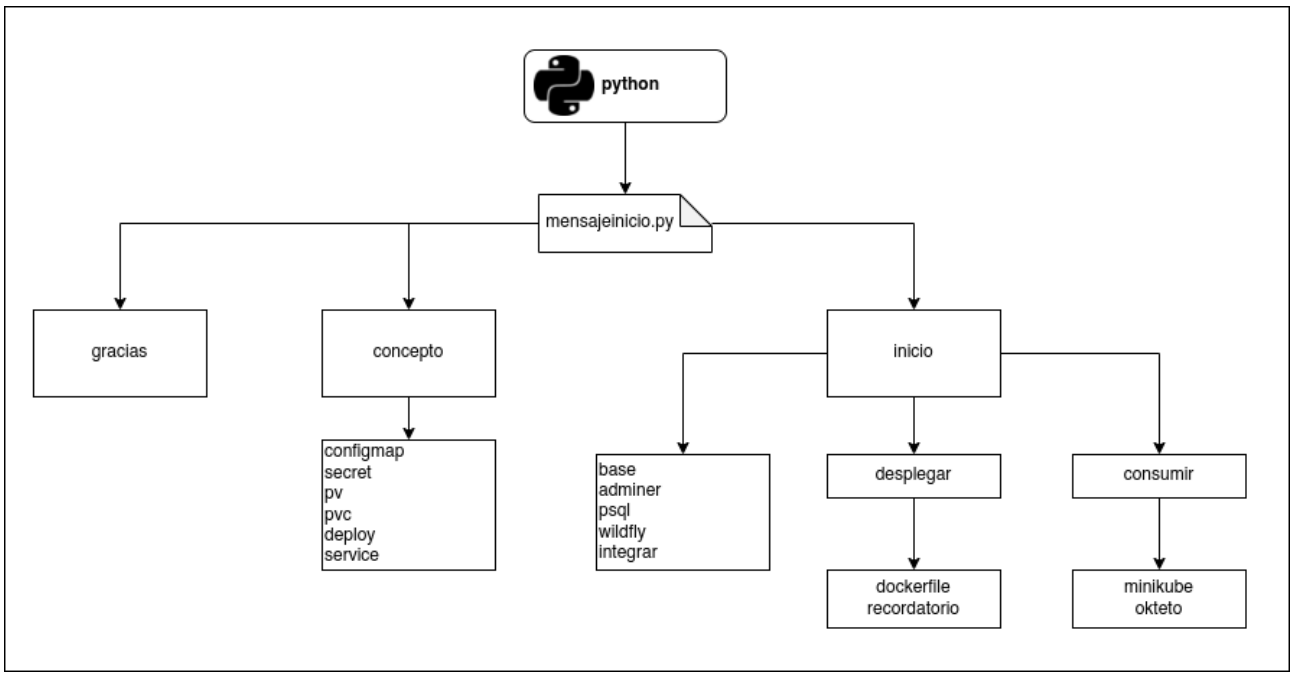


Figura 4.1: Funcionamiento del script de python creado
Fuente: Elaboración Propia

4.1.4. Resultados de la encuesta

Luego de realizado el taller demostrativo, se aplicó el cuestionario CSUQ y con el propósito de distinguir apropiadamente el rango de puntuación con el cual labora el cuestionario CSUQ, en las gráficas presentadas a continuación, se lo denominará como *Rango CSUQ*. Proporcionalmente, la columna *Encuestados* tomará por referencia al área de estudio previamente denominado.

1.- En general, estoy satisfecho con lo fácil que es utilizar Kubernetes.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	4	12
4	7	28
5	5	25
6	12	72
7	1	7
	Promedio	4,965517241

Cuadro 4.2: Resultados de la primera pregunta CSUQ
Fuente: Elaboración Propia.

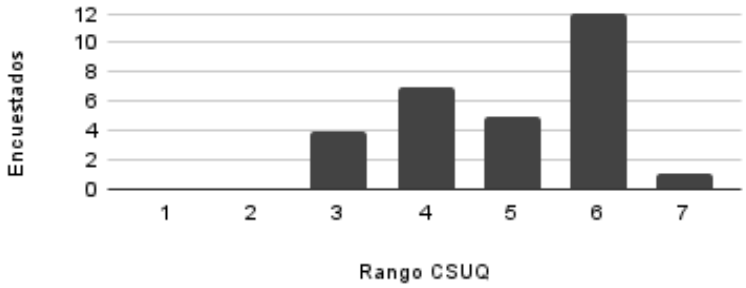


Figura 4.2: Resultados de la primera pregunta CSUQ.
Fuente: Elaboración Propia

2.- Fue simple usar esta herramienta.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	3	9
4	6	24
5	13	65
6	7	42
7	0	0
	Promedio	4,827586207

Cuadro 4.3: Resultados de la segunda pregunta CSUQ
Fuente: Elaboración Propia

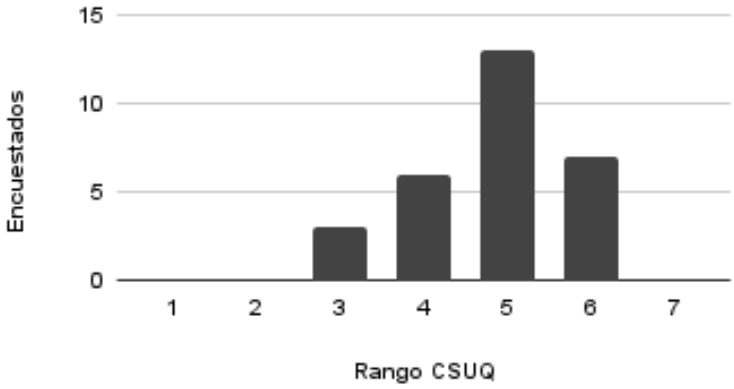


Figura 4.3: Resultados de la segunda pregunta CSUQ.
Fuente: Elaboración Propia

3.- Soy capaz de completar mi trabajo rápidamente utilizando esta herramienta.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	2	6
4	7	28
5	16	80
6	4	24
7	0	0
	Promedio	4,75862069

Cuadro 4.4: Resultados de la tercera pregunta CSUQ
Fuente: Elaboración Propia

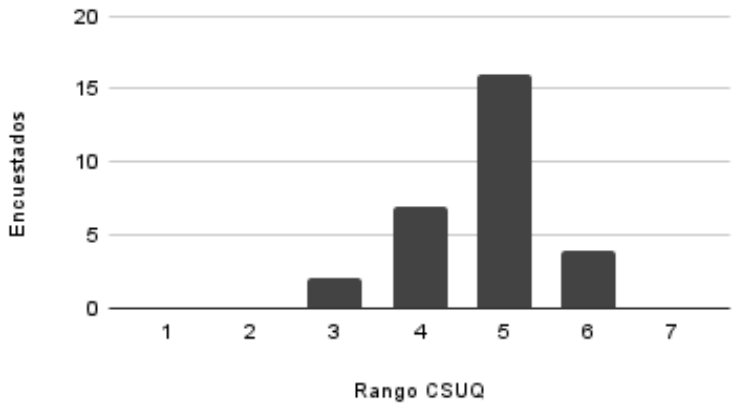


Figura 4.4: Resultados de la tercera pregunta CSUQ.
Fuente: Elaboración Propia

4.- Me siento cómodo utilizando Kubernetes.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	3	9
4	12	48
5	10	50
6	3	18
7	1	7
	Promedio	4,551724138

Cuadro 4.5: Resultados de la cuarta pregunta CSUQ
Fuente: Elaboración Propia

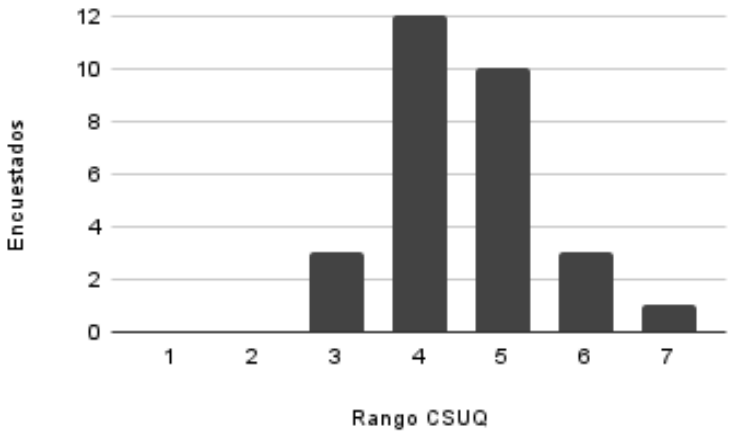


Figura 4.5: Resultados de la cuarta pregunta CSUQ.
Fuente: Elaboración Propia

5.- Fue fácil aprender a utilizar Kubernetes.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	4	12
4	6	24
5	13	65
6	5	30
7	1	7
	Promedio	4,75862069

Cuadro 4.6: Resultados de la quinta pregunta CSUQ
Fuente: Elaboración Propia

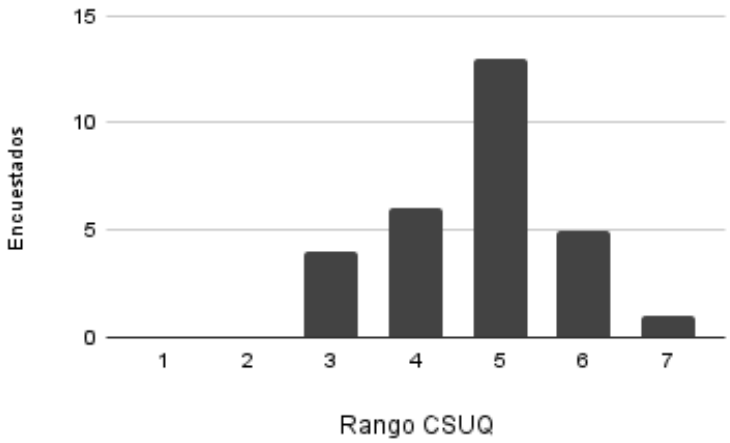


Figura 4.6: Resultados de la quinta pregunta CSUQ.
Fuente: Elaboración Propia

6.- Creo que me volví experto rápidamente utilizando Kubernetes.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	1	2
3	9	27
4	15	60
5	4	20
6	0	0
7	0	0
	Promedio	3,75862069

Cuadro 4.7: Resultados de la sexta pregunta CSUQ
Fuente: Elaboración Propia

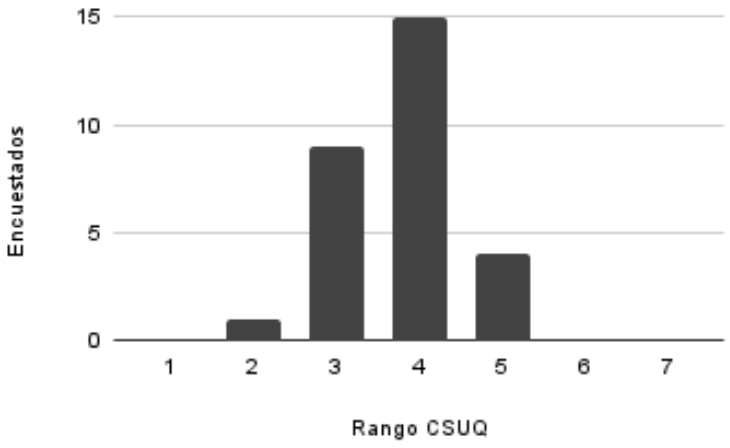


Figura 4.7: Resultados de la sexta pregunta CSUQ.
Fuente: Elaboración Propia

7.- La herramienta muestra mensajes de error que me dicen claramente cómo resolver los problemas.

Rango CSUQ	*Encuestados*	*Total*
1	1	1
2	0	0
3	2	6
4	6	24
5	13	65
6	7	42
7	0	0
	Promedio	4,75862069

Cuadro 4.8: Resultados de la séptima pregunta CSUQ
Fuente: Elaboración Propia

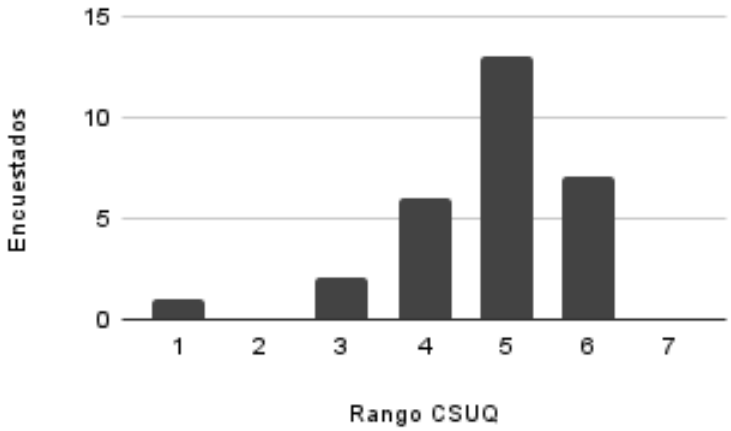


Figura 4.8: Resultados de la séptima pregunta CSUQ.
Fuente: Elaboración Propia

8.- Cada vez que cometo un error utilizando Kubernetes, lo resuelvo fácil y rápidamente.

Rango CSUQ	*Encuestados*	*Total*
1	0	0
2	0	0
3	5	15
4	5	20
5	15	75
6	4	24
7	0	0
	Promedio	4,620689655

Cuadro 4.9: Resultados de la octava pregunta CSUQ
Fuente: Elaboración Propia

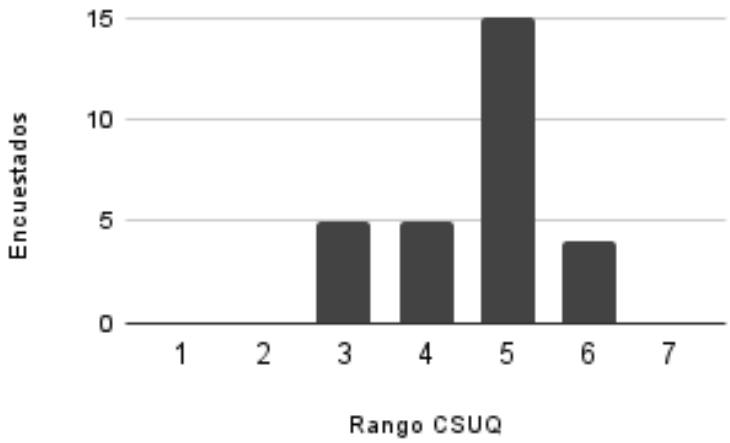


Figura 4.9: Resultados de la octava pregunta CSUQ.
Fuente: Elaboración Propia

9.- La información (como ayuda en línea, mensajes en pantalla o en la terminal y otra documentación) que provee Kubernetes es clara.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	3	9
4	8	32
5	6	30
6	10	60
7	2	14
	Promedio	5

Cuadro 4.10: Resultados de la novena pregunta CSUQ
Fuente: Elaboración Propia

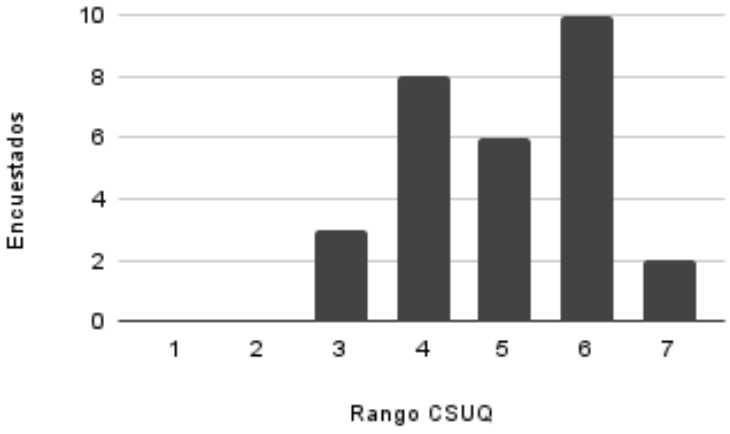


Figura 4.10: Resultados de la novena pregunta CSUQ.
Fuente: Elaboración Propia

10.- Es fácil encontrar en Kubernetes la información que necesito.

Rango CSUQ	*Encuestados*	*Total*
1	0	0
2	0	0
3	3	9
4	8	32
5	9	45
6	6	36
7	3	21
	Promedio	4,931034483

Cuadro 4.11: Resultados de la décima pregunta CSUQ
Fuente: Elaboración Propia

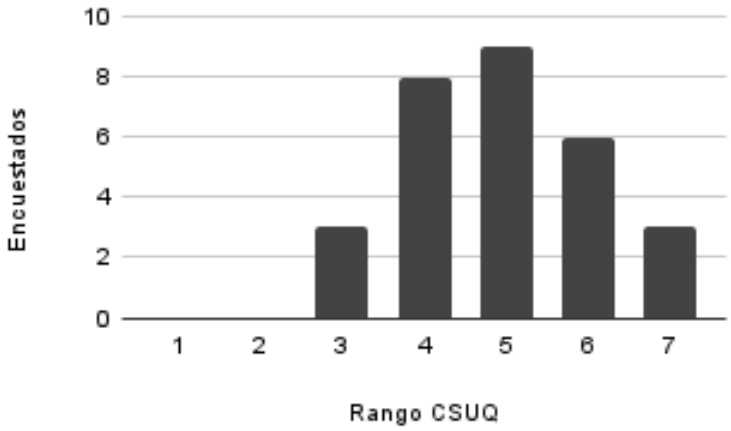


Figura 4.11: Resultados de la décima pregunta CSUQ.
Fuente: Elaboración Propia

11.- La información que proporciona Kubernetes fue efectiva ayudándome a completar las tareas.

Rango CSUQ	*Encuestados*	*Total*
1	0	0
2	0	0
3	1	3
4	8	32
5	12	60
6	8	48
7	0	0
	Promedio	4,931034483

Cuadro 4.12: Resultados de la undécima pregunta CSUQ
Fuente: Elaboración Propia

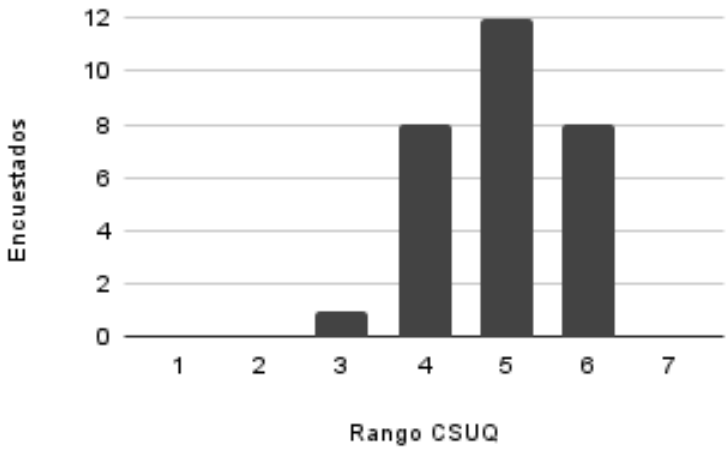


Figura 4.12: Resultados de la undécima pregunta CSUQ.
Fuente: Elaboración Propia

12.- La organización de la información de Kubernetes en la terminal o por pantalla fue clara.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	1	3
4	5	20
5	9	45
6	14	84
7	0	0
	Promedio	5,24137931

Cuadro 4.13: Resultados de la duodécima pregunta CSUQ
Fuente: Elaboración Propia

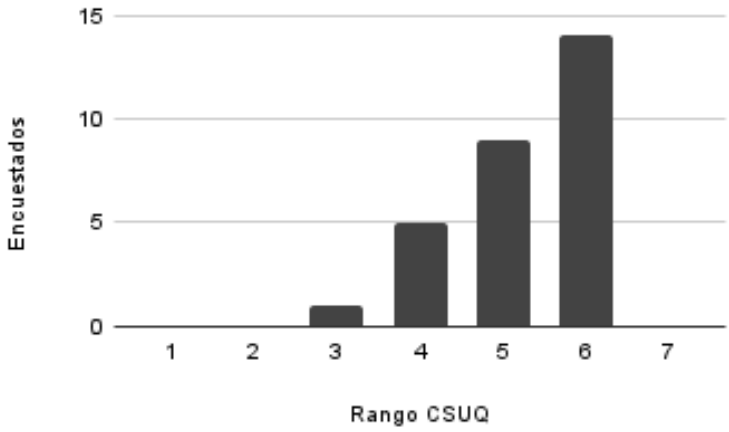


Figura 4.13: Resultados de la duodécima pregunta CSUQ.
Fuente: Elaboración Propia

13.- La interfaz de Kubernetes fue placentera.

Rango CSUQ	*Encuestados*	*Total*
1	0	0
2	0	0
3	2	6
4	5	20
5	12	60
6	9	54
7	1	7
	Promedio	5,068965517

Cuadro 4.14: Resultados de la decimotercera pregunta CSUQ
Fuente: Elaboración Propia

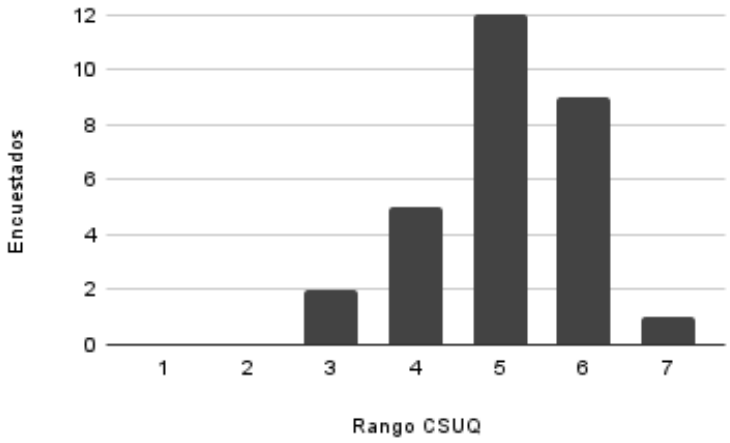


Figura 4.14: Resultados de la decimotercera pregunta CSUQ.
Fuente: Elaboración Propia

14.- Me gustó utilizar Kubernetes.

Rango CSUQ	*Encuestados*	*Total*
1	0	0
2	0	0
3	3	9
4	4	16
5	6	30
6	14	84
7	2	14
	Promedio	5,275862069

Cuadro 4.15: Resultados de la decimocuarta pregunta CSUQ
Fuente: Elaboración Propia

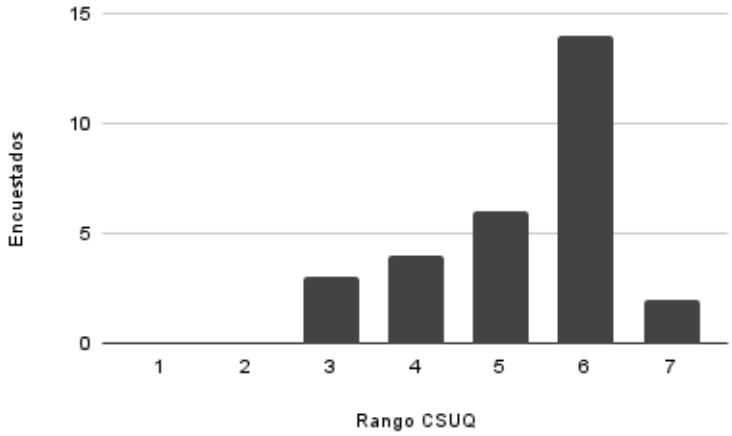


Figura 4.15: Resultados de la decimocuarta pregunta CSUQ.
Fuente: Elaboración Propia

15.- Kubernetes tuvo todas las herramientas que esperaba que tuviera.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	3	9
4	6	24
5	9	45
6	10	60
7	1	7
	Promedio	5

Cuadro 4.16: Resultados de la decimoquinta pregunta CSUQ
Fuente: Elaboración Propia

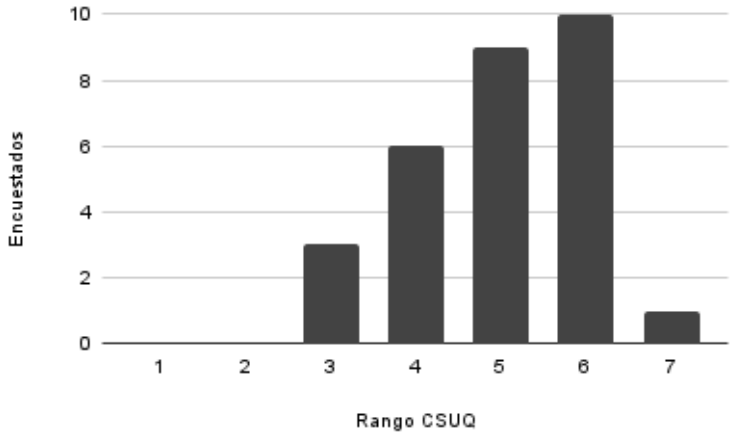


Figura 4.16: Resultados de la decimoquinta pregunta CSUQ.
Fuente: Elaboración Propia

16.- En general, estuve satisfecho con la herramienta.

Rango CSUQ	***Encuestados***	***Total***
1	0	0
2	0	0
3	3	9
4	5	20
5	6	30
6	13	78
7	2	14
	Promedio	5,206896552

Cuadro 4.17: Resultados de la decimosexta pregunta CSUQ
Fuente: Elaboración Propia

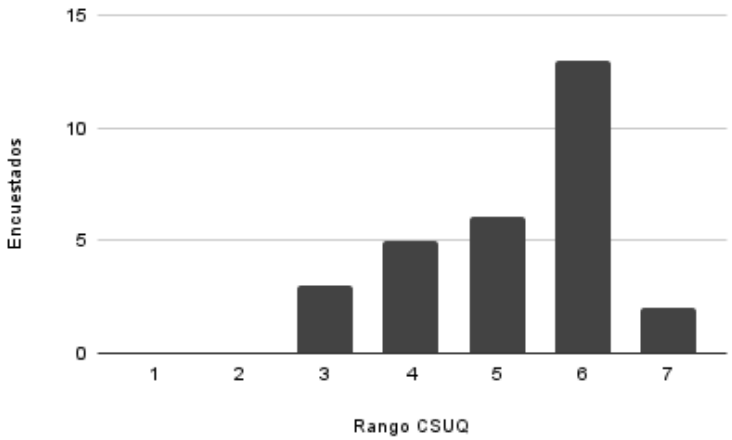


Figura 4.17: Resultados de la decimosexta pregunta CSUQ.
Fuente: Elaboración Propia

Promedio General en base a los promedios de cada pregunta

Al obtener los promedios individuales por pregunta, se requiere determinar un promedio general en base a aquellos valores en relación al número de preguntas, el cual a continuación se denominará como *Promedio General*.

Número de pregunta	***Promedio Individual***
1	4,965517241
2	4,827586207
3	4,75862069
4	4,551724138
5	4,75862069
6	3,75862069
7	4,75862069
8	4,620689655
9	5
10	4,931034483
11	4,931034483
12	5,24137931
13	5,068965517
14	5,275862069
15	5
16	5,206896552
Promedio General	**4,853448276**

Cuadro 4.18: Promedio en base a los promedios individuales por pregunta

4.2. Análisis de Impacto

El presente trabajo centra su enfoque de estudio en la demostración de la eficiencia del empleo de Kubernetes para la administración de servicios, mediante el manejo de contenedores, con la finalidad de preservar el comportamiento de las aplicaciones y su efectividad tanto en un entorno local como en producción.

Para la elaboración del análisis se determinó tres tipos de impacto, los cuales son:

- Impacto Tecnológico
- Impacto Ambiental
- Impacto Económico

4.2.1. Impacto Tecnológico

El uso de Kubernetes optimiza en gran medida las actividades fundamentales para mantener los servicios deseados en ejecución, prueba de ello es:

- Su capacidad para destruir elementos con fallos y reconstruirlos automáticamente.
- Su capacidad de migrar recursos mediante el empleo de ficheros Yaml.
- El respaldo de información que proporciona con el uso de volúmenes.

4.2.2. Impacto Ambiental

Kubernetes de manera esencial, se encuentra enfocado en mantener cantidades exorbitantes de contenedores en producción. Centrado satisfactoriamente en emplear recursos limitados de infraestructura.

4.2.3. Impacto Económico

Al emplear contenedores, Kubernetes, asegura un uso óptimo de los recursos, por tanto, los costos de operación se ven reducidos, a la vez que balancea la carga entre sus diversos nodos, permitiendo de tal modo una mayor disponibilidad de los servicios y una reducción significativa de infraestructura.

4.3. Implementación de buenas practicas

4.3.1. Costos

Es recomendable emplear "kubernetes instance calculator" para optimizar en gastos y deducir un valor próximo al costo real de mantener un clúster de Kubernetes en producción con proveedores cloud.

4.3.2. Numero de Pods en nodos

Es eficiente mantener un límite de 110 Pods por nodo en un clúster estándar. De tal modo que no exista fallos en las operaciones del mismo.

4.3.3. Arquitectura

Revisar la arquitectura de la imagen que se genera empleando una herramienta de creación de contenedores como lo es Docker, es de prioridad importancia, ya que tales elementos, al compartir recursos directamente con el kernel de la máquina huésped, hereda sus características, y puede ocasionar fallos al momento de desplegar la solución y emplearla en Kubernetes.

CONCLUSIONES, RECOMENDACIONES

Conclusiones

Utilizar Kubernetes para la administración de servicios, requiere de tiempo, debido a la extensa curva de aprendizaje que demanda esta herramienta, sin embargo, elaborar ejemplos con un bajo nivel de complejidad, otorga una comprensión general de la plataforma.

Crear una imagen de WildFly para su despliegue y gestión desde Kubernetes integrando una base de datos PostgreSQL creada en la plataforma ElephantSQL, permitió definir un entorno amigable para el uso y demostración de la eficiencia de la plataforma.

Usar Kubernetes cuando las cargas de trabajo son mínimas, no es lo adecuado, ya que esta herramienta genera beneficios para cargas de trabajo complejas, como lo sería el mantener un gran número de microservicios en constante ejecución.

Para emplear Kubernetes desde el rol de administrador, es indispensable conocer el funcionamiento de una herramienta para la creación de imágenes, en cuyo caso Docker es una excelente opción y entender los conceptos básicos del sistema operativo Linux.

Integrar a minikube con KVM no es posible, si la tecnología de virtualización no se encuentra habilitada.

Usar a AWS como proveedor cloud no fue posible, debido a que requiere de una tarjeta de crédito a la cual como estudiante no se puede acceder.

Emplear el cuestionario CSUQ determinó que, en lo que respecta al promedio individual por pregunta, la pregunta número 14, *"Me gustó utilizar Kubernetes"*, tuvo un mayor promedio al de las demás.

Recomendaciones

Recomendaciones Generales

1. Emplear latex para la gestión documental de proyectos y trabajos de investigación.
2. Emplear "play with k8s" para aprender a usar kubernetes sin la necesidad de instalarlo.
3. Estudiar el uso de vim para el desarrollo profesional.
4. Estudiar los beneficios de emplear vagrant para la virtualización.

Recomendaciones Específicas

1. Estudiar el uso de github actions para la integración continua (CI/CD).
2. Emplear helm para la administración de paquetes de Kubernetes, con la finalidad de desplegar los servicios deseados automáticamente en el clúster.
3. Usar RBAC para el manejo de permisos en Kubernetes, debido a la necesidad de implantar métricas de seguridad en el clúster.
4. Emplear MetalLB para la creación de LoadBalancers en un cluster de Kubernetes en entornos bare metal.

Bibliografía

Barajas-Bustillos, M., Reyes, R., Riva, J. & Maldonado, A. (2017). Estudio comparativo de cuestionarios para la evaluación de la usabilidad en software. *Ingeniantes*, *1*, 2.

Buchanan, S., Rangama, J. & Bellavance, N. (2019). En *Introducing azure kubernetes service* (pp. 35-50). Apress, Berkeley, CA. *https:// doi.org/ 10.1007/ 978-1-4842-5519-3_3*

García, C., Oddone, N. & de Oca, M. S. M. (2018). Mercosur en la agenda global del desarrollo: El peas y su vigencia en el marco de los ods 2030. *Revista MERCOSUR de Políticas Sociales*, *2*, 5-33. *https:// doi.org/ 10.28917/ ism.2018-v2-5*

Hedlefs, M., Gonzalez, A. d. l., Sánchez Miranda, M. & Villegas, A. (2016). Adaptación al español del cuestionario de usabilidad de sistemas informáticos csuq / spanish language adaptation of the computer systems usability questionnaire csuq. *RECI Revista Iberoamericana de las Ciencias Computacionales e Informática*, *4*, 84. *https:// doi.org/ 10.23913/ reci.v4i8.35*

Ibujes Villacís, J. M. & Franco Crespo, A. A. (2019). Uso de las TIC y relación con los Objetivos de Desarrollo Sostenible en Ecuador. *RETOS. Revista de Ciencias de la Administración y Economía*, *9*, 37-53. *http:// scielo.senescyt.gob.ec/ scielo.php? script=sci_arttext&pid=S1390-86182019000100037&nrm=iso*

Ipiales Gubio, R. M. (2021). *Desarrollo del módulo de gestión del plan operativo anual en el sistema integrado de gestión de la empresa eléctrica regional norte aplicando scrum como marco de trabajo* (B.S. thesis).

k3s.io. (s.f.). *Lightweight Kubernetes*. Cloud Native Computing Foundation. Consultado el 26 de enero de 2022, desde *https:// k3s.io/*

kubernetes.io. (s.f.-a). *kube-scheduler*. Cloud Native Computing Foundation. Consultado el 20 de diciembre de 2021, desde *https:// kubernetes.io/ docs/ reference/ command-line-tools-reference/ kube-scheduler/*

kubernetes.io. (s.f.-b). *Kubernetes Components*. Cloud Native Computing Foundation. Consultado el 23 de diciembre de 2021, desde *https:// kubernetes.io/ docs/ concepts/ overview/ components/*

kubernetes.io. (s.f.-c). *minikube start*. Cloud Native Computing Foundation. Consultado el 26 de enero de 2022, desde *https:// minikube.sigs.k8s.io/ docs/ start/*

kubernetes.io. (s.f.-d). *Persistent Volumes*. Cloud Native Computing Foundation. Consultado el 26 de enero de 2021, desde *https:// kubernetes.io/ docs/ concepts/ storage/ persistent-volumes/*

Larsson, L., Tärneberg, W., Klein, C., Elmroth, E. & Kihl, M. (2020). Impact of etcd deployment on kubernetes, istio, and application performance. *Software: Practice and Experience*, *50*(10), 1986-2007. *https:// doi.org/ 10.1002/ spe.2885*

Martin, P. (2020). En *Kubernetes* (pp. 11-13). Apress, Berkeley, CA. *https://doi.org/10.1007/978-1-4842-6494-2_2*

Mejía Rodríguez, N. M. & Barbecho Chimbo, J. P. (2021). *Despliegue de una nube híbrida para entornos de desarrollo implementando kubernetes y dockers en la plataforma openstack e infraestructura en amazon* (B.S. thesis).

Mendonça, N. C., Box, C., Manolache, C. & Ryan, L. (2021). The monolith strikes back: Why istio migrated from microservices to a monolithic architecture. *IEEE Software*, *38*(5), 17-22. *https://doi.org/10.1109/MS.2021.3080335*

Nguyen, N. & Kim, T. (2020). Toward highly scalable load balancing in kubernetes clusters. *IEEE Communications Magazine*, *58*(7), 78-83. *https://doi.org/10.1109/MCOM.001.1900660*

QEMU. (s.f.-a). *About QEMU*. Software Freedom Conservancy. Consultado el 11 de diciembre de 2021, desde *https://www.qemu.org/docs/master/about/index.html*

QEMU. (s.f.-b). *About QEMU*. Software Freedom Conservancy. Consultado el 26 de enero de 2022, desde *https://www.qemu.org/docs/master/about/build-platforms.html*

Red Hat. (2018). *¿QUÉ ES UN CONTENEDOR DE LINUX?* Red Hat Inc. Consultado el 11 de diciembre de 2021, desde *https://www.redhat.com/es/topics/containers/whats-a-linux-container*

Red Hat. (s.f.-a). *¿QUÉ ES DOCKER?* Red Hat Inc. Consultado el 11 de diciembre de 2021, desde *https://www.redhat.com/es/topics/containers/what-is-docker*

Red Hat. (s.f.-b). *¿QUÉ ES ETCD?* Red Hat Inc. Consultado el 11 de diciembre de 2021, desde *https://www.redhat.com/es/topics/containers/what-is-etcd*

Red Hat. (s.f.-c). *¿QUÉ ES KUBERNETES?* Red Hat Inc. Consultado el 11 de diciembre de 2021, desde *https://www.redhat.com/es/topics/containers/what-is-kubernetes*

Red Hat. (s.f.-d). *¿QUÉ ES KVM?* Red Hat Inc. Consultado el 11 de diciembre de 2021, desde *https://www.redhat.com/es/topics/virtualization/what-is-KVM*

Vizcaino Quiroz, Y. D. (2021). *Optimización de aplicaciones java enterprise monolíticas mediante el uso de contenedores docker* (B.S. thesis).

Wojciechowski, Ł., Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T. & Hong, M. (2021). Netmarks: Network metrics-aware kubernetes scheduler powered by service mesh, En *Ieee infocom 2021 - ieee conference on computer communications. https://doi.org/10.1109/INFOCOM42981.2021.9488670*

ANEXO 1

Instalación de nmap

1.1. Instalación de nmap en Ubuntu

Para instalar nmap ejecutamos el comando:

```
sudo apt install nmap -y
```

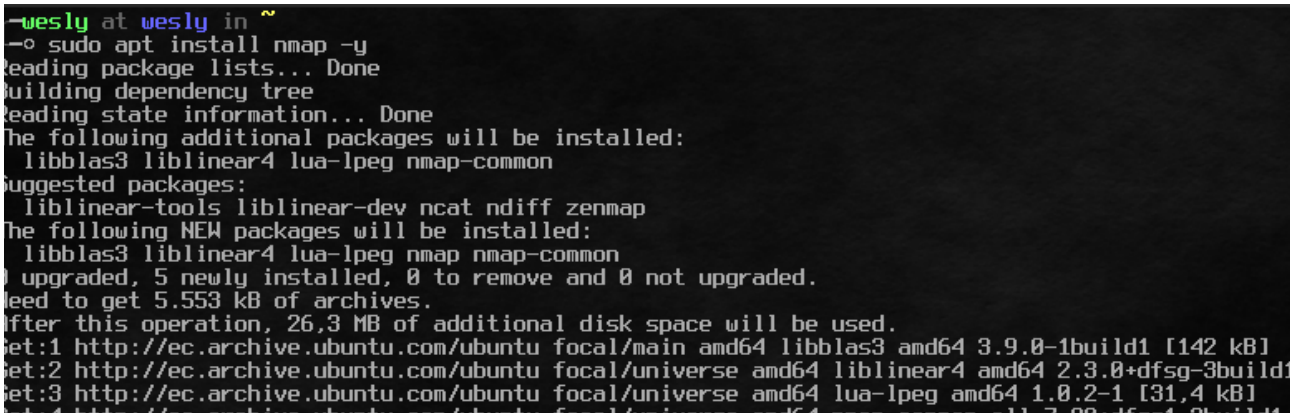
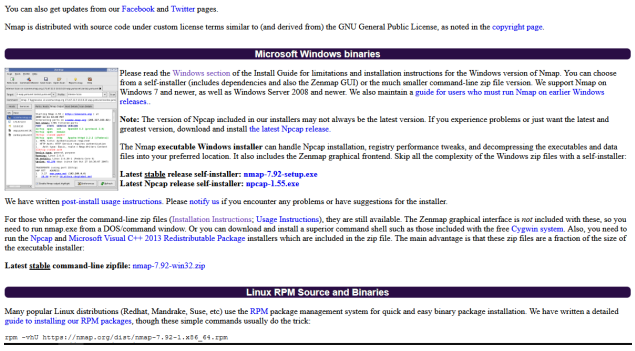


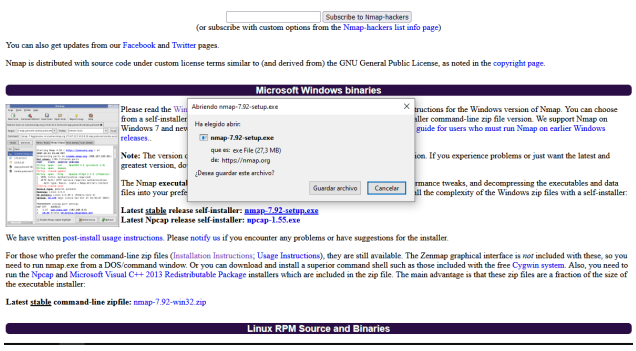
Figura 1.1: Instalación de nmap en Ubuntu

1.2. Instalación de nmap en Windows

Descargar el archivo ejecutable directamente de la pagina oficial.

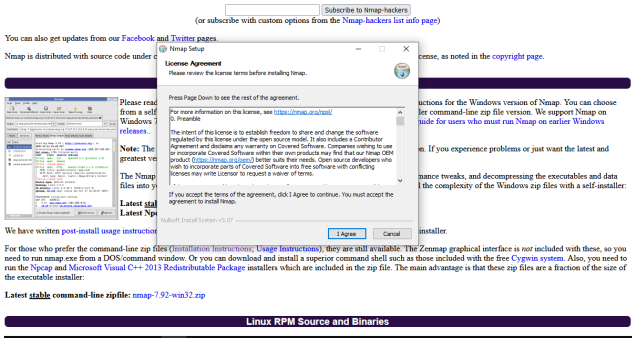


(a) Página oficial de nmap.

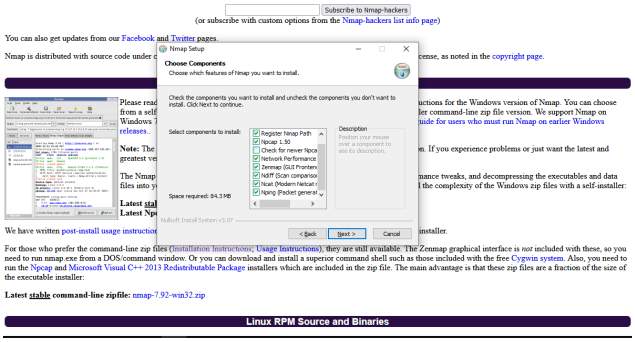


(b) Apertura del archivo ejecutable.

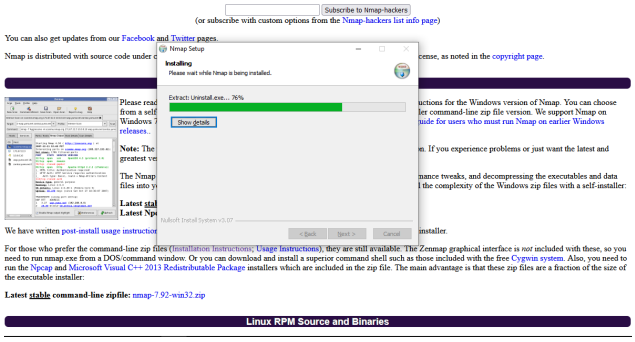
Figura 1.2: Descarga de nmap en Windows.



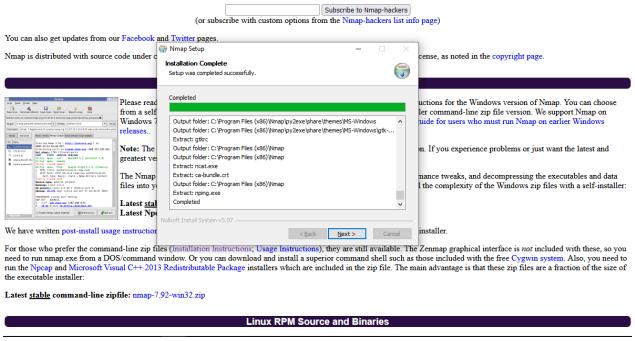
(a) Aceptacion de terminos y condiciones.



(b) Elección de los componentes.



(c) Inicio de la instalación de nmap en Windows.



(d) Instalación finalizada.

Figura 1.3: Incorporación de nmap a Windows.

ANEXO 2

Network

2.1. Obtención de la ip de la maquina

Con el comando:

```
ip -brief -color -4 a
```

Se obtiene como resultado la ip de la maquina, la cual en el presente caso es la "192.168.1.5",y la palabra "wlp2s0" informa que la maquina con dicha dirección se encuentra enlazada a internet mediante la red de área local inalámbrica. En el comando la opción -4 especifica que unicamente se den a conocer las direcciones ipv4.

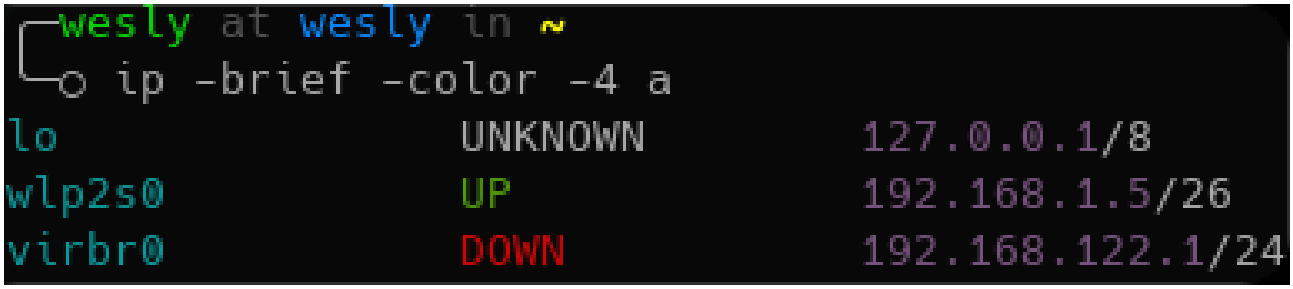


Figura 2.1: ip de la maquina

2.2. Obtención de la ip del router

Con el comando:

```
ip -brief -color -4 r
```

Se obtiene como resultado la ip del router.

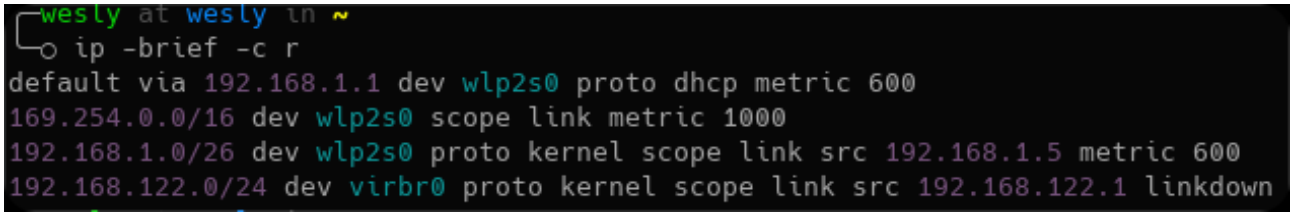


Figura 2.2: ip del router

Con ello los datos obtenidos fueron:

Ip del router= *192.168.1.1*

Notacion CIDR= /26

La notación CIDR se obtiene del / hallado en el primer valor que se encuentra en la misma fila que la ip de la máquina.

2.3. Obtención de la mascara de subred

Para ello se debe saber que la máscara de subred está compuesta por cuatro octetos los cuales individualmente sumados deben dar un total de 255.

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

Cuadro 2.1: Un octeto de los 4 disponibles.
Fuente: Elaboración Propia

Mascara de subred a Binaria:

Primer Octeto								Segundo Octeto							
255								255							
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Cuadro 2.2: Primer y Segundo Octeto
Fuente: Elaboración Propia

Primer Octeto								Segundo Octeto							
255								192							
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0

Cuadro 2.3: Tercer y Cuarto Octeto
Fuente: Elaboración Propia

Resultados:

La subred *255.255.255.192* es igual a la notación CIDR */26*.

La notación CIDR es el número de 1s en la conversión de la máscara de subred a binario, el cual es= /26

ANEXO 3

Instalación de shell in a box

3.1. Instalación de shell in a box en Ubuntu

```
sudo apt install openssl shellinabox
```

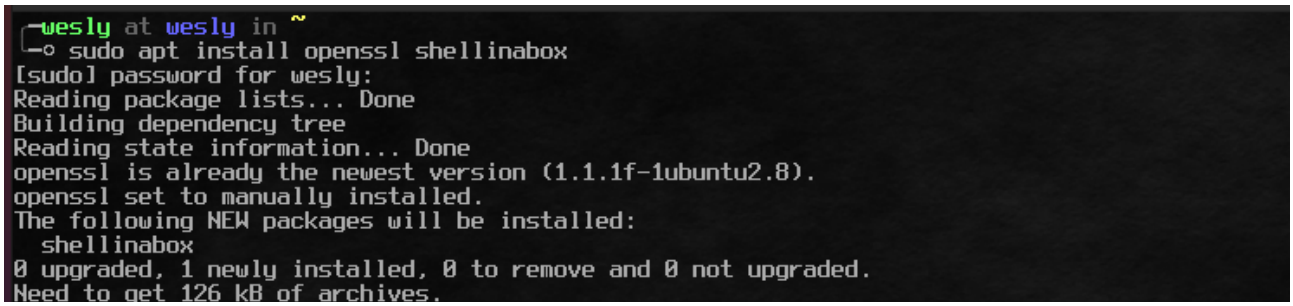


Figura 3.1: Comando para instalar Shellinabox

Se procede a acceder a el directorio “sudo vim /etc/default/shellinabox”, en el cual se pueden realizar cambios, por ejemplo, el puerto en el cual funcionará el servicio, en este caso lo dejaremos por defecto, en este caso se emplea el editor de texto “vim”, pero tranquilamente se puede emplear con “nano”, o el de su preferencia.

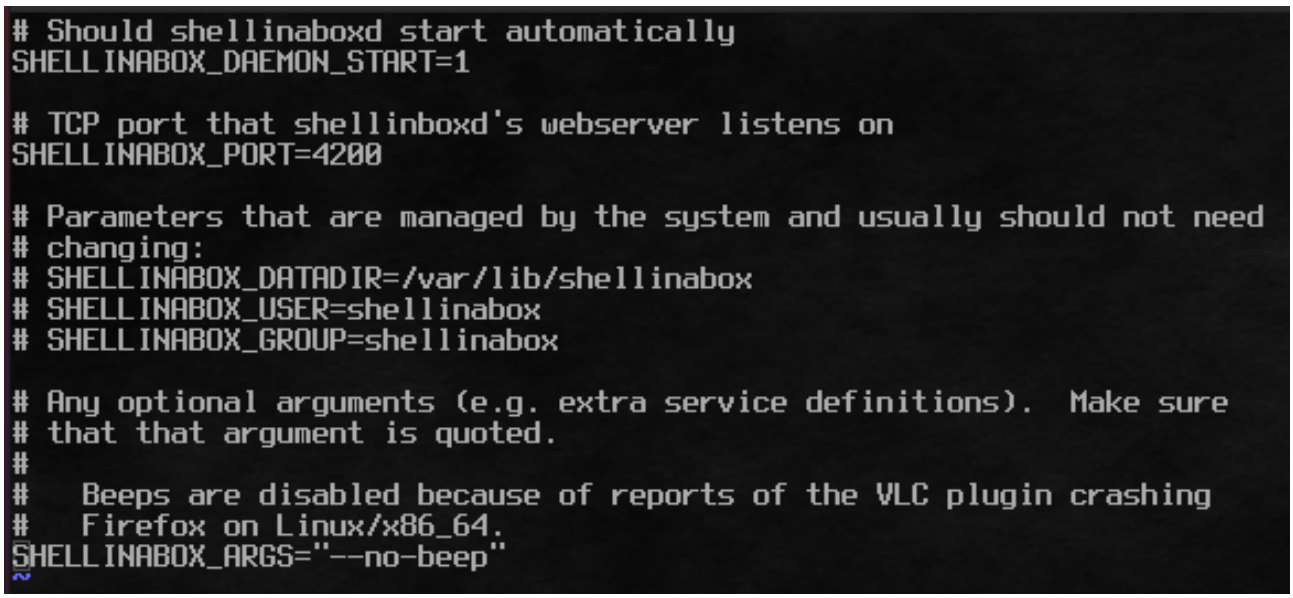


Figura 3.2: Archivo /etc/default/shellinabox

Para iniciar el servicio primero se observa el estado en el cual se encuentra, para ello se debe digitar el comando:

```
sudo systemctl status shellinabox.service
```

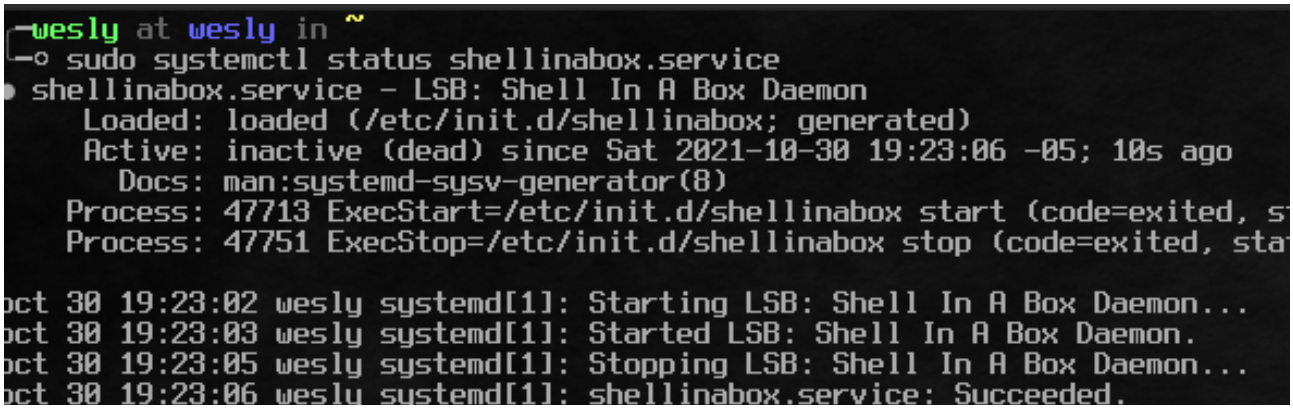


Figura 3.3: Estado del servicio de shellinabox

Si el servicio se encuentra inactivo, lo iniciamos con el comando:

```
sudo systemctl start shellinabox.service
```

Para posteriormente comprobar su estado nuevamente.

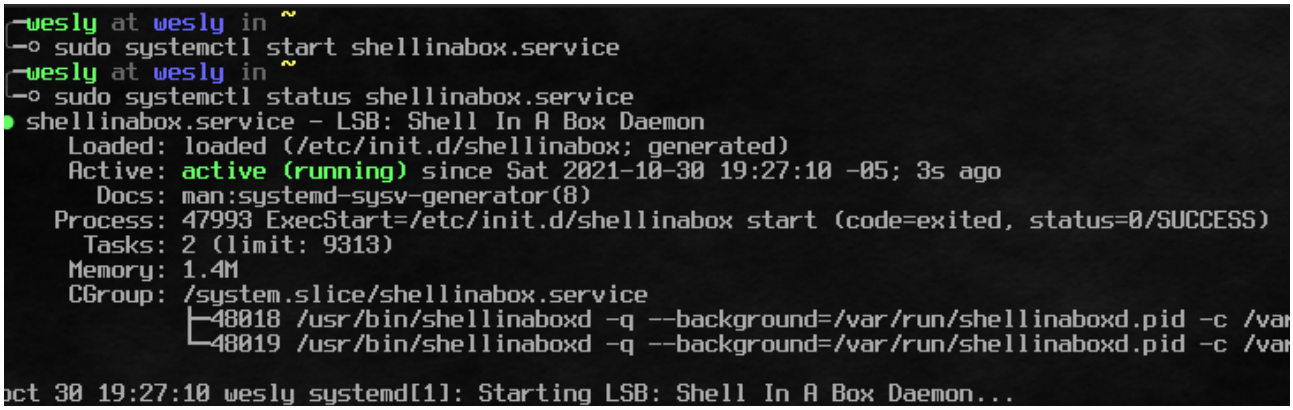


Figura 3.4: Activación del servicio de shellinabox

En caso de que el servicio se encuentre iniciado procedemos a reiniciarlo con el comando:

```
sudo systemctl restart shellinabox.service
```

Y se comprobará el estado del servicio nuevamente con el comando:

```
sudo systemctl status shellinabox.service
```

Por culminar, se deben escanear los puertos abiertos en la maquina en base a la dirección ip de la misma, por defecto shellinabox se ejecuta en el puerto 4200, sin embargo, si en el archivo de configuración se elaboraron cambios este debería ser distinto, para ello se emplea nmap:

```
nmap ip-de-la-maquina -p-
```

En este caso el comando se ejecutaría de la siguiente manera:

```
nmap 192.168.1.5 -p-
```

```
wesly at wesly in ~
 nmap 192.168.1.5 -p-
Starting Nmap 7.80 ( https://nmap.org ) at 2021-11-23 22:03 -05
Nmap scan report for 192.168.1.5 (192.168.1.5)
Host is up (0.00011s latency).
Not shown: 65528 closed ports
PORT      STATE SERVICE
111/tcp   open  rpcbind
2049/tcp  open  nfs
4200/tcp  open  vrml-multi-use
32873/tcp open  unknown
45553/tcp open  unknown
48617/tcp open  unknown
57163/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 1.92 seconds
```

Figura 3.5: Obtención de puertos

Para acceder al servicio se debe colocar en el navegador la dirección ip de la maquina y el puerto establecido para el acceso, mediante el protocolo **https**:

https://⟨ip_de_la_maquina⟩:⟨puerto_establecido⟩ En el pertinente caso es: ***https://192.168.1.5:4200***

Al ingresar al enlace mencionado, el navegador informa que existe un error de privacidad.

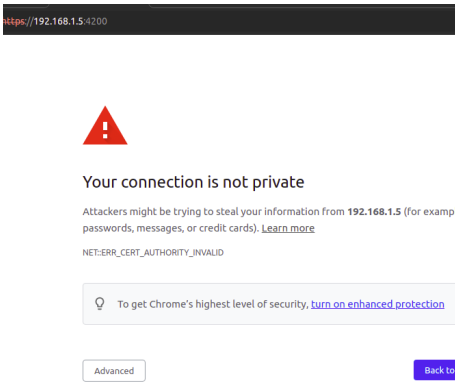


Figura 3.6: Mensaje de error de privacidad en el navegador

El error es debido a que el servicio no cuenta con un certificado válido, sin embargo, por motivos de demostración, se debe digitar en *"Advanced"* para luego seleccionar, *"Proceed to 192.168.1.5 (unsafe)"*.

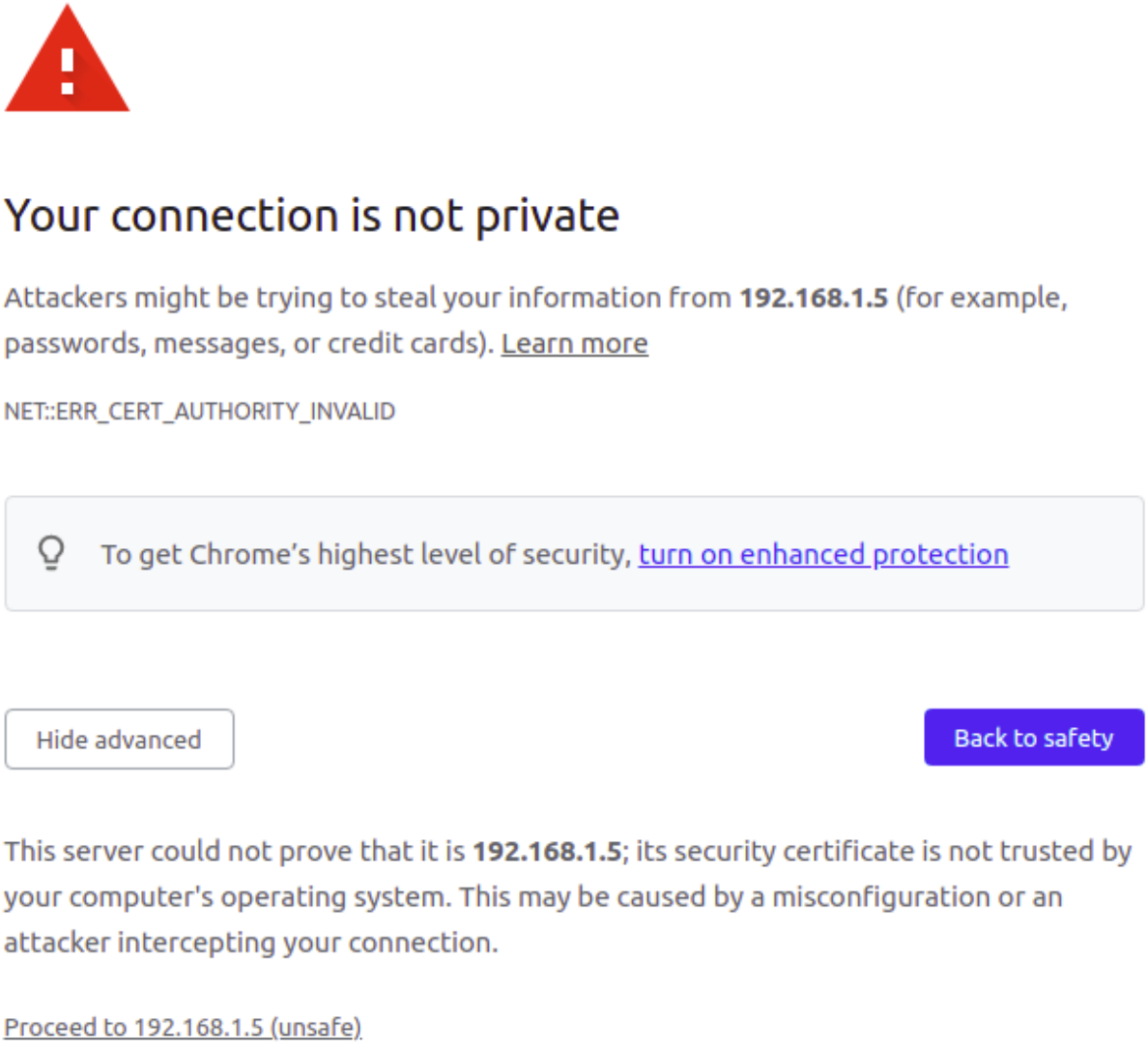


Figura 3.7: avanzar al servicio

Finalmente se debe acceder al servicio, colocando el usuario, y la contraseña del servidor linux.

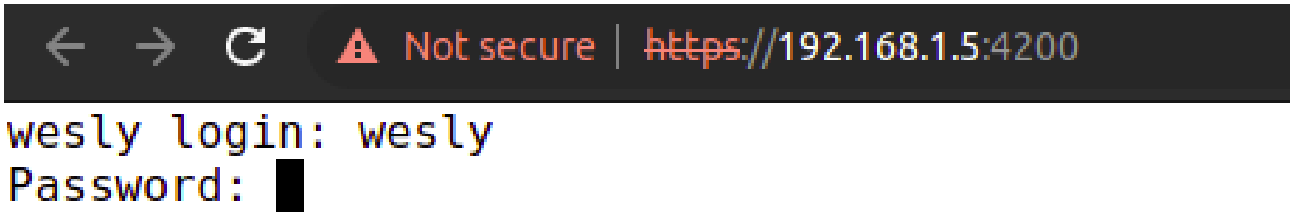


Figura 3.8: Logueo requerido

Y de forma oportuna shellinabox habilita una shell interactiva en el navegador.

Figura 3.9: Acceso verificado a shellinabox

Shell In A Box o comunmente llamado "shellinabox", aporta soluciones rentables, y estables tanto al agilizar el tiempo de gestión de los servicios, ya que únicamente se requiere de un navegador para acceder a la terminal, como en la presentación de ciertos desarrollos a otros usuarios ya que algunas plataformas de video conferencias únicamente permiten presentar el navegador, sea por fallos en el sistema operativo, por fallos en la aplicación de escritorio, o sencillamente porque de esa manera se encuentra diseñada.

ANEXO 4

Instalación de qemu-kvm

4.1. Habilitación de la tecnología de Virtualización Vt-x

Se debe acceder a la bios de la maquina, el método de ingreso depende en la mayoria de la marca del mismo. En equipos *"Dell"* al iniciar o reiniciar el equipo, presionando la tecla *"F12"*, es posible ingresar.

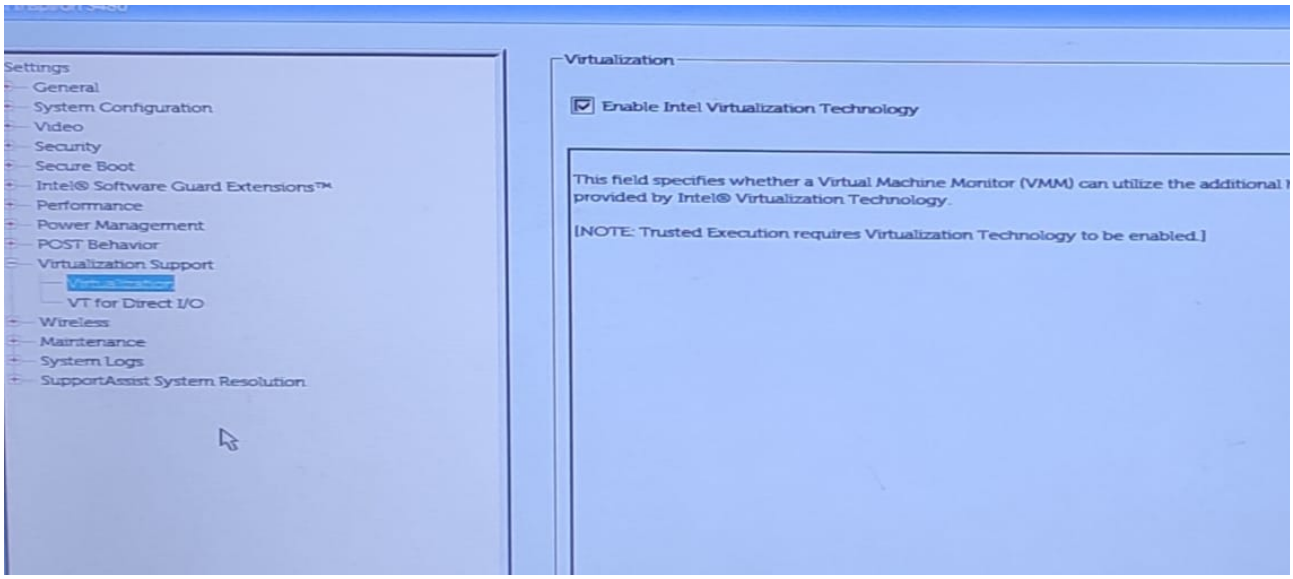


Figura 4.1: Bios de un equipo Dell

4.1.1. Verificación del funcionamiento de la tecnología de Virtualización desde un servidor linux

Se debe iniciar el servidor de linux, y digitar el comando:

```
cat /proc/cpuinfo | egrep -c "(vmx | svm)"
```

El cual comprueba la disponibilidad de la tecnología de virtualización. La palabra “vmx” es un referente hallado en procesadores de tipo intel, y “svm” es un referente hallado en procesadores de tipo amd.

Figura 4.2: Disponibilidad de la tecnología de virtualización

4.2. Instalación en Ubuntu

Se procede a instalar los paquetes qemu-kvm y virt-manager, mediante el comando:

```
sudo apt install qemu-kvm virt-manager -y
```

Dichos paquetes instalan y habilitan el servicio de kvm.

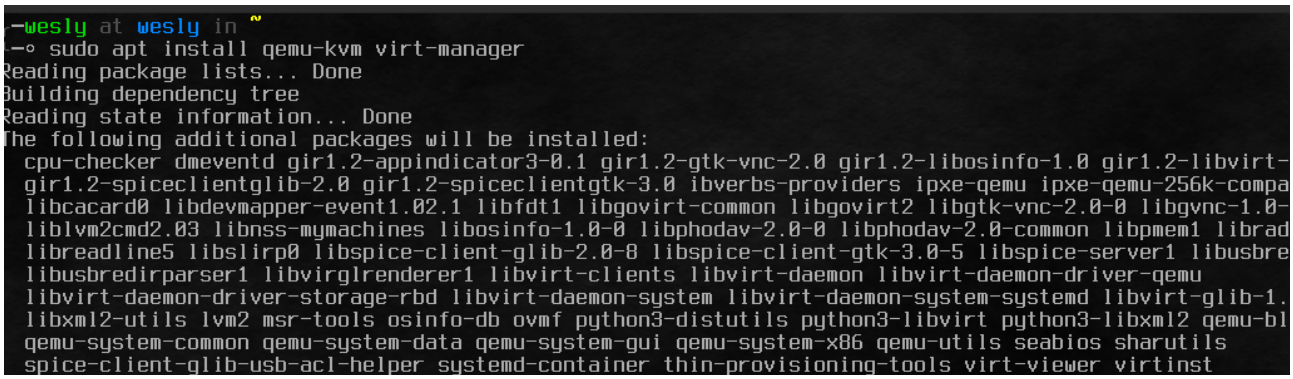


Figura 4.3: Instalación de paquetes qemu-kvm virt-manager

Para observar los módulos de kvm instalados, se los puede encontrar directamente con el siguiente comando:

```
cat /proc/modules | grep kvm
```

O de igual manera se los puede revisar con el comando "lsmod", el cual, permite observar el estado de los módulos en el kernel de Linux, a la vez, se filtra los datos con el comando *grep* de la siguiente manera:

```
lsmod | grep "kvm"
```

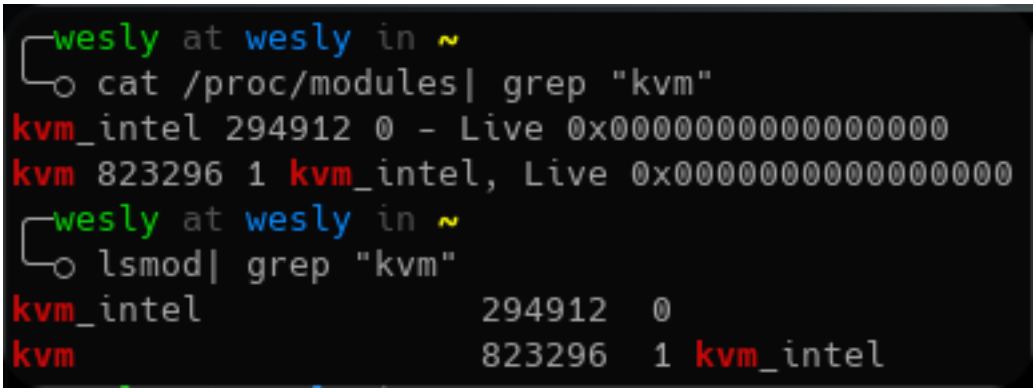


Figura 4.4: Obtención de los modulos de kvm

En caso de no obtener un resultado, se recomienda instalar los módulos libvirt-daemon-system, libvirt-clients, virtinst. Sin embargo, estos módulos por defecto se deberían encontrar instalados con el comando:

```
sudo apt install qemu-kvm virt-manager
```

El cual anteriormente fue mencionado. De no ser el caso, se deberá digitar la siguiente instrucción:

```
sudo apt install libvirt-daemon-system libvirt-clients virtinst
```

Otro medio para comprobar el correcto funcionamiento de kvm es revisar el estado del servicio **libvirtd** con el comando:

```
sudo systemctl status libvirtd.service
```

Figura 4.5: Estado del servicio de libvirtd

De encontrarse el respectivo servicio inactivo, se lo debería encender con el comando:

```
sudo systemctl start libvirtd.service
```

Para adecuadamente comprobar una vez más el estado del servicio con el comando:

```
sudo systemctl status libvirtd.service
```

Si el servicio sigue sin funcionar es probable que los módulos de kvm no se encuentren cargados, para ello, digitaremos el siguiente comando:

```
sudo modprobe kvm  kvm_intel
```

Posteriormente se observará en el listado de los módulos si aquellos correspondientes a kvm se encuentran cargados, para ello se deberá digitar el siguiente comando:

```
lsmod| grep "kvm"
```

Si el procesador del equipo en el cual se esta laborando es **AMD**, es necesario cambiar el módulo kvm_intel con el módulo kvm_amd:

```
sudo modprobe kvm  kvm_amd
```

ANEXO 5

Usuarios en linux

5.1. Incorporación de un usuario a los grupos correspondientes a kvm

Colocando por ejemplificación al usuario actual del sistema, el cual no es **root** se deberá seguir los procesos a continuación mencionados.

En el listado de los grupos debe existir uno llamado "libvirt", para encontrarlo, se deberá mostrar los grupos existentes en el sistema y filtrar por la palabra "libvirt". Tal acción se puede hacer de diversas formas, unas de ellas son empleando los comandos:

```
cat /etc/group | grep "libvirt"
```

```
getent group | grep libvirt
```

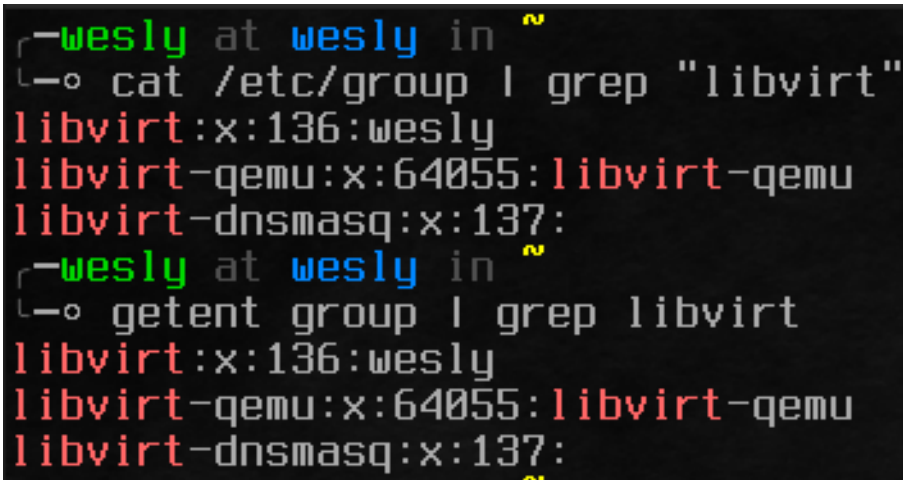


Figura 5.1: Obtención del grupo libvirt

Para obtener los grupos a los cuales pertenece un usuario en específico se emplea el comando:

```
groups $USER
```

O el comando:

```
getent group | grep $USER
```

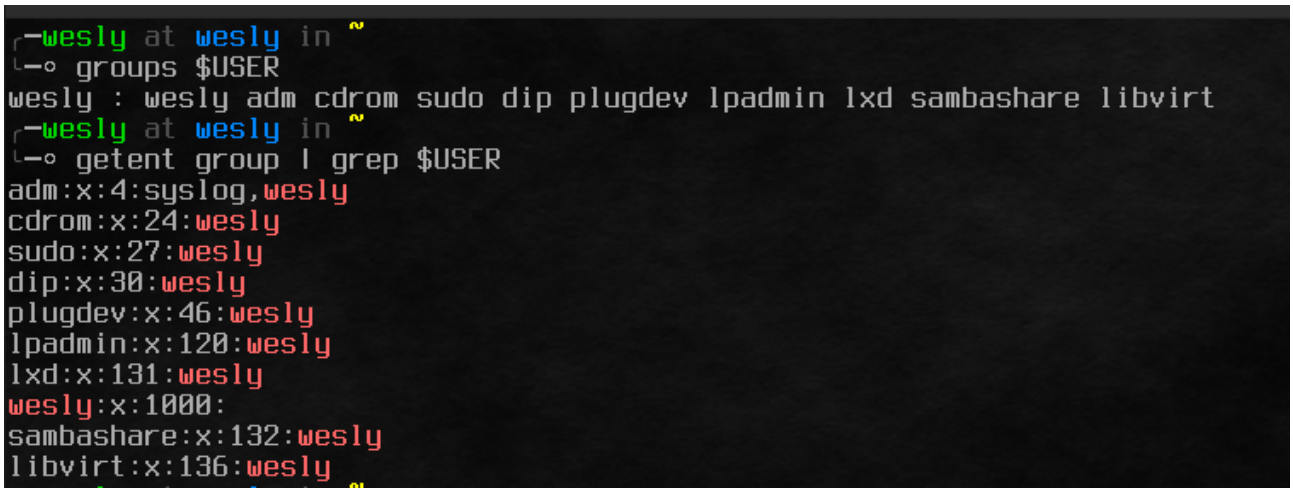


Figura 5.2: Grupos de un usuario

"$USER" hace referencia al nombre del usuario actual. Y como se observa el usuario actual se encuentra agregado al grupo "libvirt", de no estarlo, se lo puede incorporar mediante el comando:

```
sudo usermod -a $USER -G libvirt
```

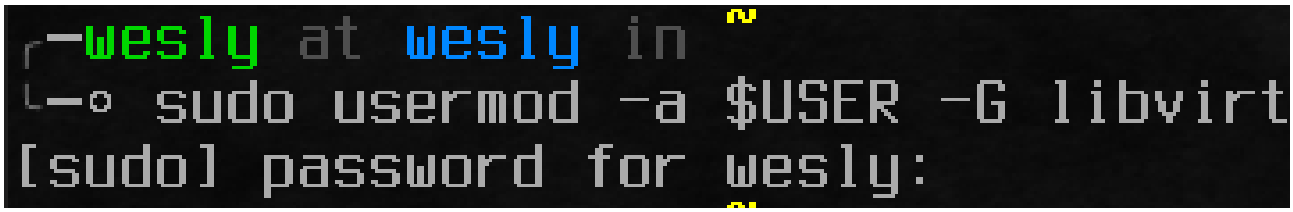


Figura 5.3: Primer método para agregar un usuario

O con el comando:

```
sudo adduser $USER libvirt
```

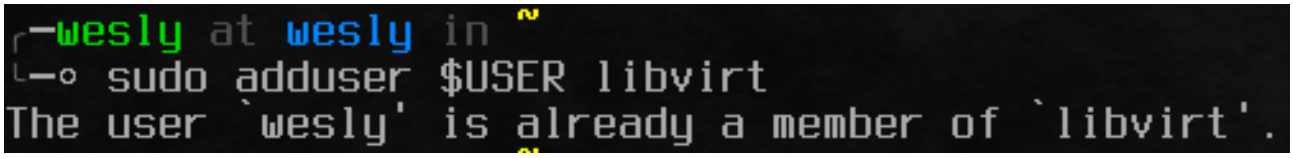


Figura 5.4: Segundo método para agregar un usuario

Por último se deberá reiniciar el computador, y con ello los cambios respectivos serán aplicados.

ANEXO 6

Virtualbox

6.1. Instalación de VirtualBox en Ubuntu

En aquellas distribuciones basadas en Debian, su proceso de instalación se lo puede realizar directamente desde la línea de comandos con la instrucción:

```
sudo apt install virtualbox
```

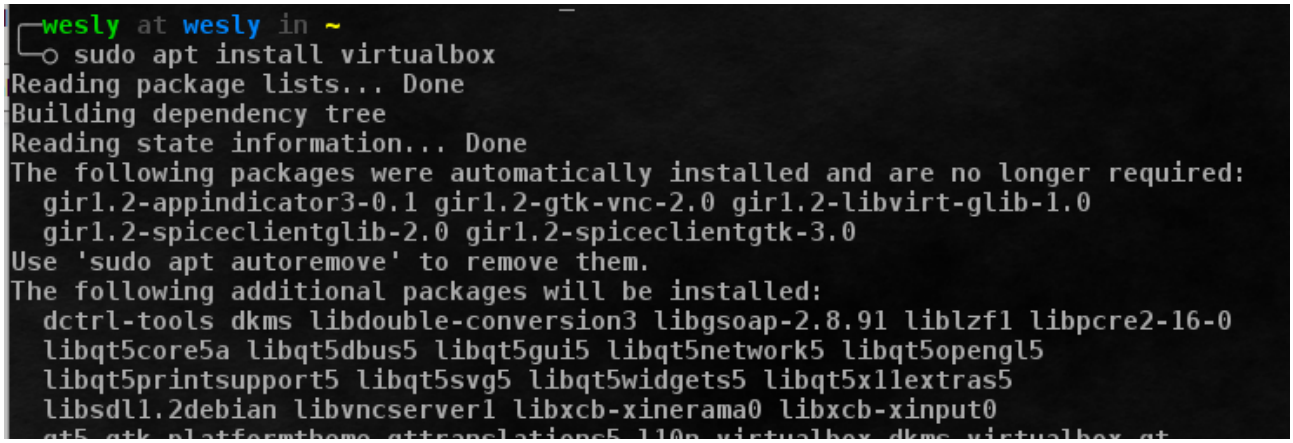


Figura 6.1: Instalación de virtualbox

ANEXO 7

Instalación de vagrant

7.1. Instalación de vagrant en Ubuntu

Se procede ha acceder al sitio web oficial de vagrant, a la sección de descargas, en ella se encontraran las indicaciones para su respectiva instalación, tanto en sistemas linux, windows y mac. En tal caso, al emplear el sistema operativo Linux Ubuntu, se deberá seguir el proceso de instalación para esta distribución. Cabe destacar que Vagrant de preferencia trabaja con virtualbox, de igual manera ofrece un gran soporte para Hyper-V y Docker, sin embargo, se adapta a hipervisores como VMware, kvm, parallels u otros.

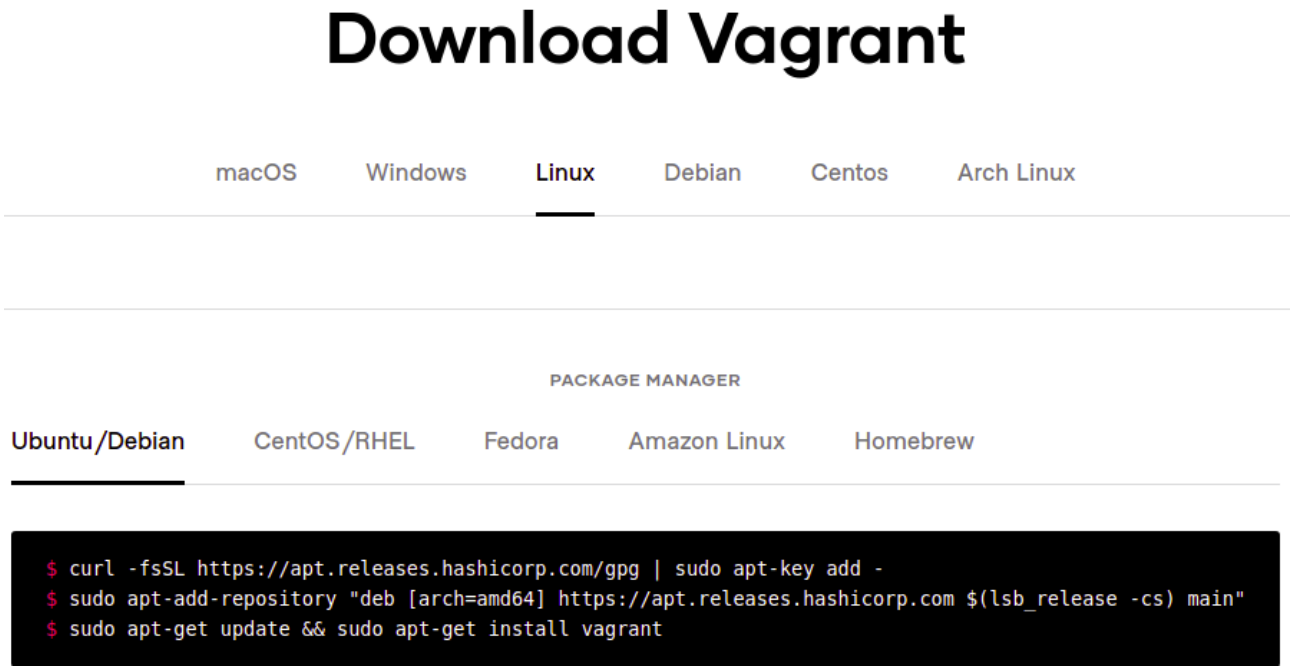


Figura 7.1: Página oficial de vagrant en la sección de instalación

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
```

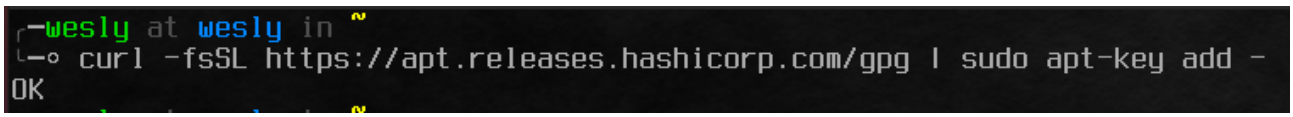


Figura 7.2: Instalación de vagrant primer paso

```
sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com \
$(lsb_release -cs) main"
```

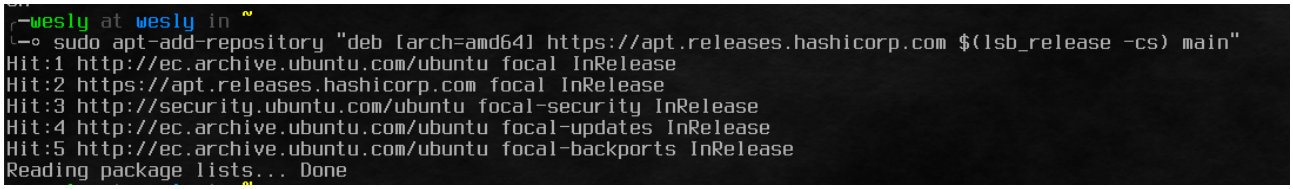


Figura 7.3: Instalación de vagrant segundo paso

```
sudo apt update && sudo apt install vagrant
```

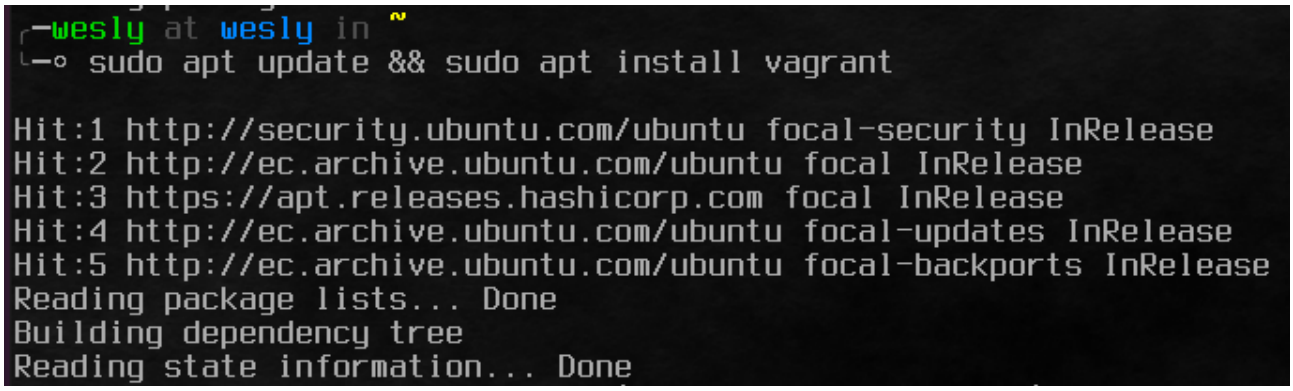


Figura 7.4: Instalación de vagrant tercer paso

7.2. Levantar una máquina virtual con vagrant

7.2.1. Creación de la máquina virtual

Con el comando:

```
vagrant init generic/centos8
```

Se crea un Vagrantfile con un box por defecto, caso contrario se lo puede hacer con el comando:

```
vagrant init
```

El cual otorga unicamente el archivo Vagrantfile.

7.2.2. Customización del Vagrantfile

Existe una variedad de elementos, los cuales se pueden personalizar de acuerdo a los requisitos que se solicite en Vagrant, sin embargo, es recomendable agregar:

- config.vm.network: Se especifica las reglas del network, incluyendo el forwardeo de puertos.
- config.vm.hostname: Nombre del hostname de la máquina a levantar.
- config.vm.provider: Nombre del virtualizador.
  - ➢ vb.name: Nombre de la máquina virtual.
  - ➢ vb.cpus: número de cpus a emplear.
  - ➢ vb.memory: Cantidad de memoria.

Ejemplo de Vagrantfile customizado

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "generic/centos8"
  # config.vm.network :forwarded_port, guest: 80, host: 8089
  config.vm.network :private_network, ip: "10.0.4.3"
  config.vm.hostname = "minikubelocal"
  config.vm.provider "virtualbox" do |vb|
    vb.name = "centos8minikube"
    vb.cpus=3
    vb.memory=3072
  end
  # Instalacion de ufw para el tratamiento del firewall
  # ufw habilita los puertos requeridos de k3s y el puerto 80 para nginx o apache
      para comprobar que las reglas se encuentren levantads.
  config.vm.provision "shell", inline: <<-SHELL
    echo "Cambiando de Centos8 a Centos8 Stream"
    dnf --disablerepo '*' --enablerepo=extras swap centos-linux-repos centos-
        stream-repos -y
    dnf distro-sync -y

    echo "Limpiar cache"
    yum clean all
    yum clean metadata
    yum clean packages

    echo "------ Instalacion de ufw para el firewall y nginx para exponer un
        servicio por defecto-------"

    yum -y install nginx

    echo "------------Iniciar y habilitar ufw, e iniciar nginx---------------"
    systemctl start nginx

    echo "----------------- Habilitar puertos de kubernetes -----------------"
    firewall-cmd --add-port=80/tcp
    firewall-cmd --add-port=8080/tcp --permanent

    echo "Control plane"
    firewall-cmd --add-port=6443/tcp --permanent
    firewall-cmd --add-port=2379/tcp --permanent
    firewall-cmd --add-port=2380/tcp --permanent
    firewall-cmd --add-port=10250/tcp --permanent
    firewall-cmd --add-port=10259/tcp --permanent
    firewall-cmd --add-port=10257/tcp --permanent

    echo "---------------Worker Nodes-----------------------"
    firewall-cmd --add-port=30000-32767/tcp --permanent
    echo "--------------Version de centos-------------------"
    cat /etc/os-release
  SHELL
end
```

7.2.3. Comandos básicos de vagrant

Algoritmo 7.1: Levanta una máquina virtual de acuerdo a las sentencias del Vagrantfile

```
$ vagrant up
```

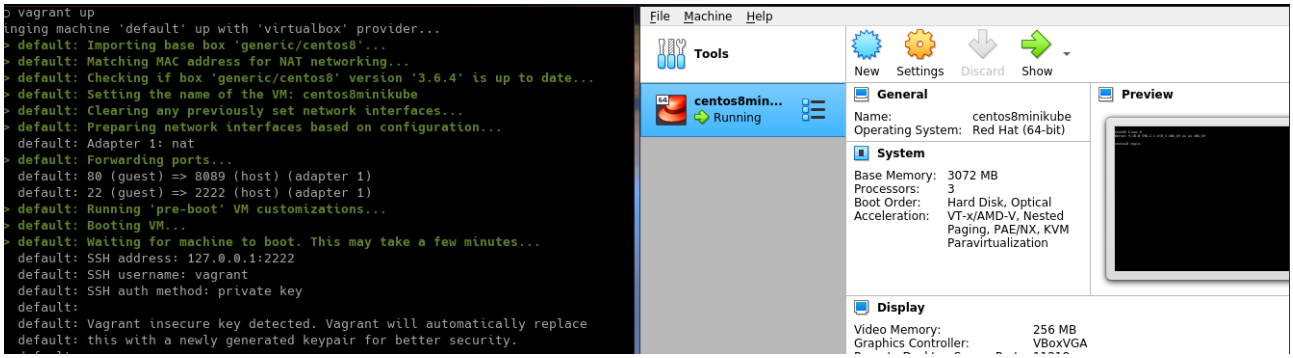


Figura 7.5: Levantar una máquina virtual con Vagrant

Algoritmo 7.2: Acceder mediante ssh a la máquina virtual levantada

```
$ vagrant ssh
```

```
└─○ vagrant ssh
Last login: Wed Jan 26 19:56:54 2022 from 10.0.2.2
[vagrant@minikubelocal ~]$ whoami
vagrant
[vagrant@minikubelocal ~]$ cat /etc/os-release
NAME="CentOS Linux"
VERSION="8"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="8"
PLATFORM_ID="platform:el8"
PRETTY_NAME="CentOS Linux 8"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:8"
HOME_URL="https://centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"
CENTOS_MANTISBT_PROJECT="CentOS-8"
CENTOS_MANTISBT_PROJECT_VERSION="8"
```

Figura 7.6: Conexión ssh

Algoritmo 7.3: Lista los boxes de vagrant en la máquina

```
$ vagrant box list
```

Figura 7.7: Lista de boxes

Algoritmo 7.4: Detiene la máquina de vagrant ejecutándose

```
$ vagrant halt
```

Figura 7.8: Detener la máquina ejecutándose

Algoritmo 7.5: Detruye la máquina de vagrant establecida

```
$ vagrant destroy
```

Figura 7.9: Destruir la máquina

ANEXO 8

Raspberry Pi 4

8.1. Elementos del Raspberry Pi 4

Para el funcionamiento del Raspberry Pi 4, se requiere unicamente del cargador y de la placa, la cual es en si el Raspberry Pi, la placa que almacena toda la funcionalidad. Sin embargo el paquete *Bqeel Raspberry Pi 4 Model 4GB*, el cual fue adquirido. Otorga los elementos y herramientas necesarias para preservarlo, y de tal modo evitar un daño en el dispositivo. Existen muchas otras soluciones en el mercado, pero en esta ocación emplearemos la emitida por Bqeel.

Figura 8.1: Vista general de los elementos

Figura 8.2: Cargador

Figura 8.3: Cables micro hdmi

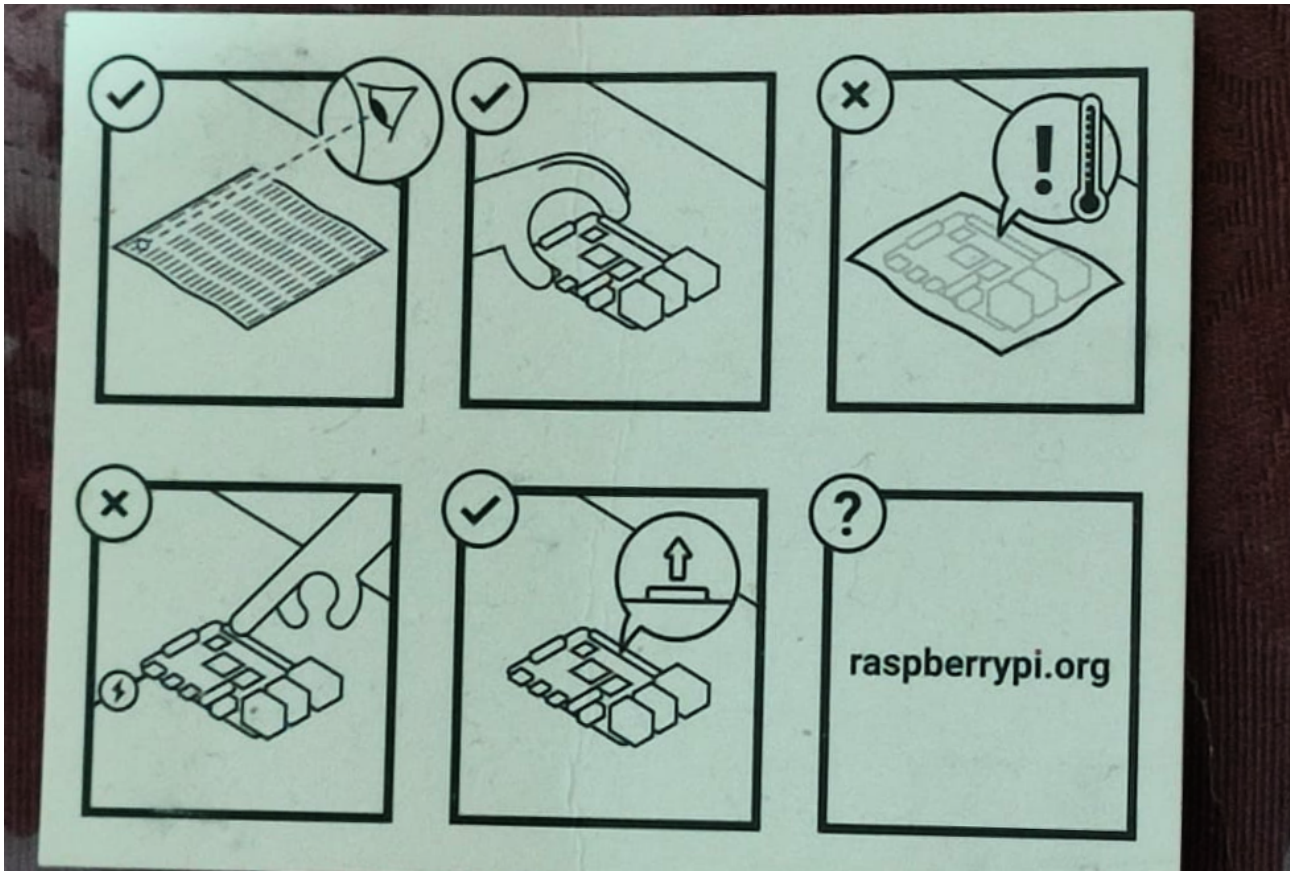


Figura 8.4: Indicaciones

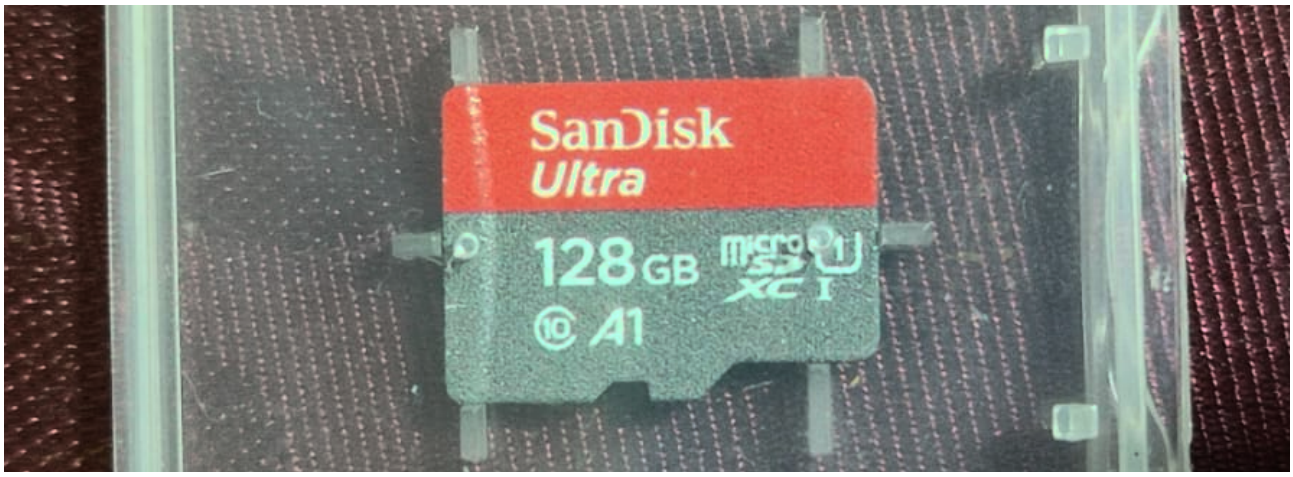


Figura 8.5: Tarjeta microSD

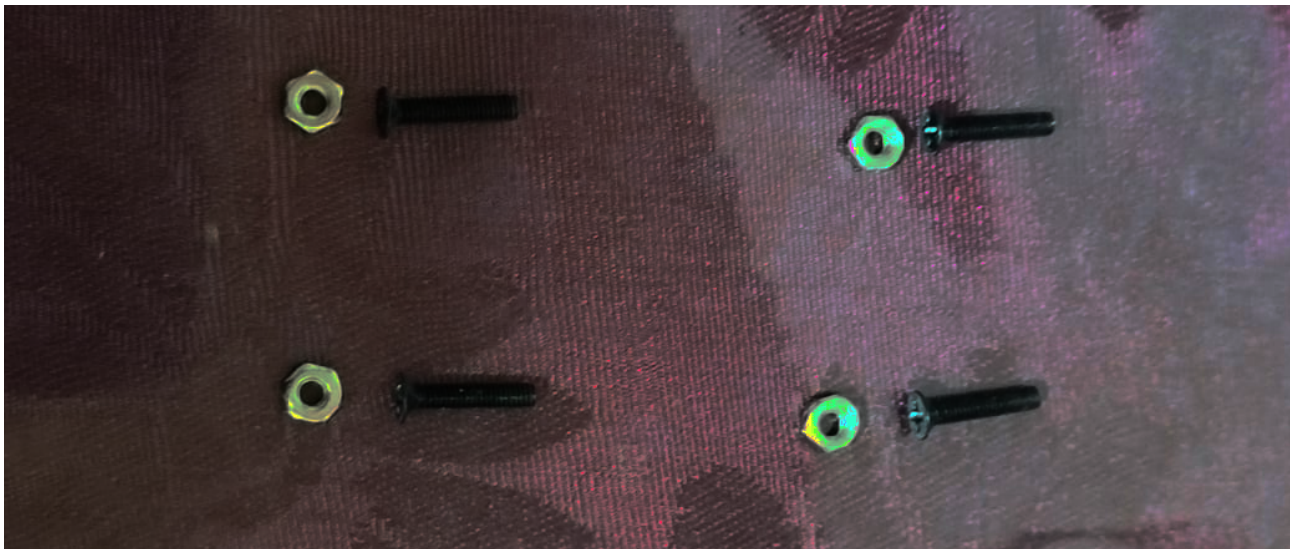


Figura 8.6: Tornillos

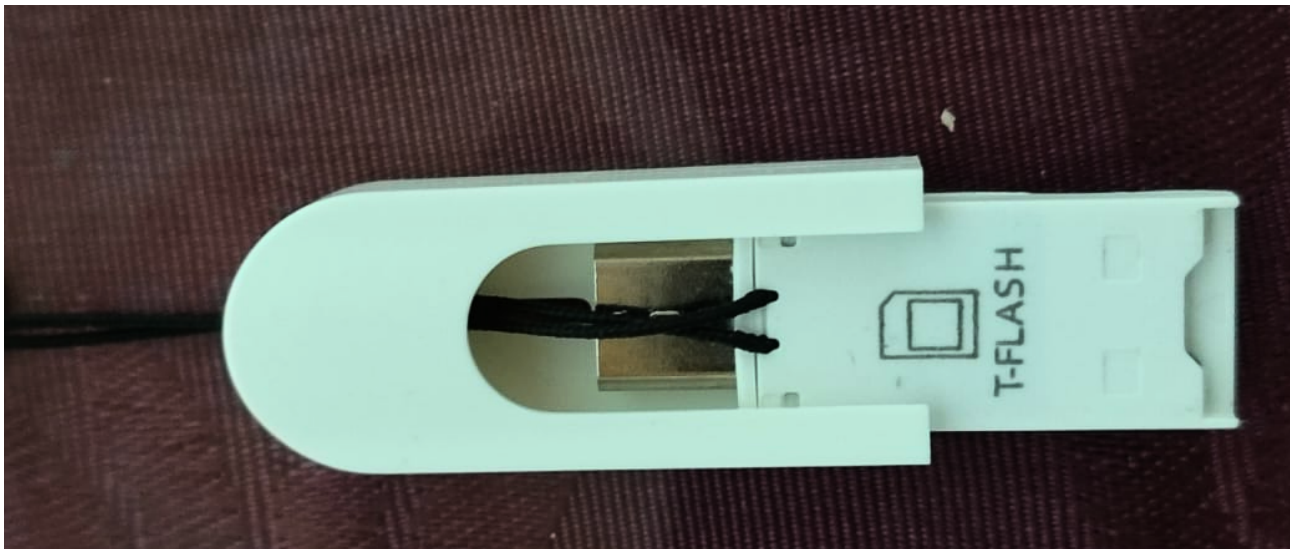


Figura 8.7: Card Reader

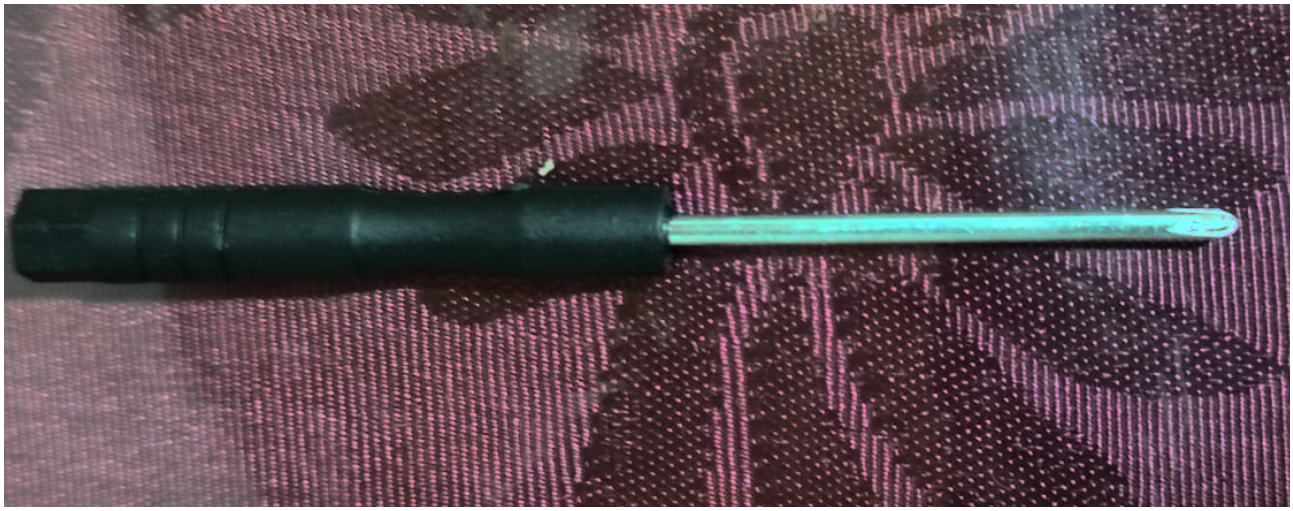


Figura 8.8: Destornillador

Figura 8.9: Ventilador

Figura 8.10: Parte baja del Raspberry Case

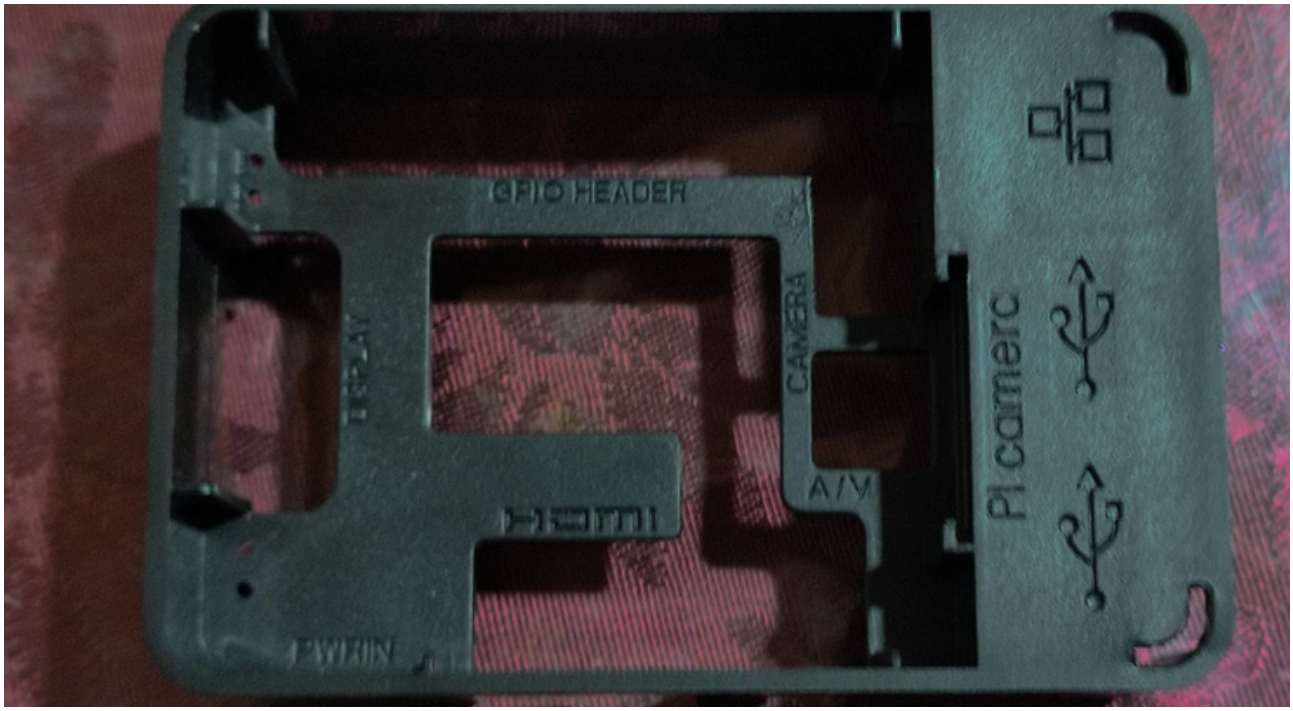


Figura 8.11: Parte intermedia del Raspberry Case

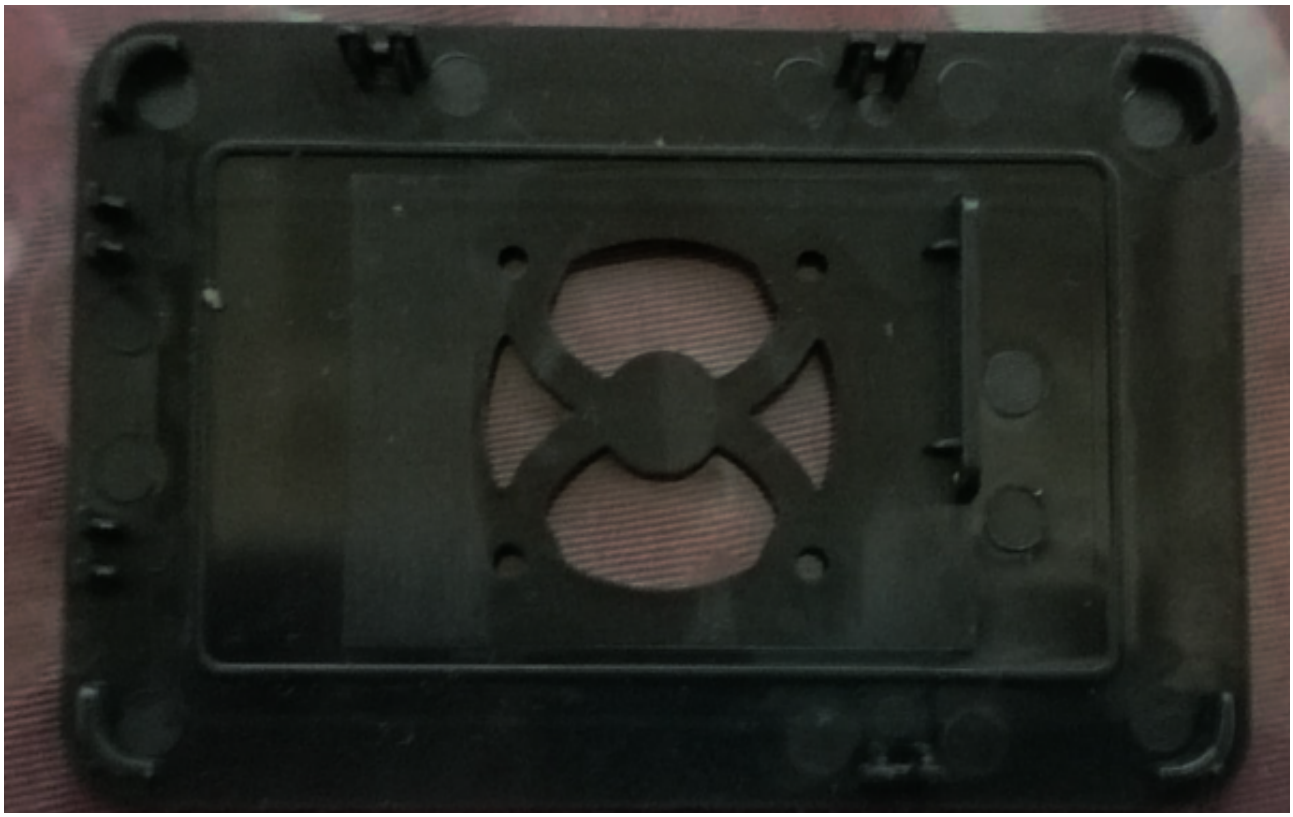


Figura 8.12: Parte superior del Raspberry Case

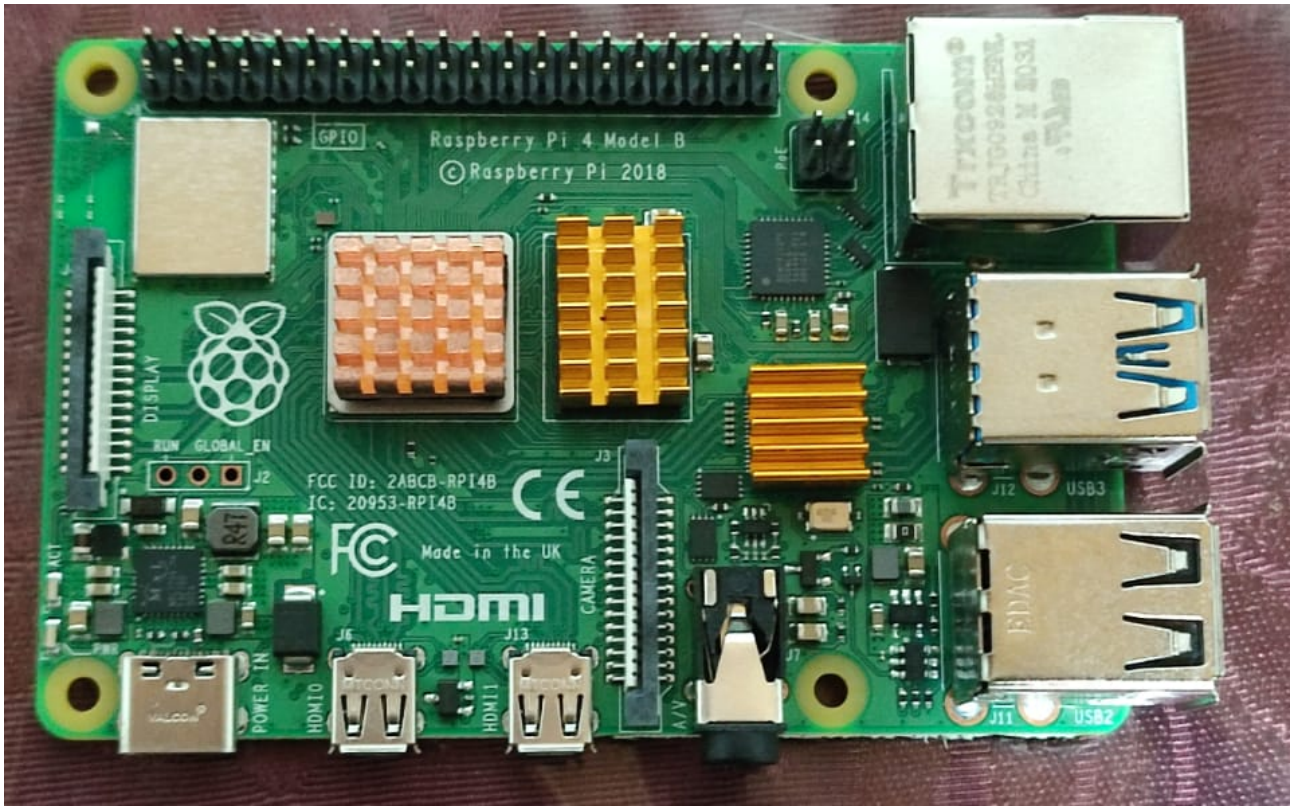


Figura 8.13: Raspberry Pi 4

8.2. Ensamblaje del Raspberry Pi 4

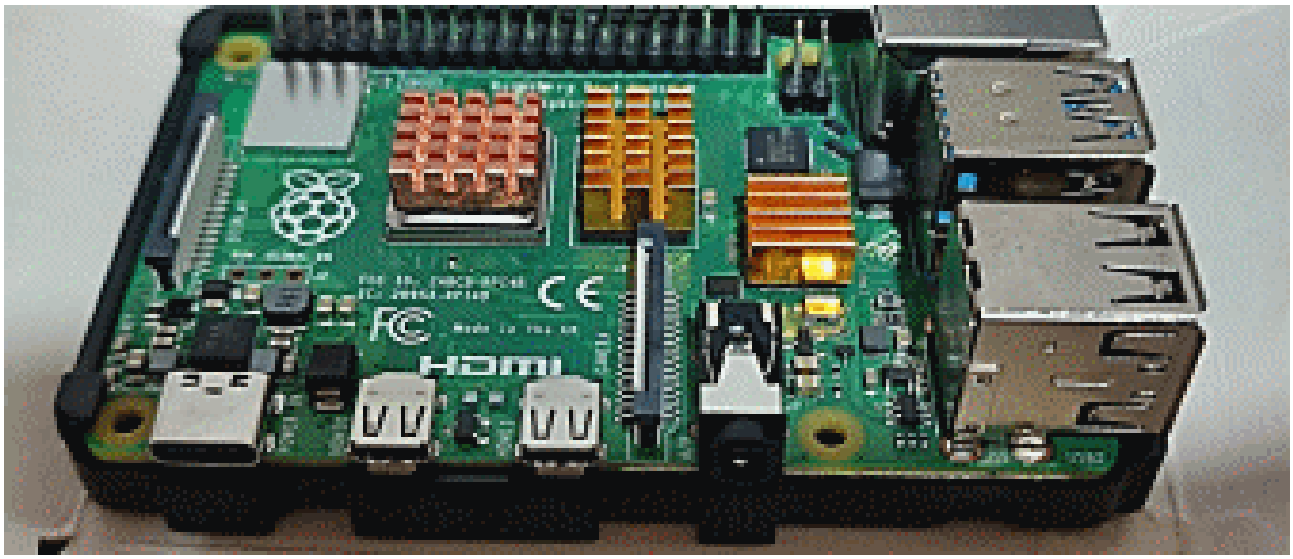


Figura 8.14: Unión de la parte inferior del Raspberry Case con el Raspberry Pi

Figura 8.15: Unión de la parte intermedia del Raspberry Case con la sección anteriormente armada

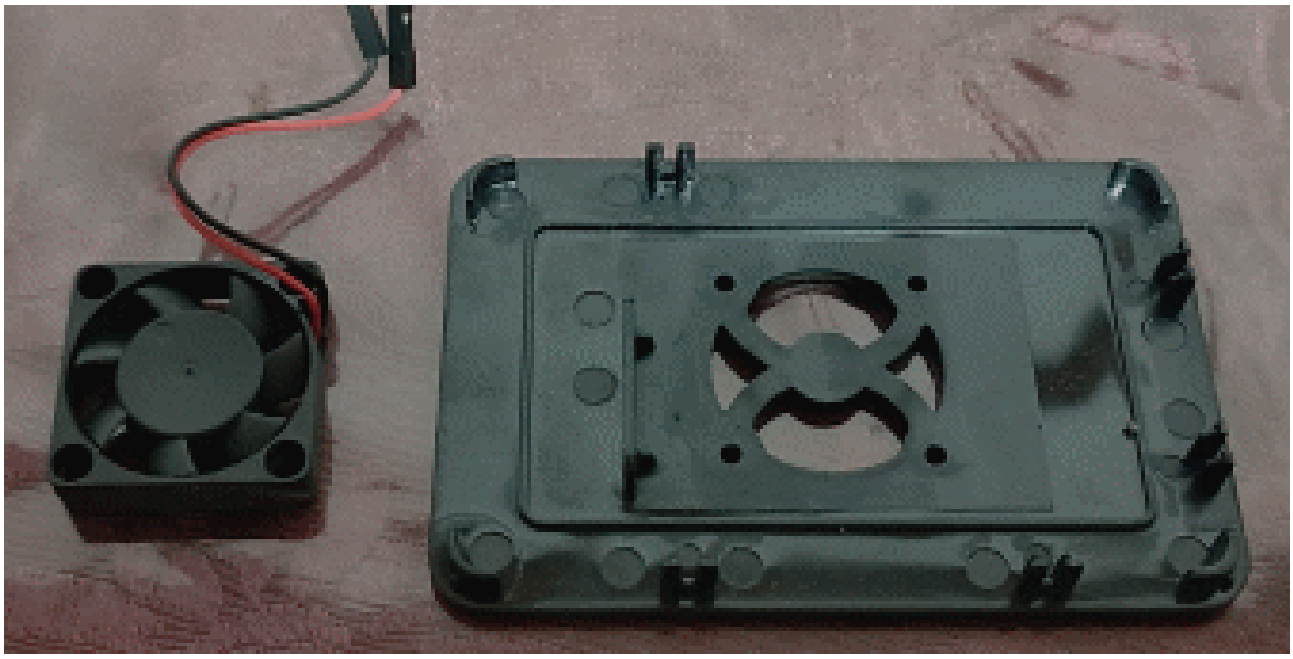


Figura 8.16: Unión del ventilador con la parte superior del Raspberry Case

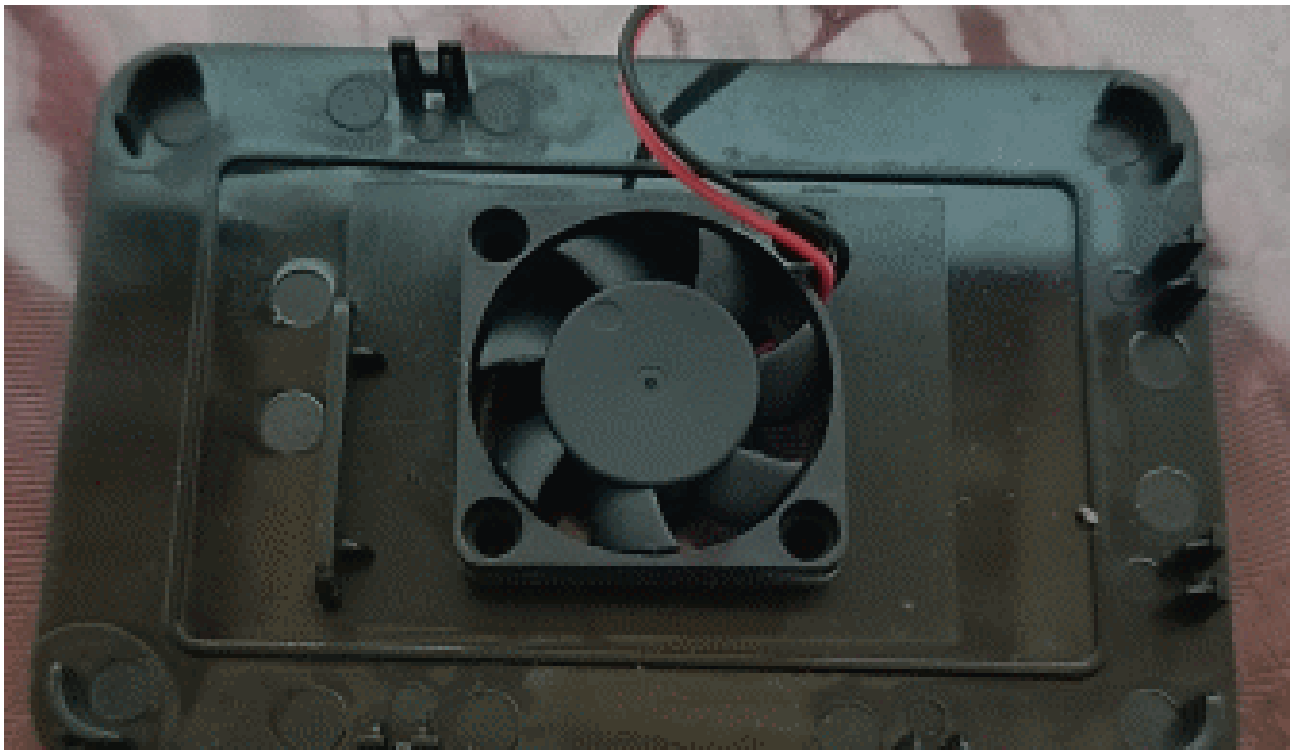


Figura 8.17: Posicionamiento del ventilador en la siguiente postura

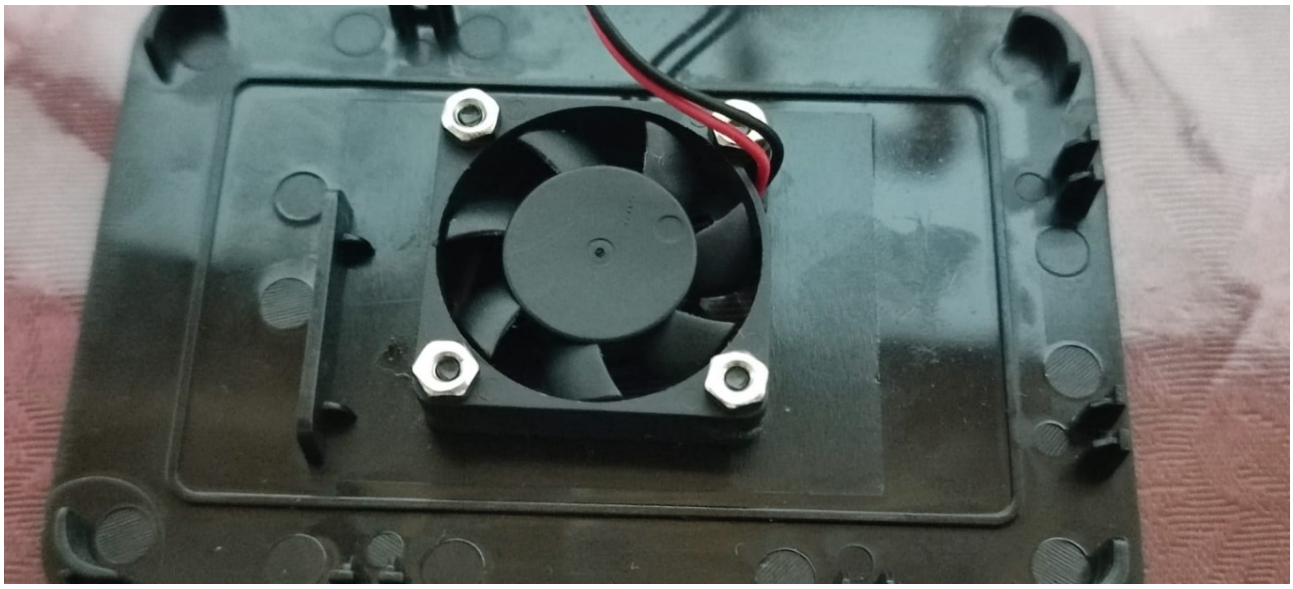


Figura 8.18: Atornillamiento de la parte superior del Raspberry Case con el ventilador, en la posición previamente establecida

Figura 8.19: Atornillamiento de la parte superior del Raspberry Case con el ventilador, en ambos lados.

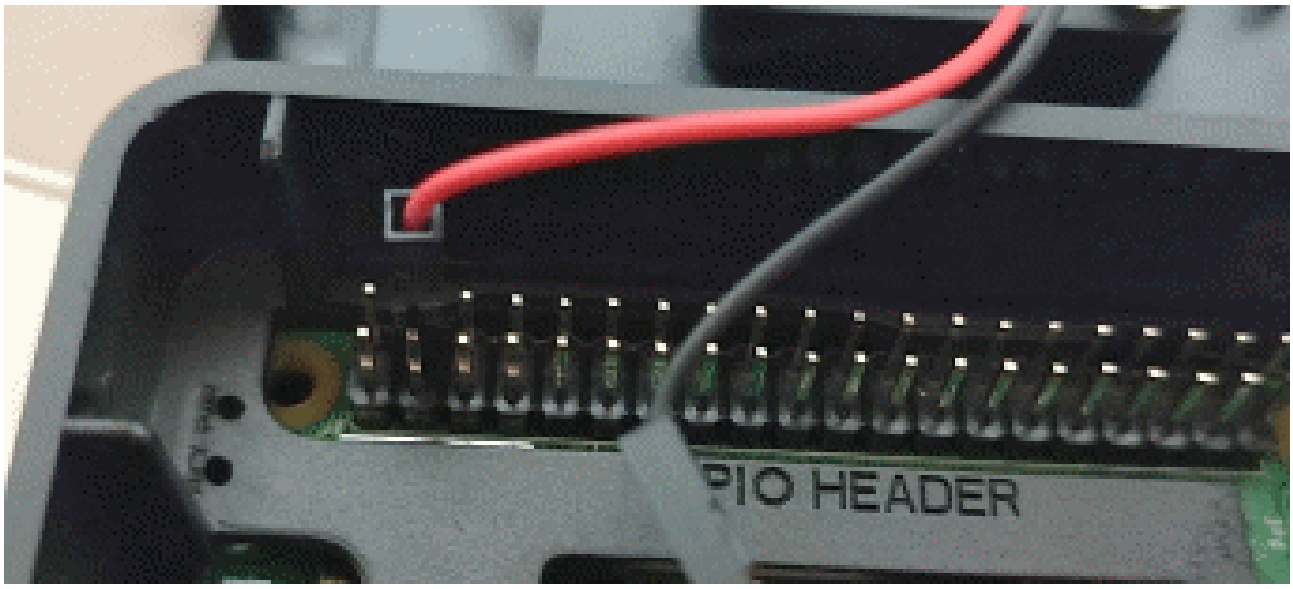


Figura 8.20: Conexión del cable rojo.

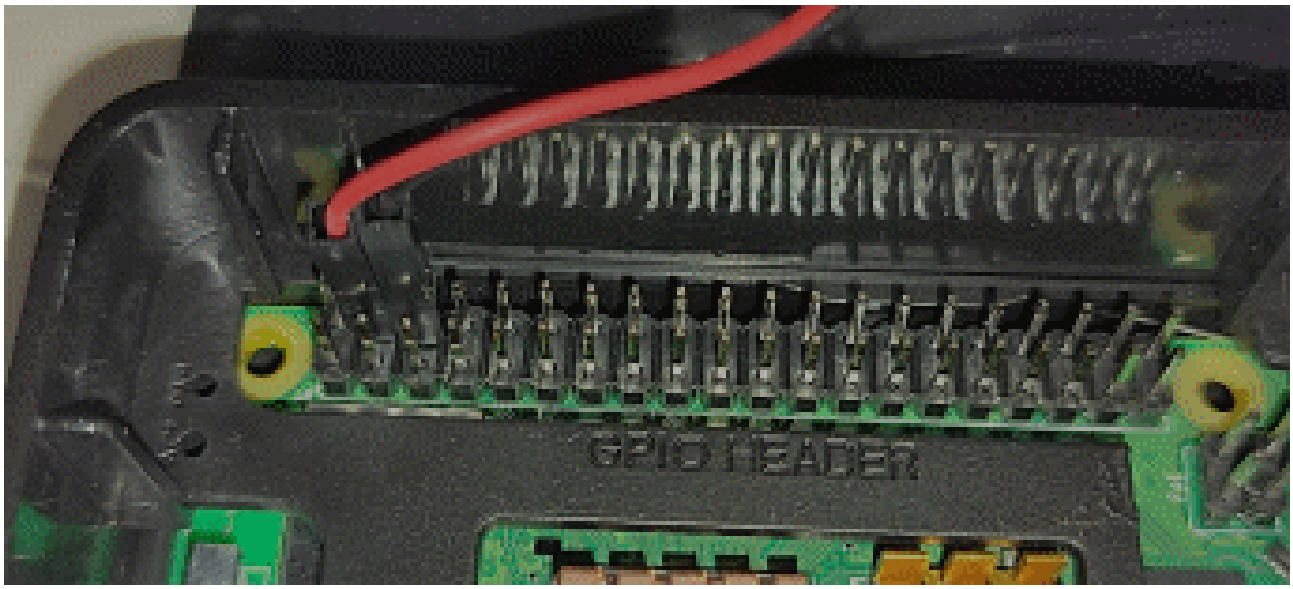


Figura 8.21: Conexión del cable negro próximo al cable rojo.

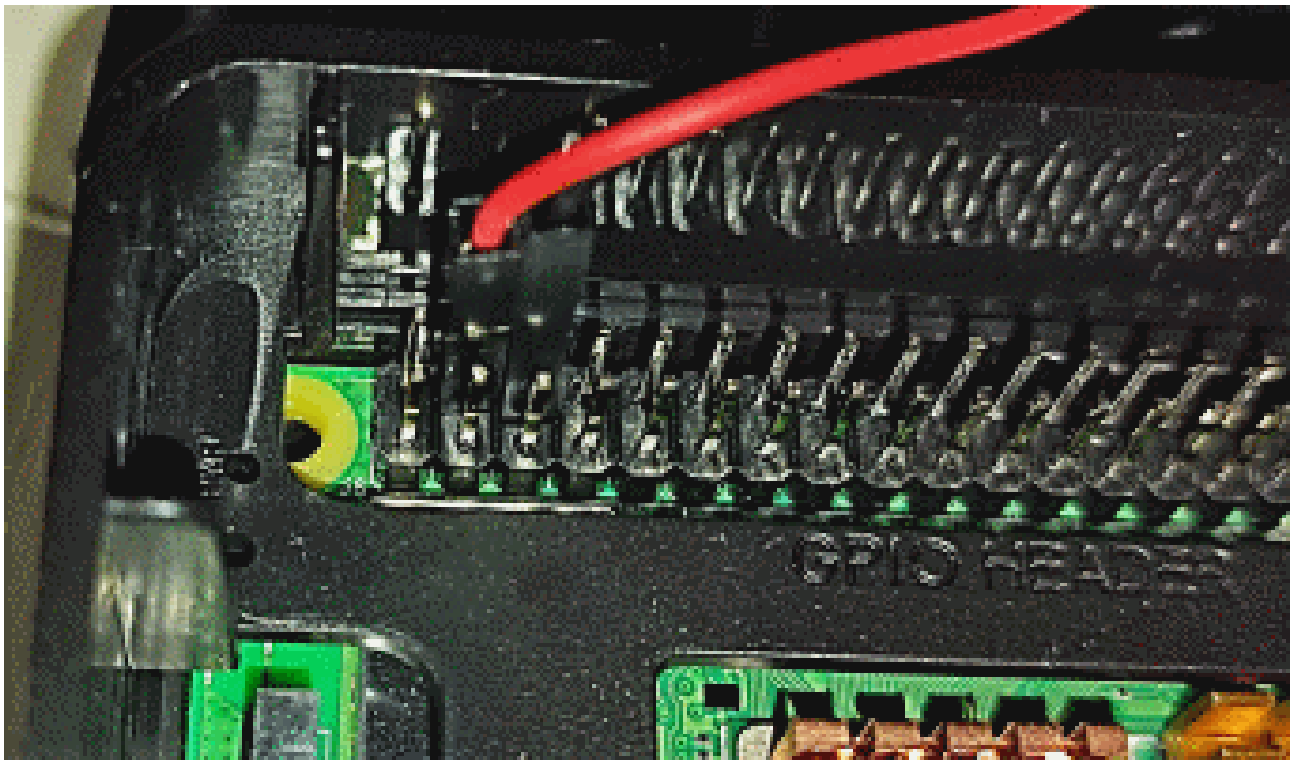


Figura 8.22: Verificación del posicionamiento de los cables.

Figura 8.23: Tapar el dispositivo.

8.3. Pruebas de funcionalidad del Raspberry Pi 4

Figura 8.24: Conexión del cargador.

Figura 8.25: Conexión del Cargador al regulador de voltaje.

Figura 8.26: Encender el Raspberry Pi.

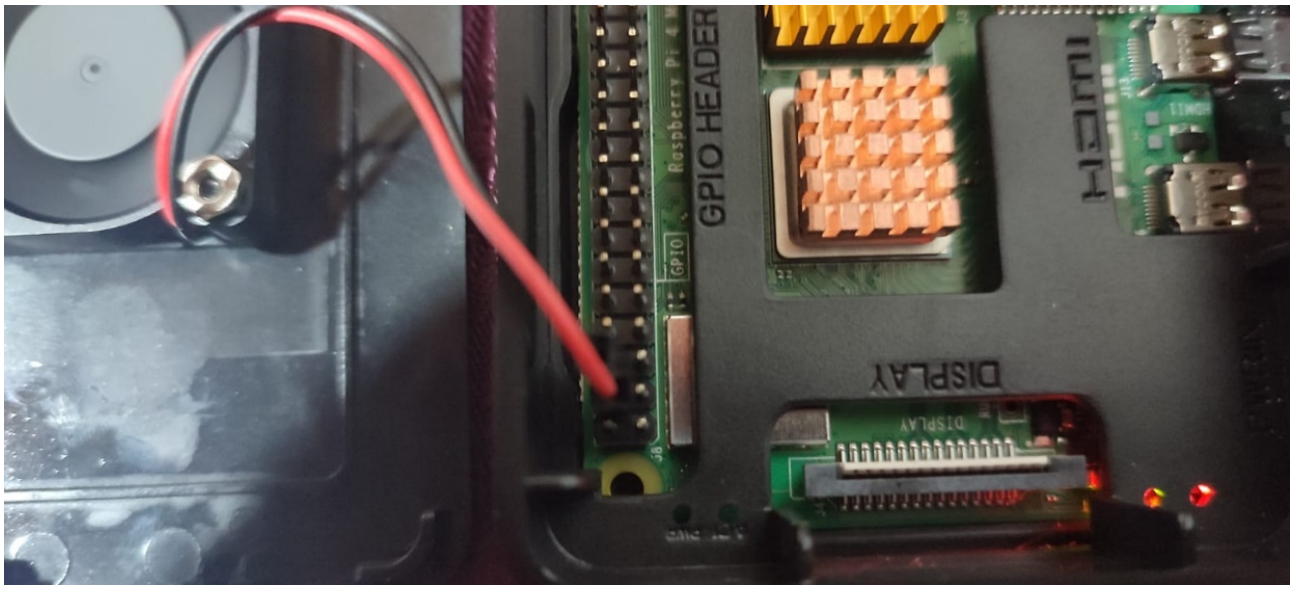


Figura 8.27: Verificación del funcionamiento integrado del ventilador y el Raspberry Pi.

8.4. Configuración de la ISO del Raspberry Pi 4

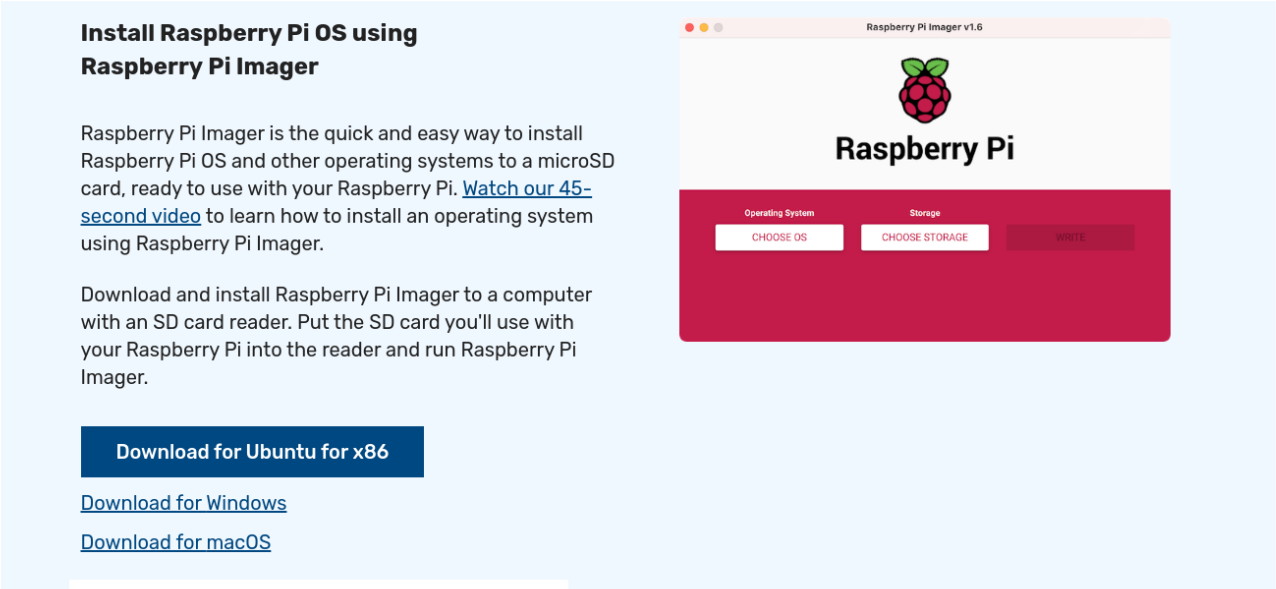


Figura 8.28: Descarga del Raspberry Pi Imager.

```
wesly at wesly in ~/Downloads/DownloadsChromium
└─o sudo apt install ./imager_1.6.2_amd64.deb
[sudo] password for wesly:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'rpi-imager' instead of './imager_1.6.2_amd64.deb'
The following packages were automatically installed and are no longer required:
  libadplug-2.3.1-0 libaudiofile1 libbinio1v5 libfaad2 libfluidsynth2 libinstpa
  libsidplayfp4 libupnp13 libwildmidi2 linux-headers-5.11.0-38-generic linux-hw
  linux-modules-extra-5.11.0-38-generic timgm6mb-soundfont
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libqt5qml5 libqt5quick5 libqt5quickcontrols2-5 libqt5quicktemplates2-5 qml-mo
  qml-module-qtquick-templates2 qml-module-qtquick-window2 qml-module-qtquick2
Suggested packages:
```

Figura 8.29: Instalación del Raspberry Pi Imager en Ubuntu.

Figura 8.30: Inserción del microSD requerido para la instalación de un Sistema Operativo en el Raspberry Pi.

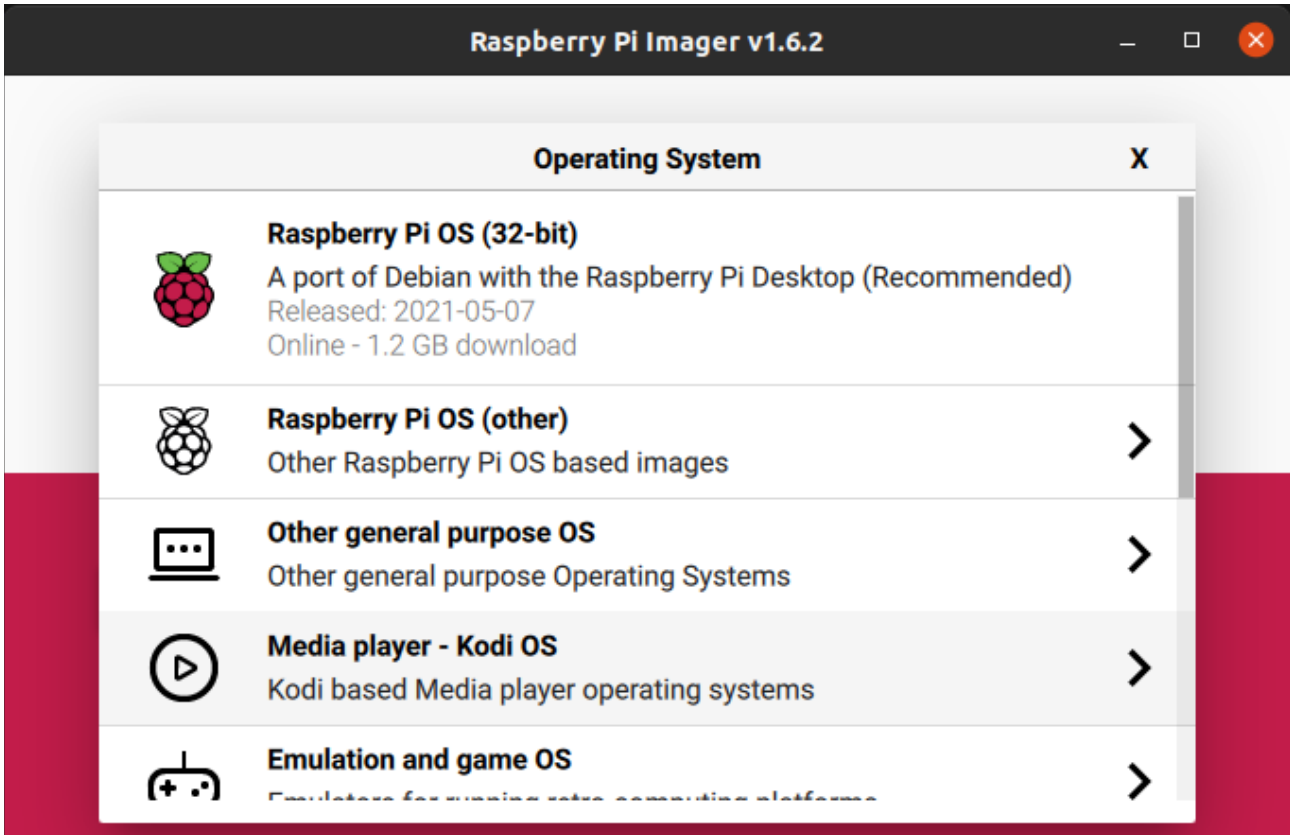


Figura 8.31: Selección del Raspberry Pi OS ha instalar.

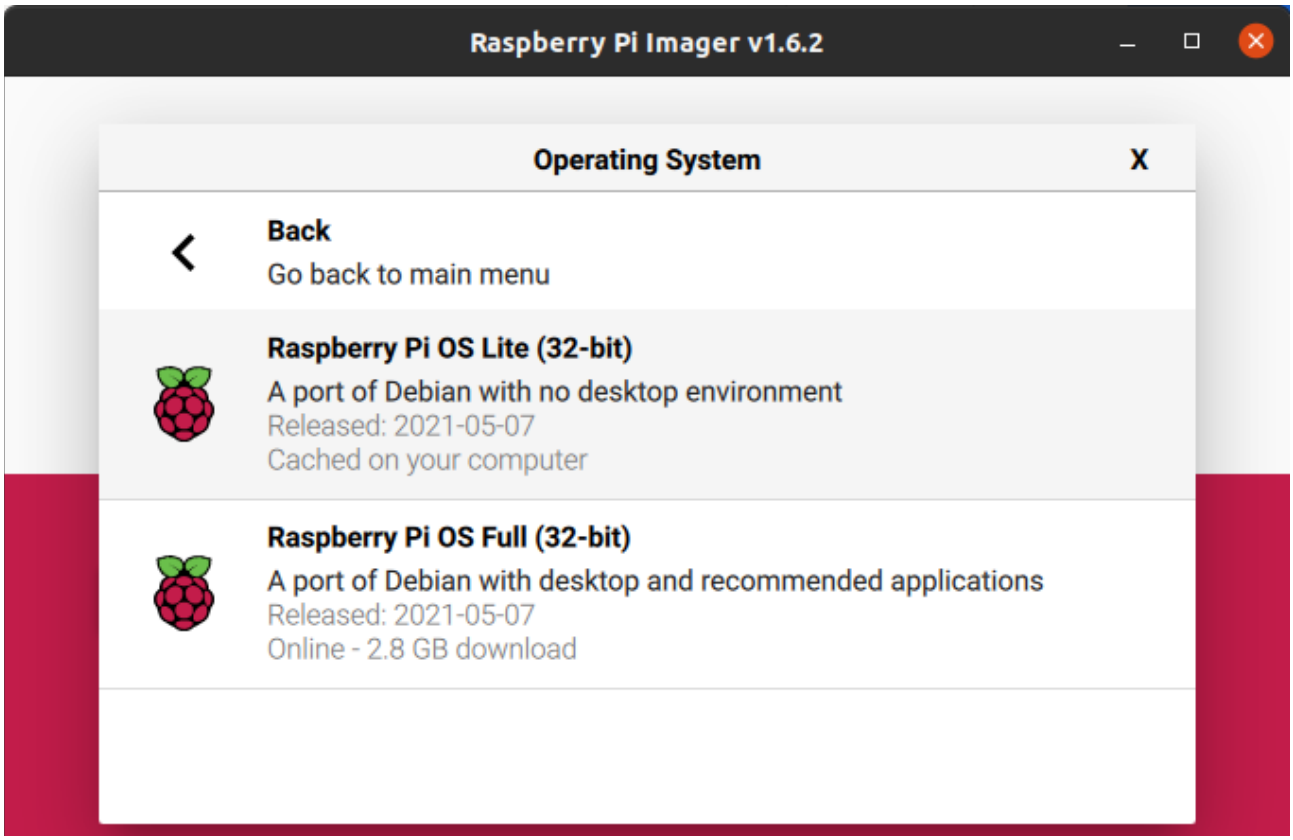


Figura 8.32: Elección de la imagen Lite, únicamente para tener una terminal, omitiendo la interfaz gráfica.

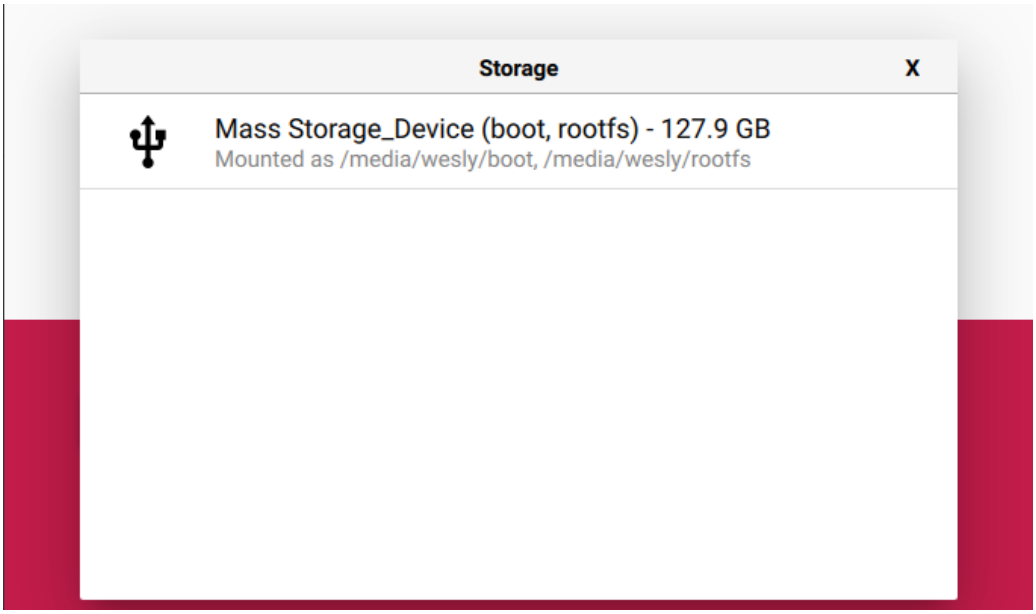


Figura 8.33: Selección en store del dispositivo en donde será booteada la ISO.

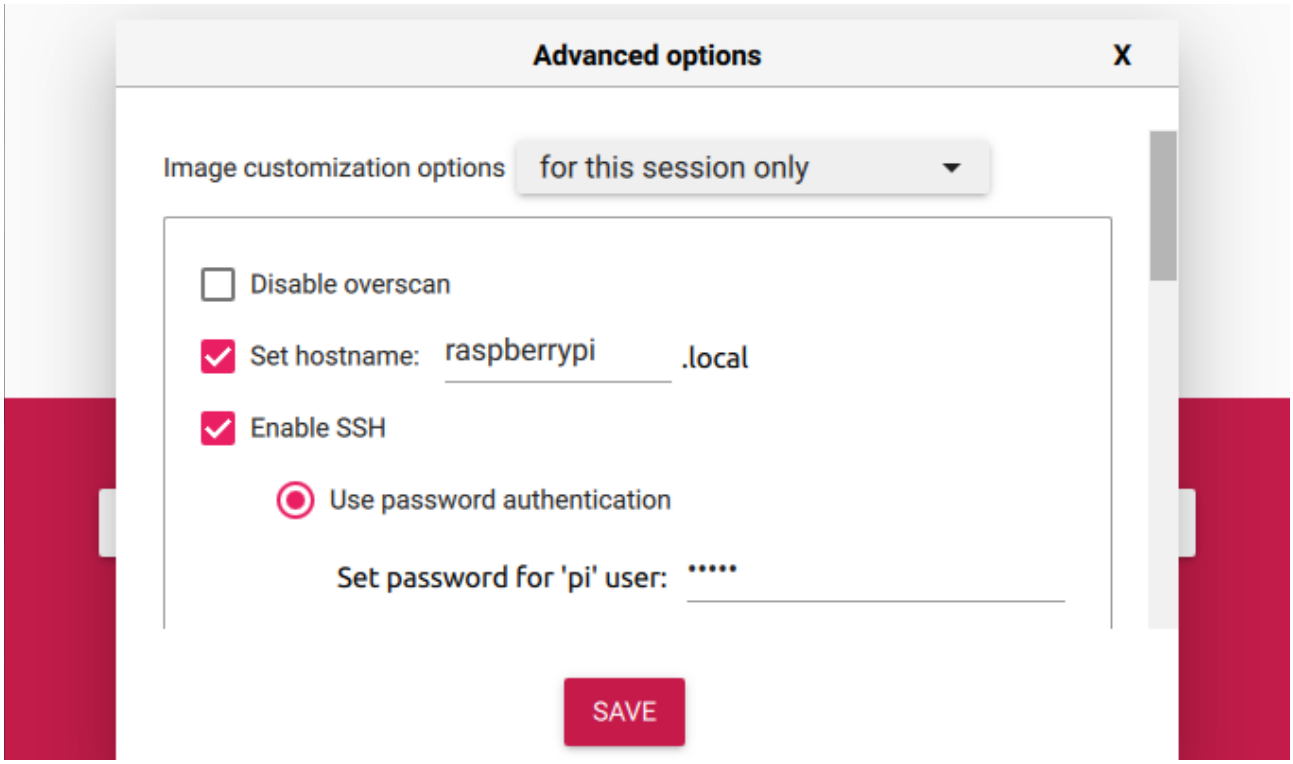


Figura 8.34: Pulsar *"ctrl + shift+ x"* y habilitar el puerto ssh.

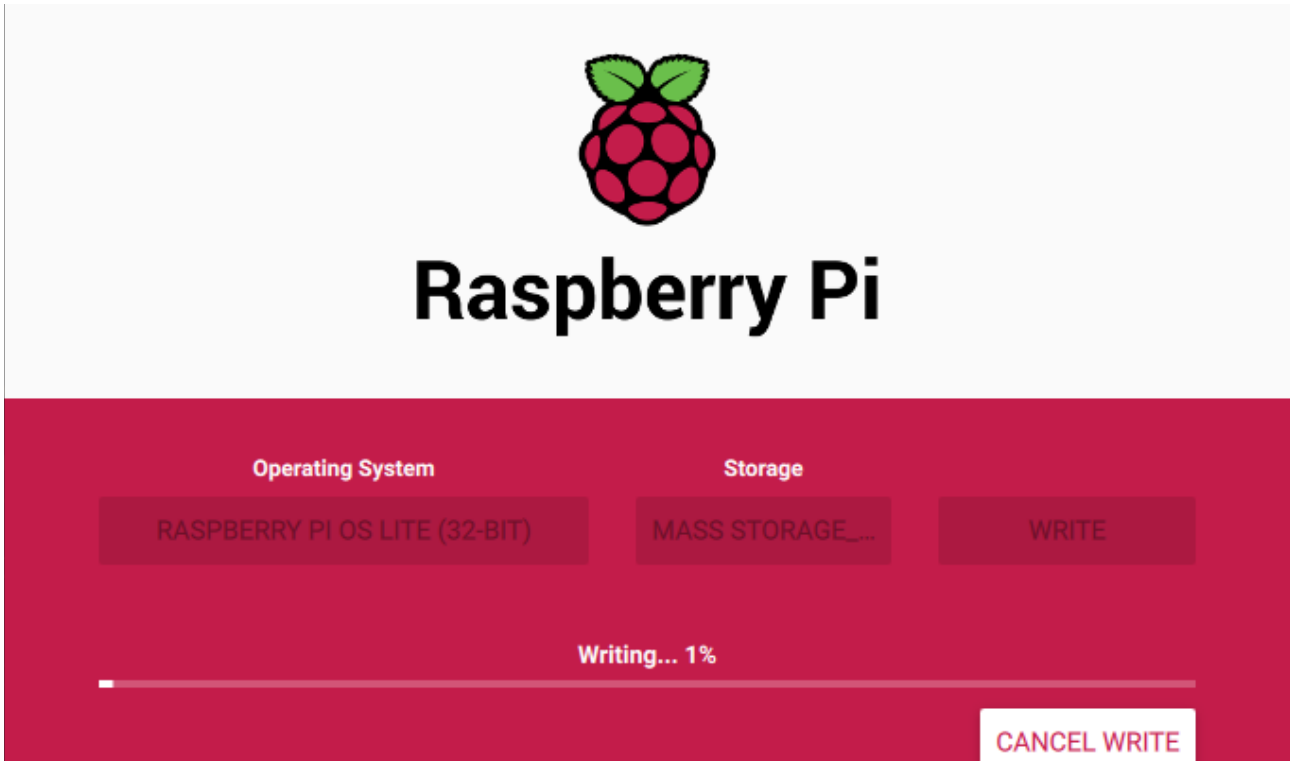


Figura 8.35: Inicio del booteo del Raspberry Pi con una imagen ligera.

8.5. Configurar el WIFI en el Raspberry Pi 4

Se debe conectar el Raspberry Pi a una red LAN, es decir se lo debe enlazar mediante un cable de red directamente al router, con la finalidad de tener acceso a internet.

Figura 8.36: Conexión del Raspberry Pi a una red LAN.

En caso de que al intentar realizar una conexión ssh mediante el hostname que se proporcionó al momento de realizar el booteo del Raspberry Pi, de la siguiente manera:

```
ssh raspberrypi.local -l pi
```

Si el acceso al dispositivo es rechazado, es preferible escanear los puertos y obtener la ip determinada. Por ello, para tal acción se emplea *nmap*.

8.5.1. Escaneo de la red

Para escanear la red se requiere obtener la dirección ip del router y la máscara de la subred. Por ende se emplea el siguiente comando:

```
nmap -sn ip_router/CIDR
```

```
nmap -sn 192.168.1.1/26
```

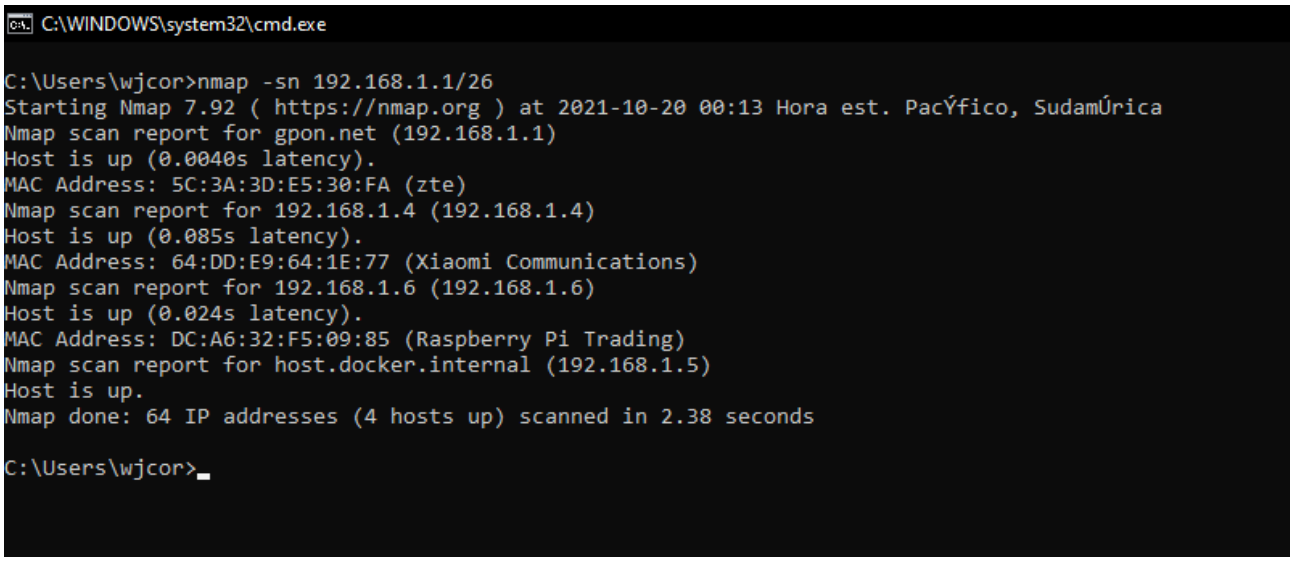


Figura 8.37: Escaneo de la red

De tal modo que la dirección ip del Raspberry Pi es la ***192.168.1.6***

8.5.2. Acceso al Raspberry Pi

Para ingresar al Raspberry Pi se requiere una terminal para establecer una conexión ssh, y de digitar el comando:

```
ssh 192.168.1.6 -l pi
```

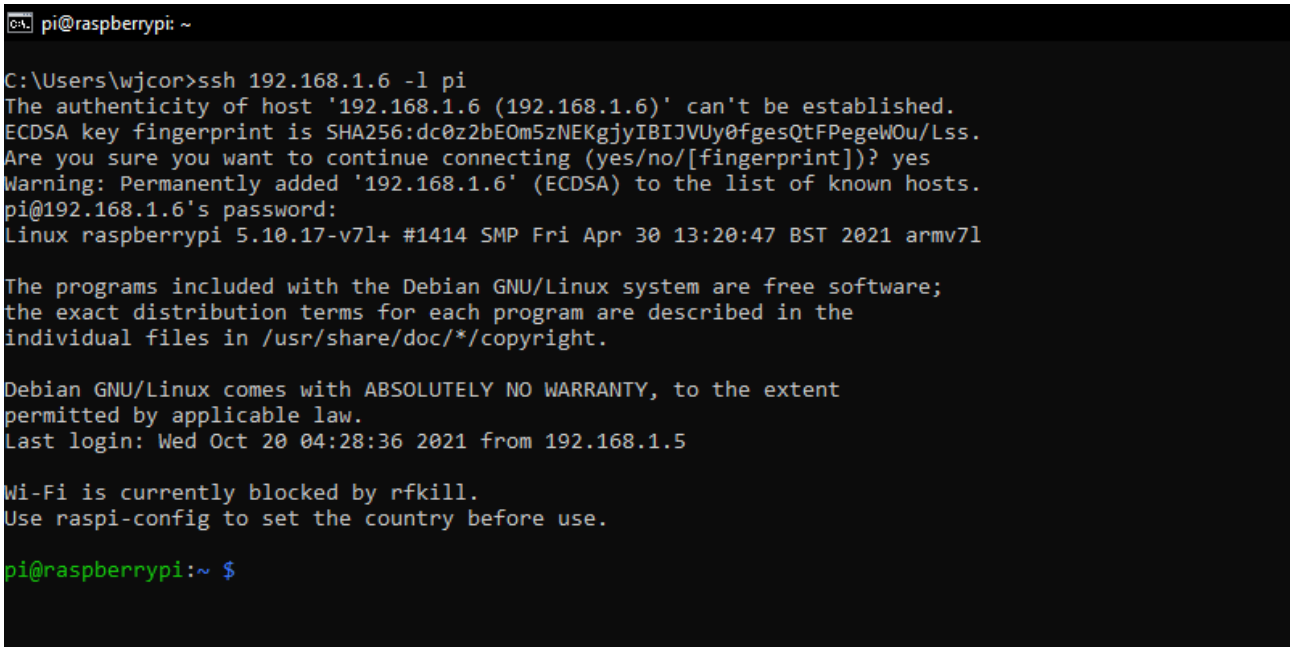


Figura 8.38: Ingreso al Raspberry Pi

8.5.3. Habilitar el Wireless LAN para el acceso inalambrico

Al ganar acceso al Raspberry Pi, en la terminal se deberá continuar con los siguientes pasos:

```
sudo raspi-config
```

Por consiguiente con el tabulador se debe seleccionar System Options.

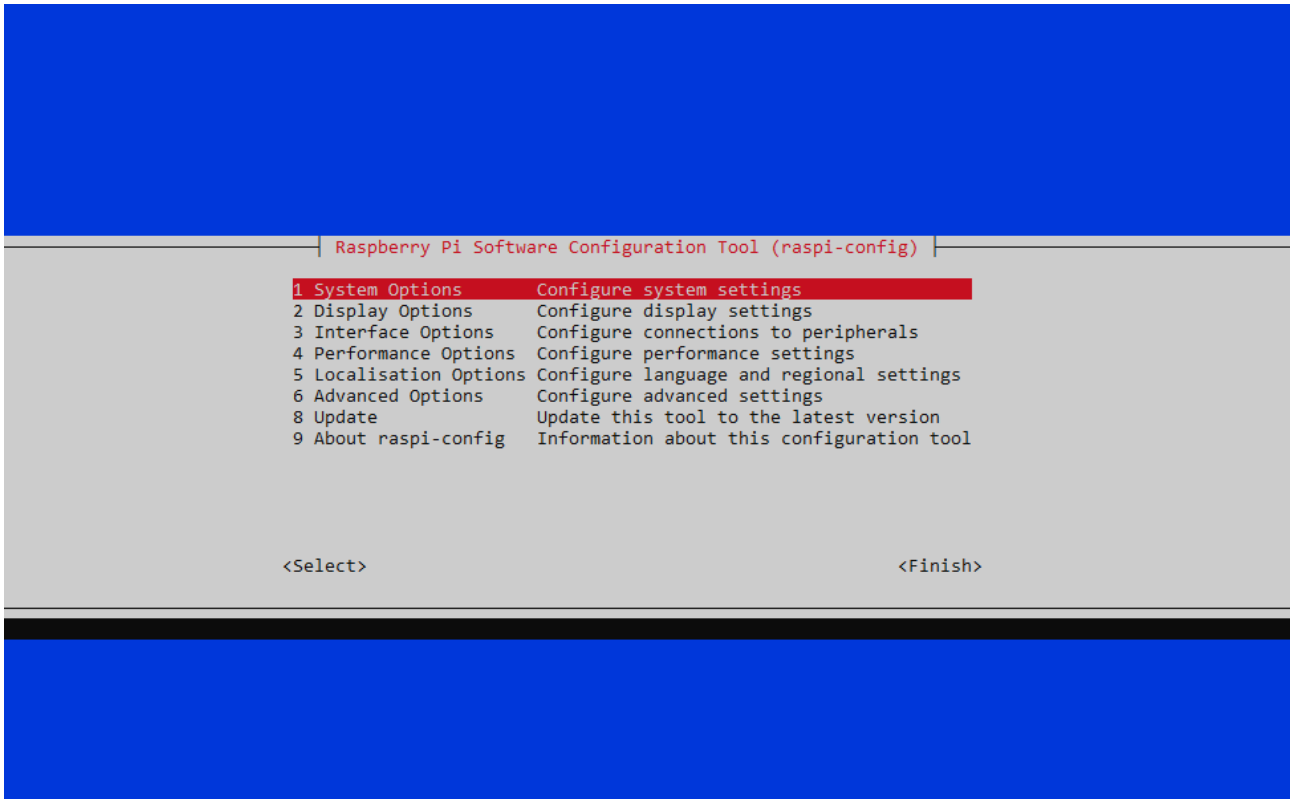


Figura 8.39: System Options

Se debe seleccionar Wireless LAN.

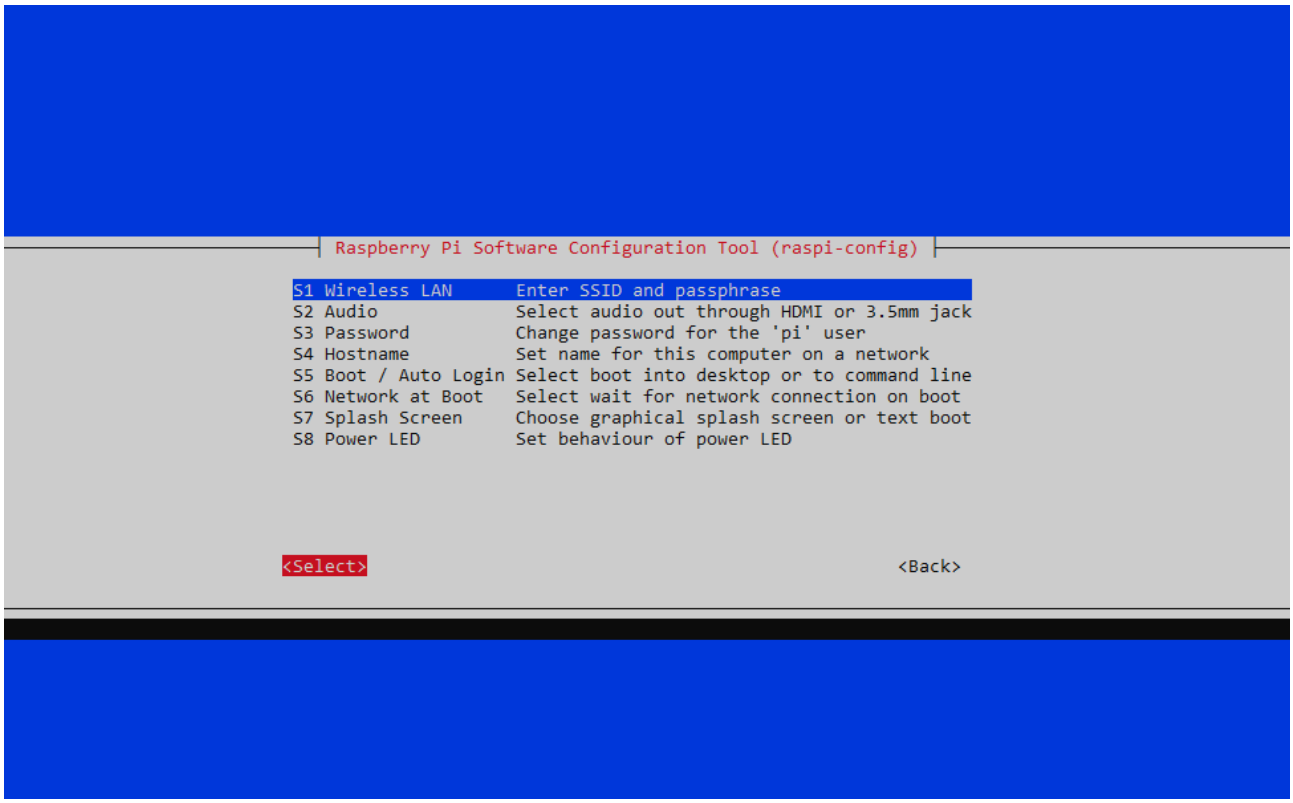
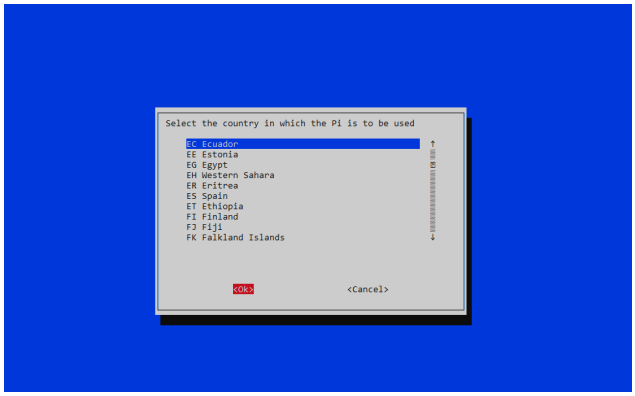


Figura 8.40: Wireless LAN

Seleccionar el país.



(a) Selección del país.

(b) Confirmación de la región.

Figura 8.41: Configuración de la región.

Ingresar el SSID, que vendría a ser el nombre de la red a la cual el dispositivo se desea conectar.

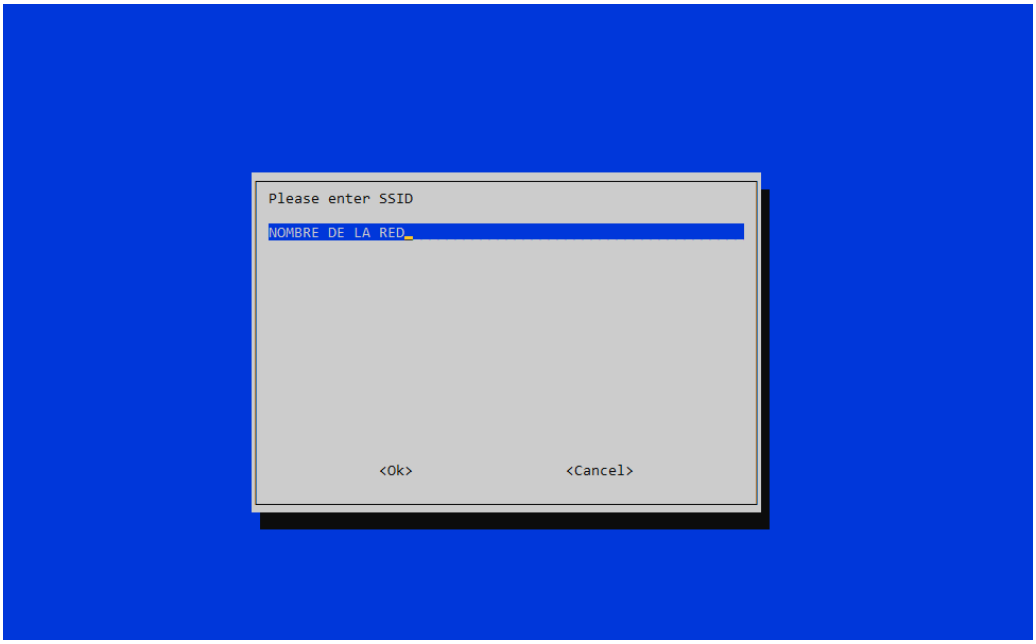


Figura 8.42: Ingreso del SSID.

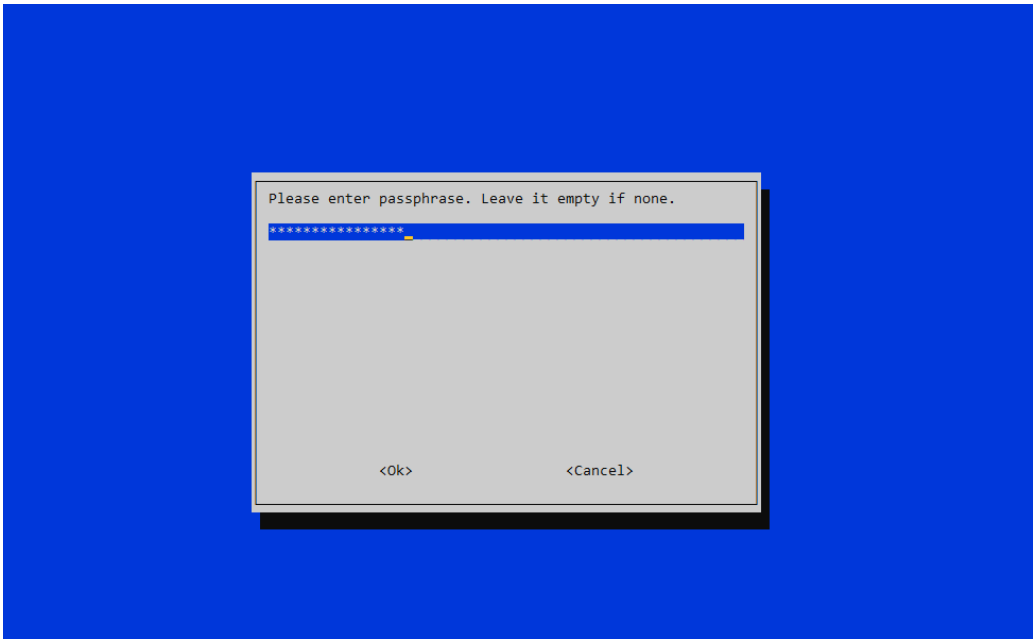


Figura 8.43: Ingreso de la contraseña de la red.

Una vez hecha la configuración se finalizará el proceso y se procederá a reiniciar el Raspberry Pi.

```
sudo reboot
```

Por finalizar, se deberá retirar el cable de red ya que no será necesario, debido a que el dispositivo obtuvo acceso a la red mediante WLAN.

8.5.4. Configuración del DHCP en el Raspberry Pi

Al reiniciar el dispositivo, y al intentar acceder al mismo por la ip 192.168.1.6, se presentaría la idea de que existen fallos en la configuración previa y por ende el acceso seria denegado, pero, la verdad es que, al reiniciar el dispositivo, la dirección ip con la cual el router habilitó el acceso cambió, por ello es necesario establecer una ip por defecto con la finalidad de obtener una manera de permitir el ingreso constantemente, por ende se establece una dirección ip fija mediante el protocolo DHCP.

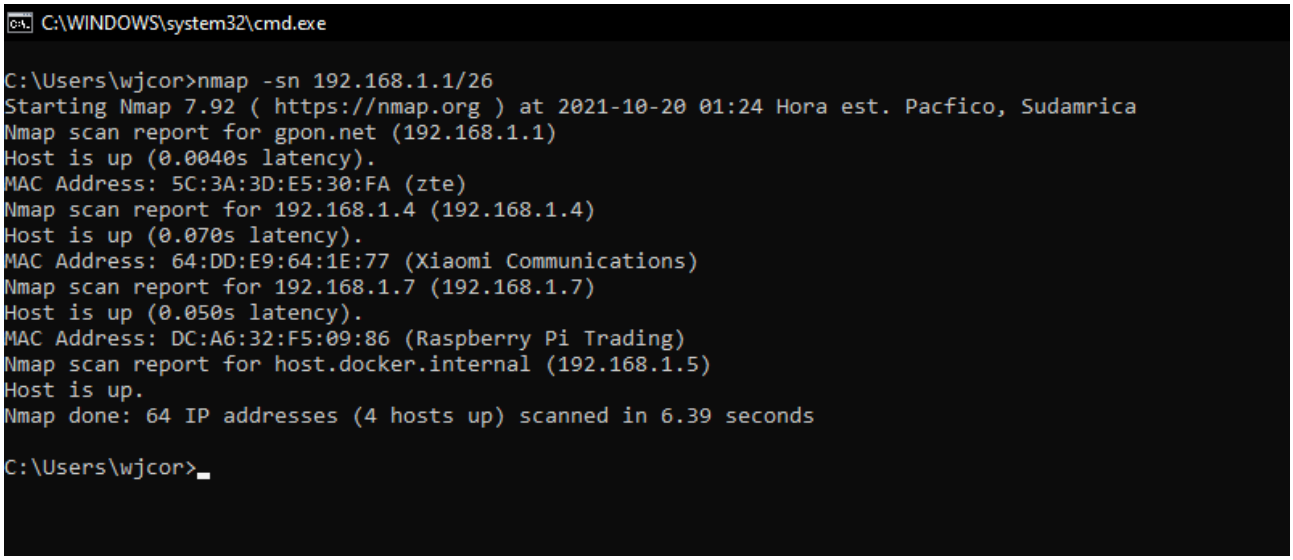


Figura 8.44: Escaneo de la red después del reinicio.

Por consiguiente, se debe ingresar al Raspberry Pi mediante la nueva dirección ip asignada, para ello se debe ingresar el comando:

```
ip -brief -color -4 a
```

El cual muestra la dirección ip agregada al Raspberry Pi.

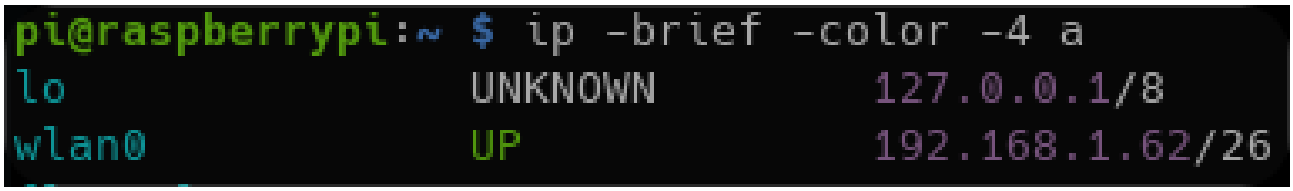


Figura 8.45: Obtención de la dirección ip del Raspberry Pi.

De forma detallada se observa que la dirección ip respectiva,es la 192.168.1.7

De igual forma, se observa que la dirección ip del broadcast es la 192.168.1.63, la misma que no puede ser usada en el proceso para establecer una IP fija por medio del protocolo DHCP, lo mismo sucede con la dirección ip 192.168.1.1, la cual se encuentra destinada para el router.

Del comando presentado en la imagen anterior se observa que el Wireless Local Area Network es wlan0, el mismo que será necesario para configurar el DHCP.

Para habilitar el servicio de DHCP en el Raspberry Pi, se debe editar el fichero "/etc/dhcpcd.conf", el cual puede ser abierto con el editor de texto de preferencia, en tal caso se empleará vim, "vim /etc/dhcpcd.conf", y se agregará las siguientes líneas de código:

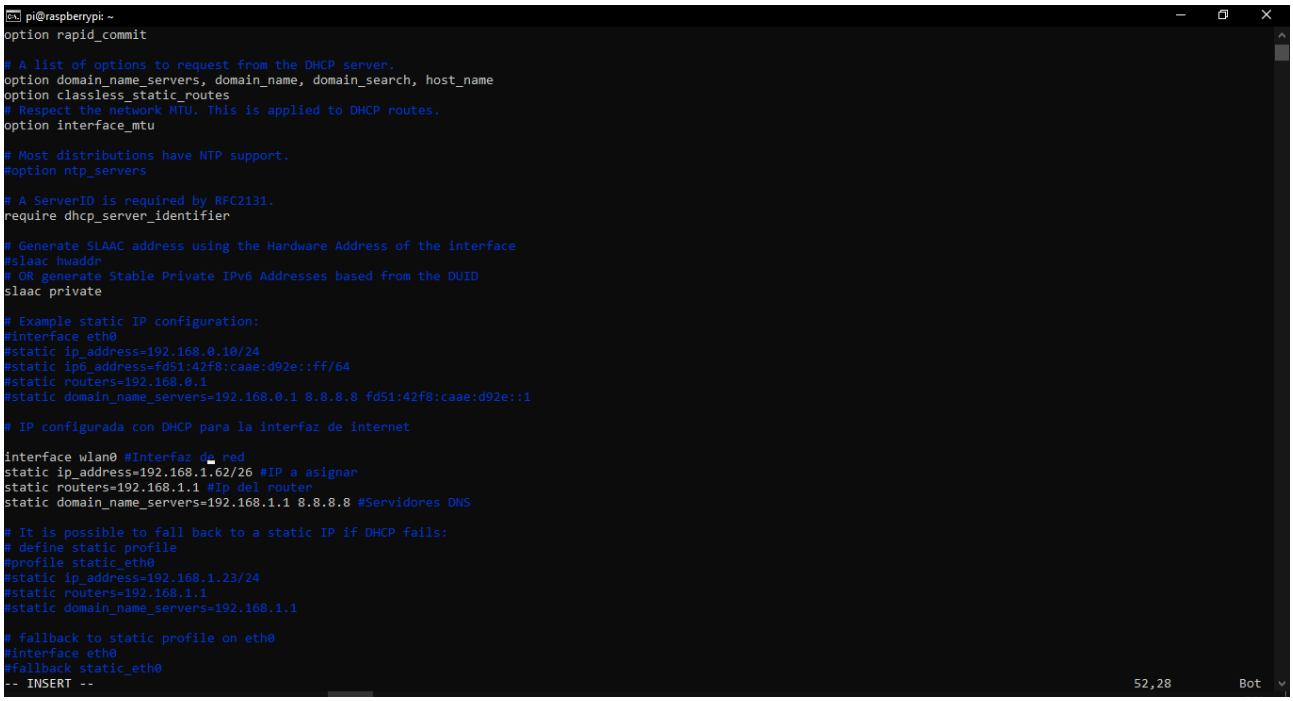


Figura 8.46: Edición del fichero /etc/dhcpcd.conf.

- interface= es la Wireless Local Area Network.
- ip_address= es la dirección ip a asignar.
- routers= es la IP del router.
- domain_name_servers= son los servidores DNS.

Finalmente una vez guardado los cambios, se debe reiniciar el dispositivo nuevamente, con ello los cambios serán fijados de manera permanente.

ANEXO 9

Instalación de minikube en CentOS integrado a kvm

9.1. Proceso

9.1.1. Instalación de epel-release

```
sudo yum install epel-release
```

```
[vagrant@minikubelocal ~]$ cat /proc/cpuinfo | egrep -c "vmx"
3
[vagrant@minikubelocal ~]$ sudo yum install epel-release
Failed to set locale, defaulting to C.UTF-8
CentOS Linux 8 - AppStream                      3.8 MB/s | 8.4 MB     00:02
CentOS Linux 8 - BaseOS                         1.0 MB/s | 4.6 MB     00:04
CentOS Linux 8 - Extras                          24 kB/s |  10 kB     00:00
Extra Packages for Enterprise Linux 8 - x86_64  1.2 MB/s |  11 MB     00:09
Extra Packages for Enterprise Linux Modular 8 - x8 311 kB/s | 979 kB     00:03
Package epel-release-8-13.el8.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[vagrant@minikubelocal ~]$
```

Figura 9.1: epel-release

9.1.2. Descargar binario de acuerdo a la arquitectura del ordenador

```
curl -LO https://storage.googleapis.
    com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

```
[vagrant@minikubelocal ~]$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 67.3M  100 67.3M    0     0  5096k      0  0:00:13  0:00:13 --:--:-- 5503k
[vagrant@minikubelocal ~]$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
[vagrant@minikubelocal ~]$
```

Figura 9.2: epel-release

9.1.3. Ver la RAM de la pc

```
free -h -t
```

```
[vagrant@minikubelocal ~]$ free -h -t
              total        used        free      shared  buff/cache   available
Mem:          2.8Gi       183Mi       2.0Gi        16Mi       572Mi       2.4Gi
Swap:         2.1Gi          0B       2.1Gi
Total:        4.8Gi       183Mi       4.1Gi
```

Figura 9.3: RAM de la pc

9.1.4. Ver la CPU de la pc

Emplear el comando:

```
cat /proc/cpuinfo | egrep -c "processor"
```

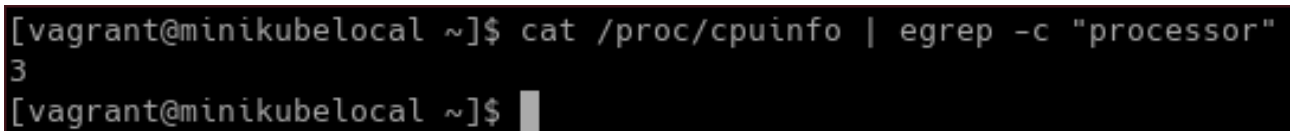


Figura 9.4: CPU de la pc método 1

O con el comando:

```
lscpu | grep "CPU(s):" | head -n 1
```

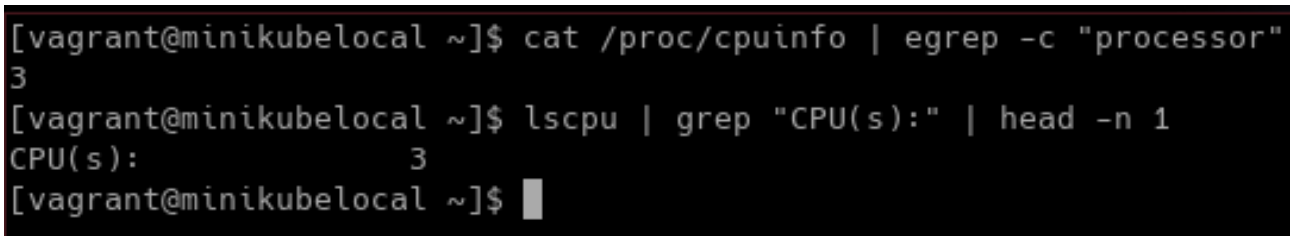


Figura 9.5: CPU de la pc método 2

9.1.5. Instalar qemu-kvm

```
sudo yum -y install libvirt qemu-kvm virt-install virt-top libguestfs-tools
    bridge-utils
```

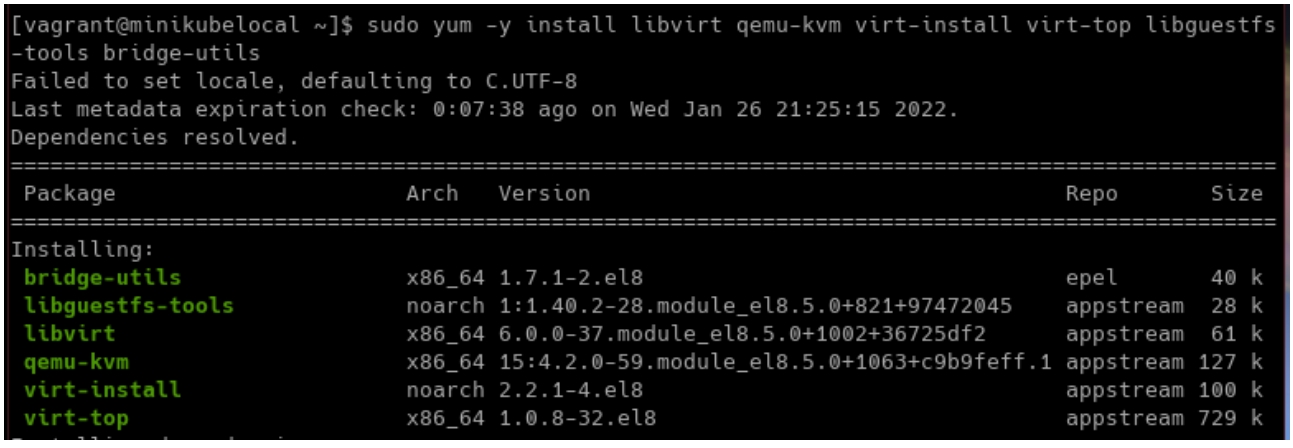


Figura 9.6: Instalar qemu-kvm

9.1.6. Editar el fichero libvirtd.conf

Abrir el fichero:

```
sudo vim /etc/libvirt/libvirtd.conf
```

Agregar o descomentar las sentencias:

```
unix_sock_group = "libvirt"
unix_sock_rw_perms = "0770"
```

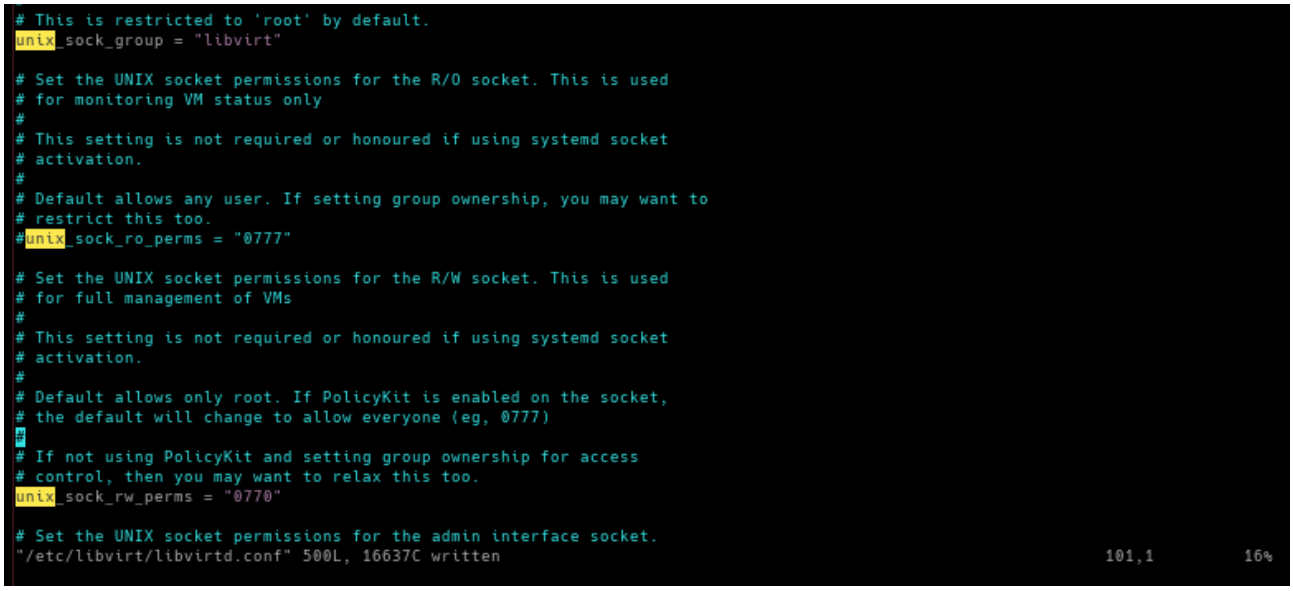


Figura 9.7: Editar el fichero libvirtd.conf

9.1.7. Salir, verificar el estado de libvirt e iniciarlo

```
systemctl status libvirtd.service
systemctl start libvirtd.service
```

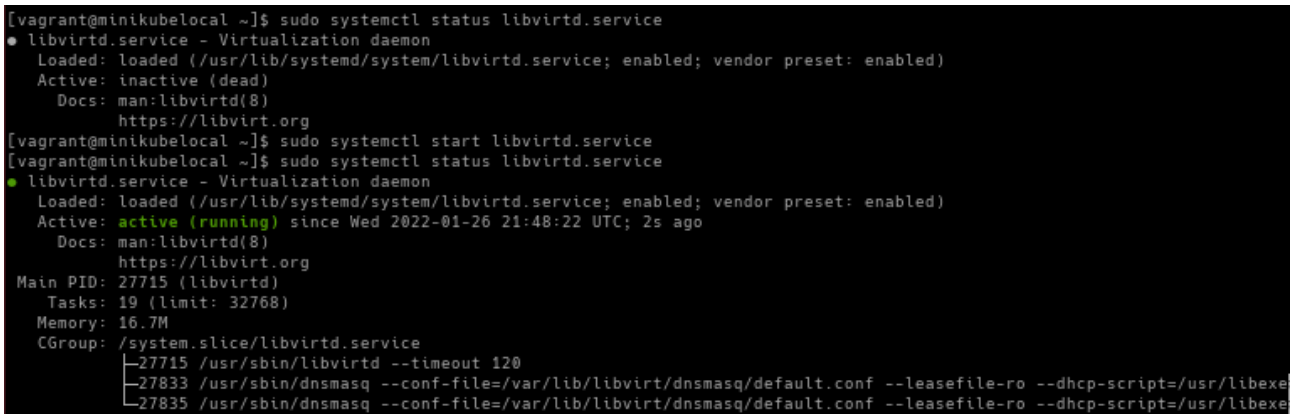


Figura 9.8: Verificar estado e iniciar el servicio de libvirt

9.1.8. Agregar al usuario ("vagrant") al grupo libvirt

```
sudo usermod -a vagrant -G libvirt
```

```
[vagrant@minikubelocal ~]$ sudo usermod -a vagrant -G libvirt
[vagrant@minikubelocal ~]$ group
groupadd   groupdel   groupmems  groupmod   groups
[vagrant@minikubelocal ~]$ groups vagrant
vagrant : vagrant libvirt
[vagrant@minikubelocal ~]$
```

Figura 9.9: Agregar usuario al grupo libvirt

9.1.9. Iniciar minikube

9.1.10. Agregar al usuario ("vagrant") al grupo libvirt

```
minikube start
```

```
[vagrant@minikubelocal ~]$ minikube version
minikube version: v1.25.1
commit: 3e64b11ed75e56e4898ea85f96b2e4af0301f43d
[vagrant@minikubelocal ~]$ minikube start
😄  minikube v1.25.1 on Centos 8.5.2111
✨  Automatically selected the kvm2 driver
💾  Downloading driver docker-machine-driver-kvm2:
    > docker-machine-driver-kvm2....: 65 B / 65 B [----------] 100.00% ? p/s 0s
    > docker-machine-driver-kvm2: 11.61 MiB / 11.61 MiB  100.00% 2.86 MiB p/s 4

❗  The requested memory allocation of 2200MiB does not leave room for system overhead (total system memory: 2824MiB). You may f
ace stability issues.
💡  Suggestion: Start minikube with less memory allocated: 'minikube start --memory=2200mb'

💿  Downloading VM boot image ...
    > minikube-v1.25.0.iso.sha256: 65 B / 65 B [-------------] 100.00% ? p/s 0s
    > minikube-v1.25.0.iso: 226.25 MiB / 226.25 MiB [] 100.00% 4.39 MiB p/s 52s
👍  Starting control plane node minikube in cluster minikube
💾  Downloading Kubernetes v1.23.1 preload ...
    > preloaded-images-k8s-v16-v1...: 504.42 MiB / 504.42 MiB  100.00% 4.53 MiB
🔥  Creating kvm2 VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🐳  Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
```

Figura 9.10: Iniciar minikube

9.1.11. Verificar el funcionamiento de minikube

```
minikube kubectl -- get pods -A
minikube kubectl get nodes
```

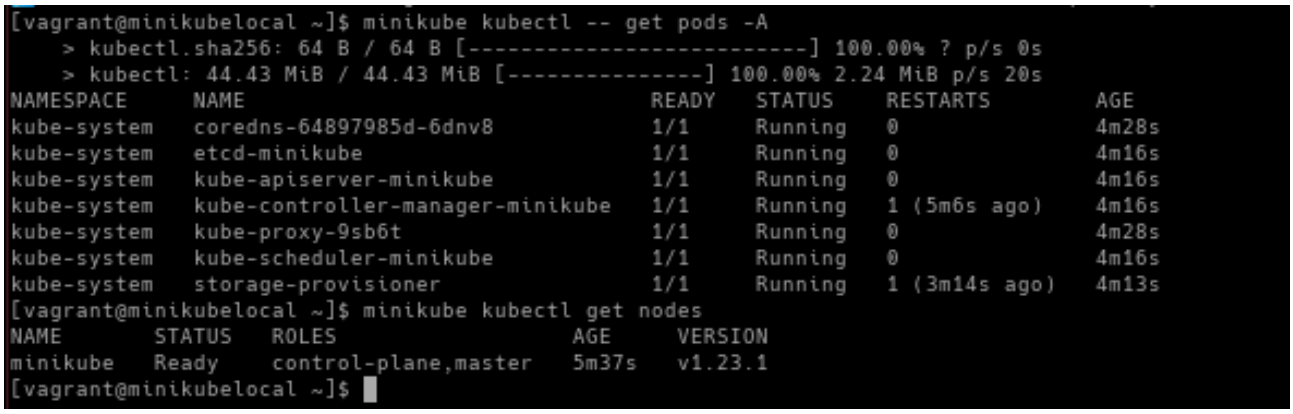


Figura 9.11: Iniciar minikube

9.1.12. Confirmar la lista de máquinas en qemu

```
sudo virsh -c qemu:///system list --all
```

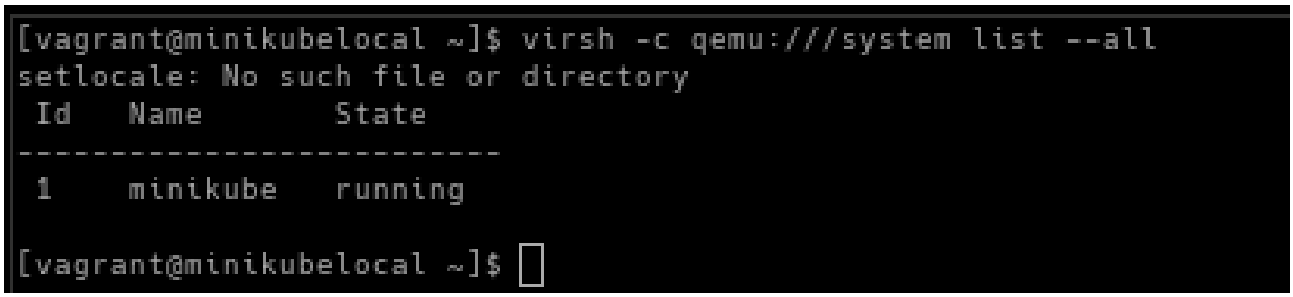


Figura 9.12: minikube en qemu-kvm

9.2. kubectl

Al hacer uso de minikube, kubectl se obtiene como comando nativo de la herramienta, sin embargo, si se desea acceder a diferentes clusters, no unicamente locales es necesario instalar kubectl.

Los pasos a continuación demostrados, provienen directamente de la documentación de kubernetes:

9.2.1. Descargar el binario

```
$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.
    io/release/stable.txt)/bin/linux/amd64/kubectl"
```

```
[vagrant@minikubelocal ~]$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd6
4/kubectl"

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0    956      0 --:--:-- --:--:-- --:--:--   950
100 44.4M  100 44.4M    0     0  1445k      0  0:00:31  0:00:31 --:--:-- 1111k
```

Figura 9.13: Descargar kubectl

9.2.2. Descargar kubectl checksum

```
$ curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)
    /bin/linux/amd64/kubectl.sha256"
```

```
[vagrant@minikubelocal ~]$ curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubect
l.sha256"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0    428      0 --:--:-- --:--:-- --:--:--   428
100    64  100    64    0     0     75      0 --:--:-- --:--:-- --:--:--     0
```

Figura 9.14: Descargar kubectl checksum

9.2.3. Validar el binario de kubectl

```
echo "$(<kubectl.sha256)  kubectl" | sha256sum --check
```

```
[vagrant@minikubelocal ~]$ echo "$(<kubectl.sha256)  kubectl" | sha256sum --check
kubectl: OK
```

Figura 9.15: Validar kubectl

9.2.4. Instalar kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

```
[vagrant@minikubelocal ~]$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Figura 9.16: Instalar kubectl

9.2.5. Verificar la versión de kubectl

```
kubectl version --client
```

Figura 9.17: Instalar kubectl

9.2.6. Enlace de minikube con kubectl

En caso de que kubectl se encuentre enlazado a minikube u otro cluster de kubernetes las sentencias a continuación permitirán obtener de manera general los servicios ejecutándose:

Obtener toda la información del cluster.

```
kubectl get all -o wide
```

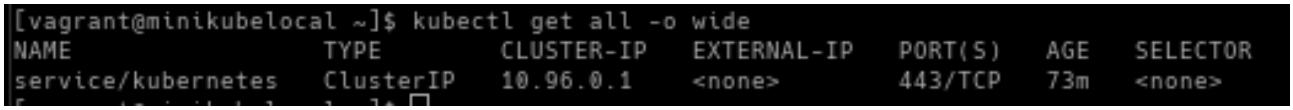


Figura 9.18: Obtener toda la información del cluster

Obtener los nodos en ejecución.

```
kubectl get nodes
```

Figura 9.19: Obtener los nodos del cluster

ANEXO 10

Enlace de okteto cloud con una máquina local

10.1. Creación de una cuenta en okteto cloud

Abrir la página de okteto cloud.

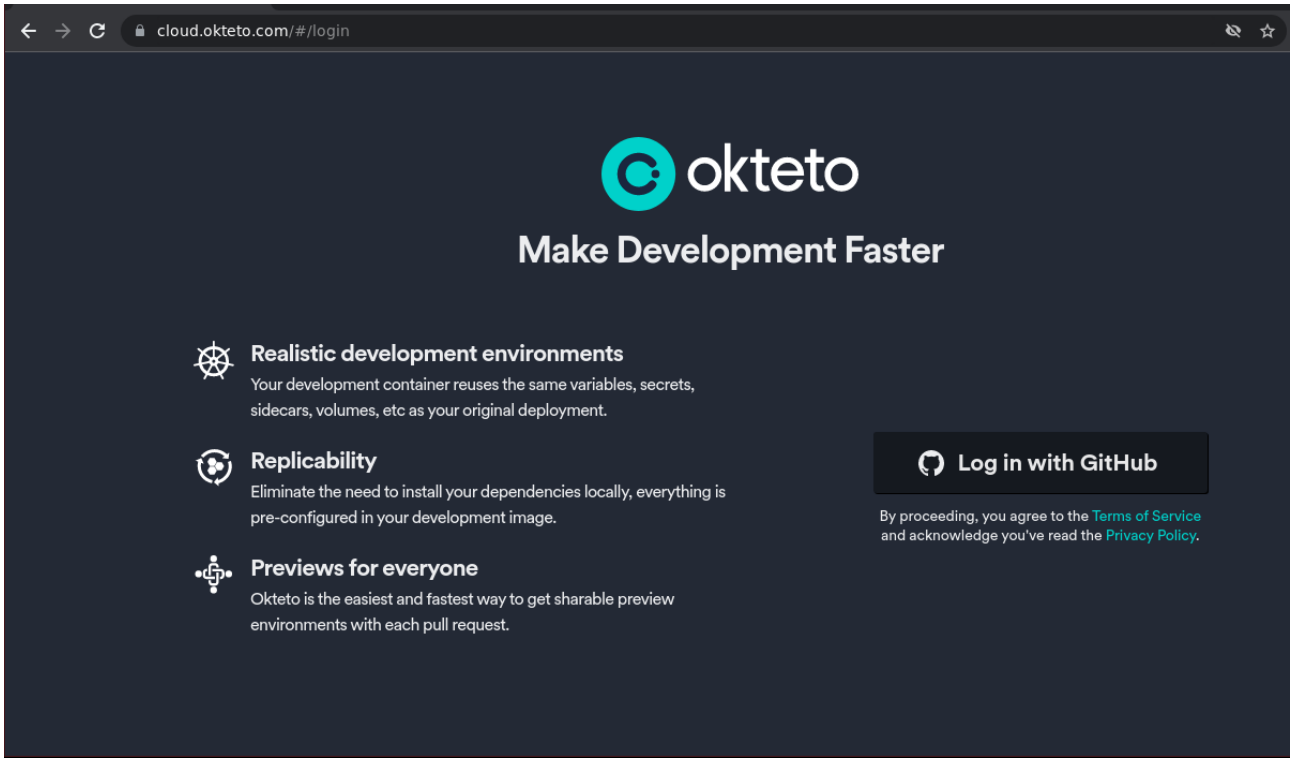


Figura 10.1: Página oficial de okteto cloud

Iniciar sesión con github.

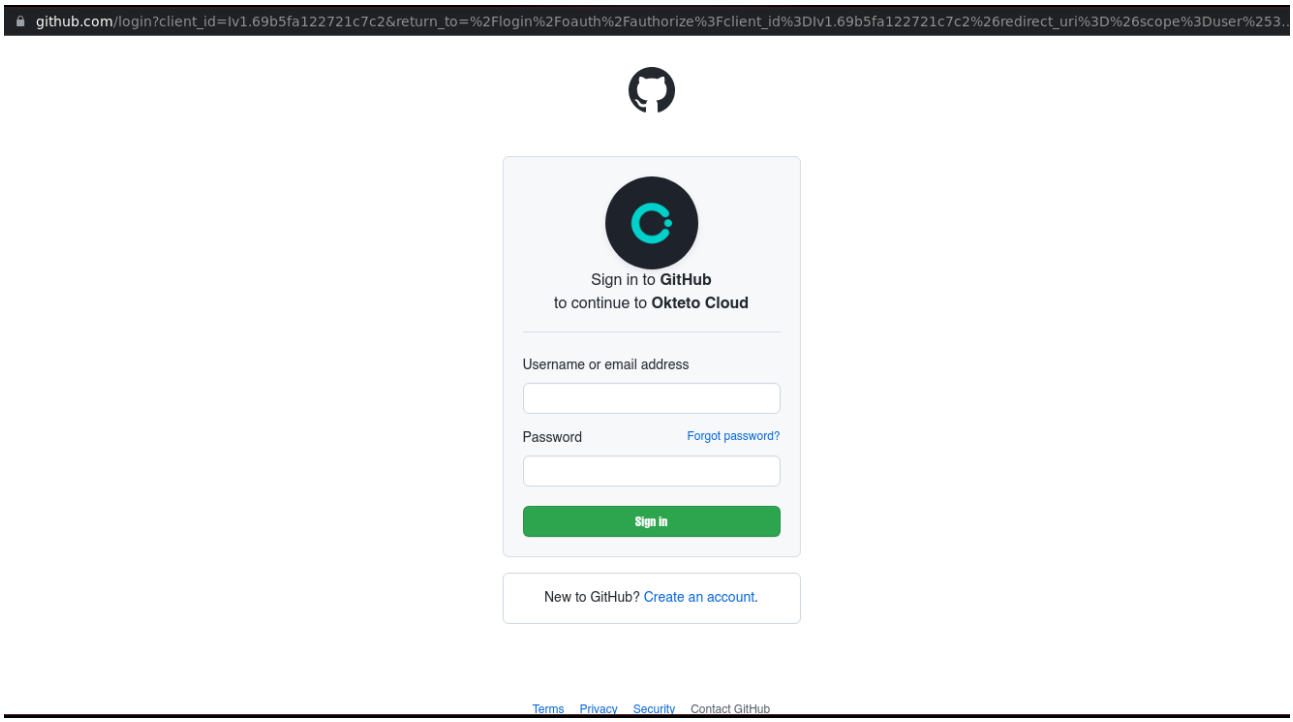


Figura 10.2: Iniciar sesión mediante github

Al acceder a la cuenta, es necesario dirigirse a **"Settigs"** y pulsar **"Download Config File"**.

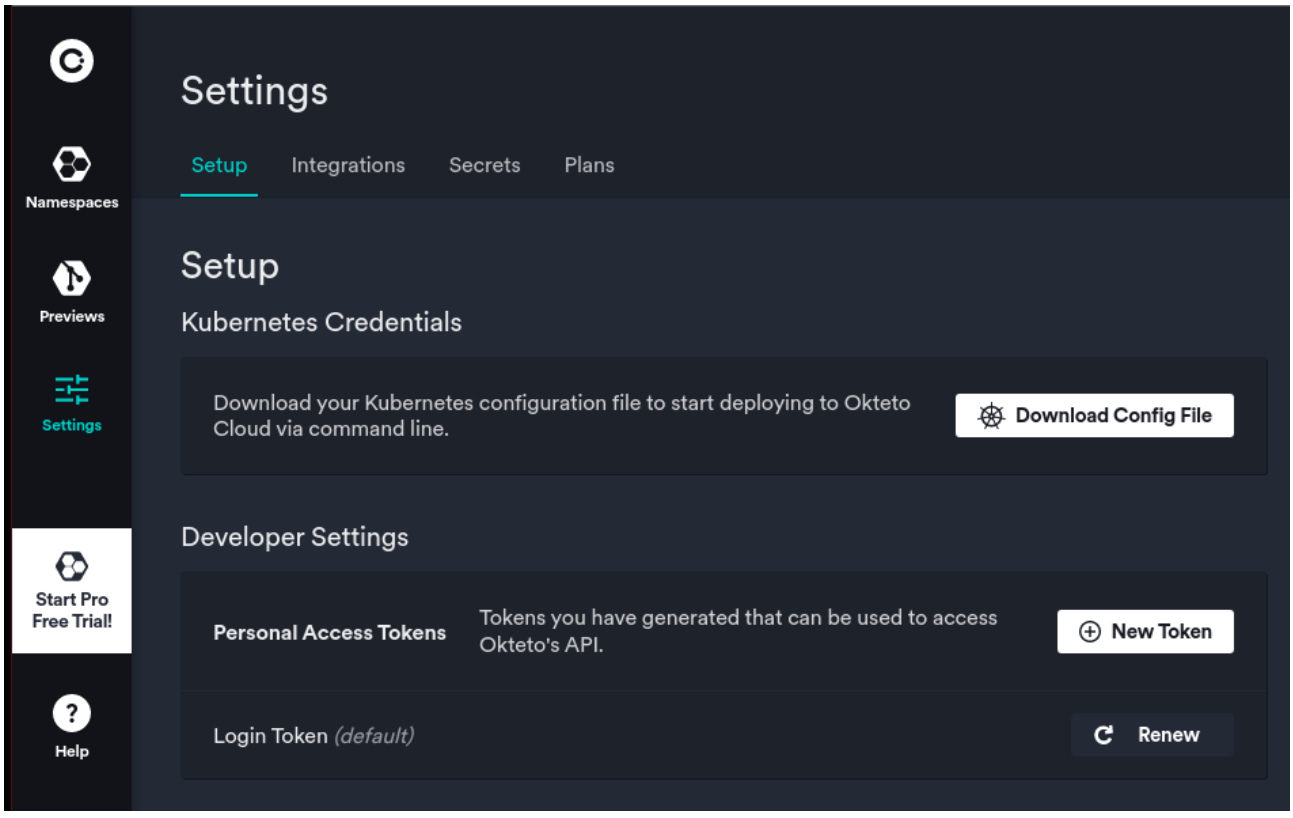


Figura 10.3: Descargar el archivo de configuración

Copiar el comando para enlazar a kubectl con okteto cloud desde la terminal.

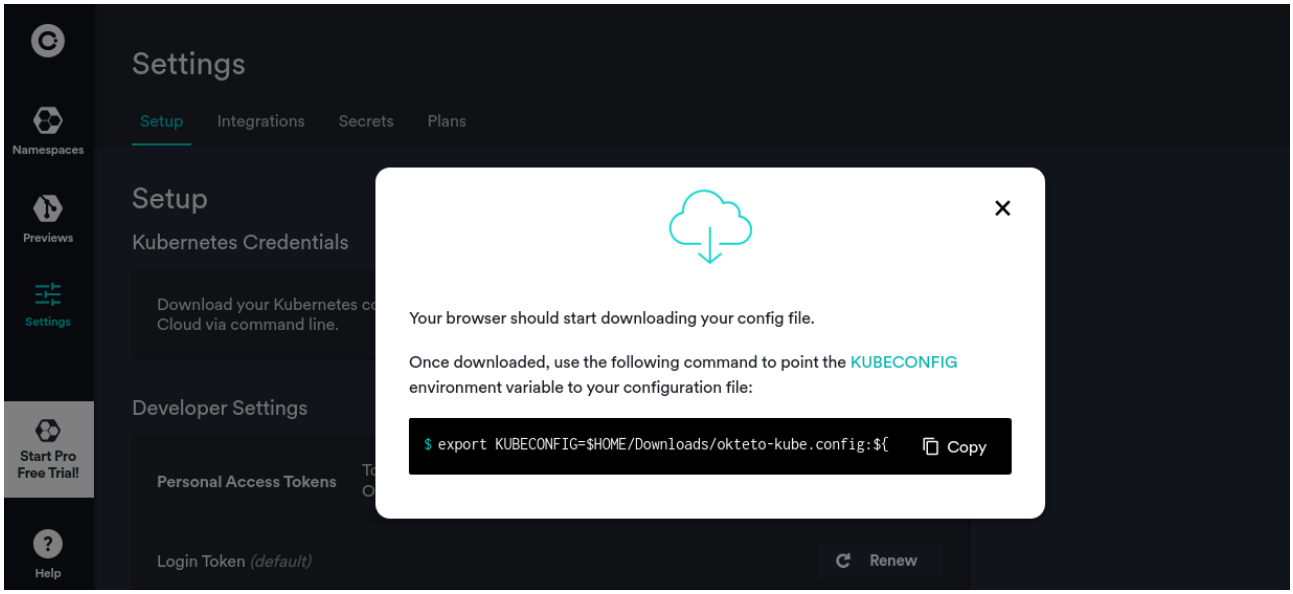


Figura 10.4: Copiar comando de enlace

En la terminal del equipo local mediante kubectl se emplea el comando copiado desde okteto cloud, en el cual unicamente cambia la ruta en la que se encuentra localizado el archivo **okteto-kube.config**, el mismo que para el presente caso se encuentra en el directorio principal.

```
export KUBECONFIG=$HOME/okteto-kube.config:${KUBECONFIG:-$HOME/.kube/config}
```

```
pi@raspberrypi:~ $ kubectl config get-contexts
CURRENT   NAME   CLUSTER   AUTHINFO   NAMESPACE
pi@raspberrypi:~ $ export KUBECONFIG=$HOME/okteto-kube.config:${KUBECONFIG:-$HOME/.kube/config}
```

Figura 10.5: Enlace a okteto cloud

```
pi@raspberrypi:~ $ kubectl config get-contexts
CURRENT   NAME                 CLUSTER
*         cloud_okteto_com     cloud_okteto_com
```

Figura 10.6: Verificación

ANEXO 11

Clúster de Kubernetes con k3s

11.1. Vagrantfile

Para la creación del clúster se dará uso a un archivo Vagrantfile, el cual dispone de:

- El master
- Dos nodos
- ufw para el tratamiento del firewall

El nodo master se incorpora con la finalidad de en caso no se disponga de un servidor externo, el clúster se lo pueda elaborar, por tato se lo puede quitar del Vagrantfile de no ser necesario.

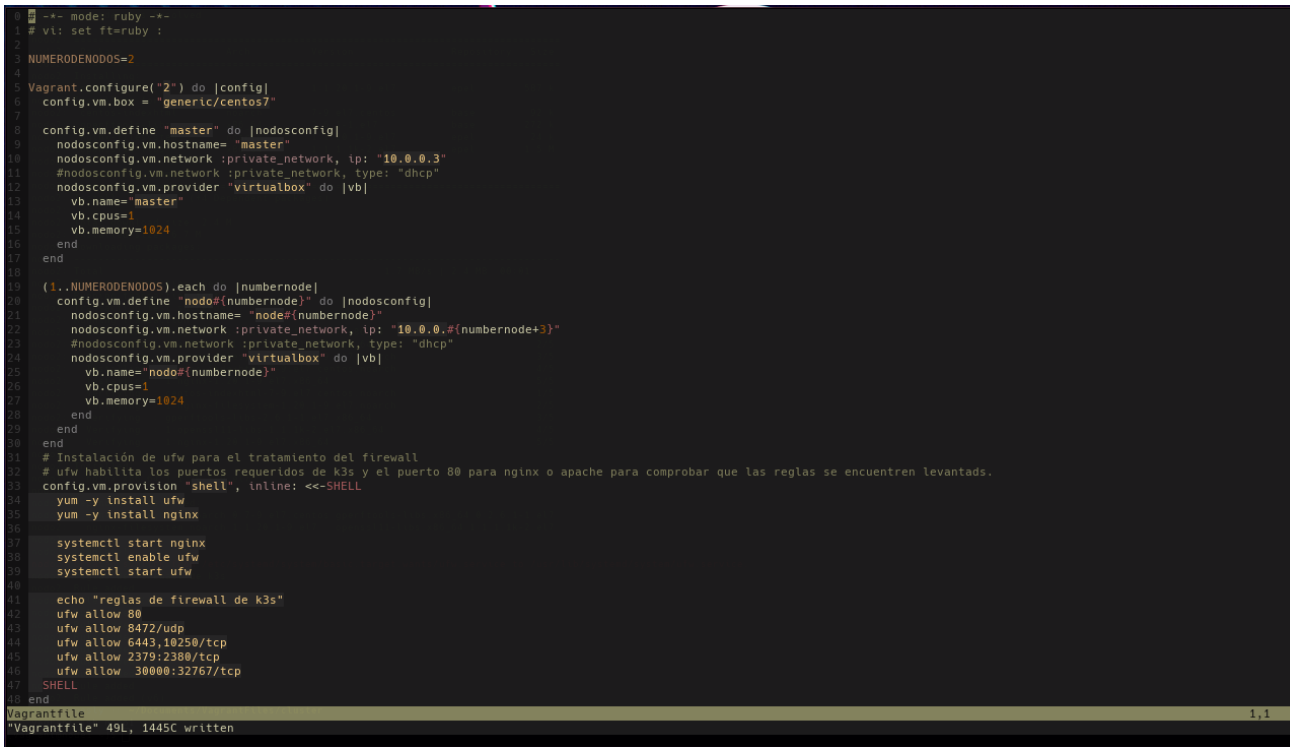


Figura 11.1: Vagrantfile del clúster

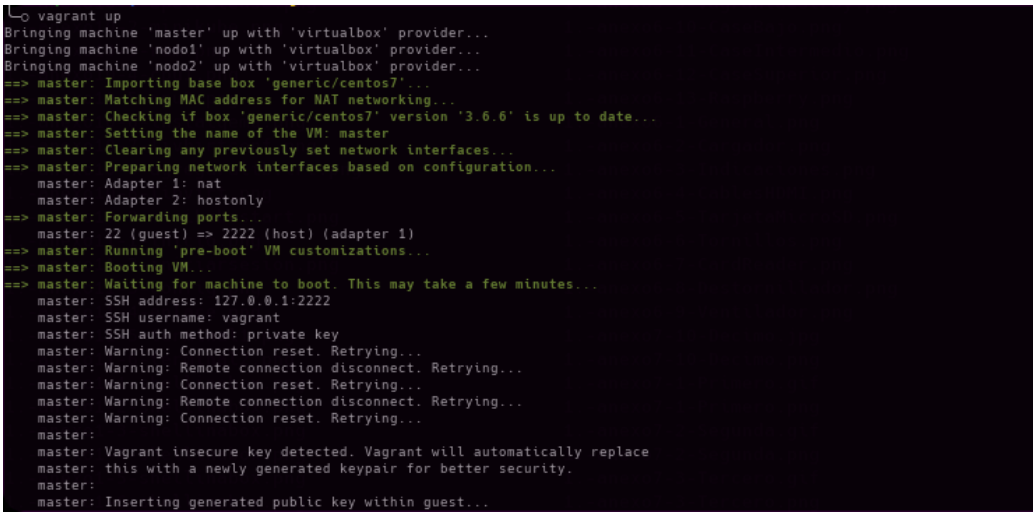


Figura 11.2: Creación de los nodos

11.2. Proceso

11.2.1. Observar el estado de las máquinas levantadas

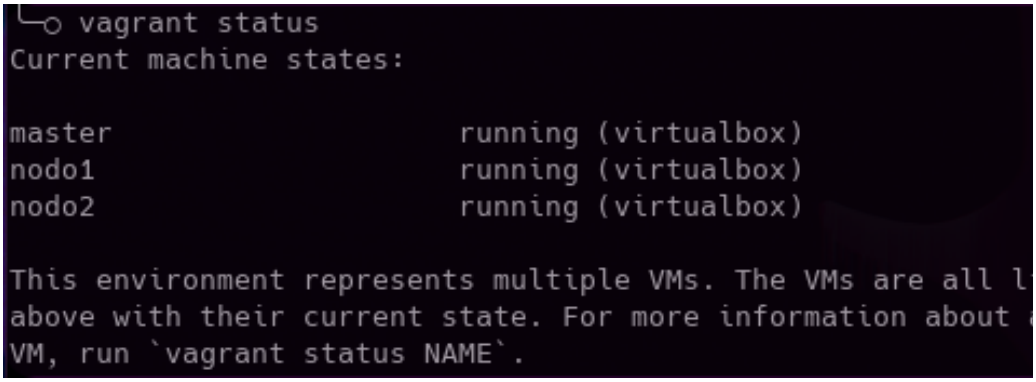


Figura 11.3: Estado nodos del clúster

11.2.2. Desde el Raspberry Pi 4

Editar el archivo /boot/cmdline.txt"

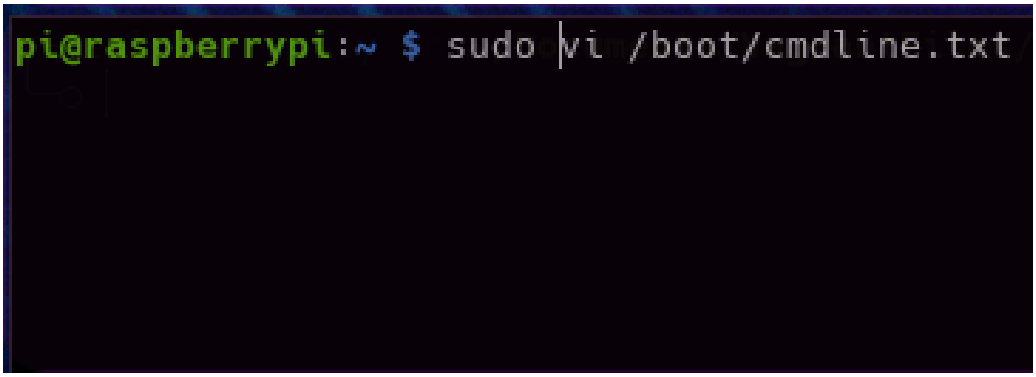


Figura 11.4: Editar archivo /boot/cmdline.txt

Y agregar las líneas:

```
cgroup_memory=1 cgroup_enable=memory
```

Figura 11.5: Líneas adicionadas

Posteriormente se reinicia el sistema, y se preocede a instalar k3s.

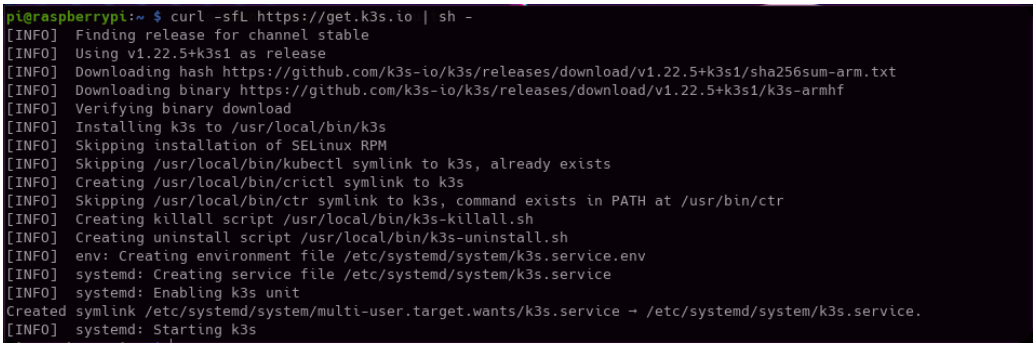


Figura 11.6: Instalación de k3s

11.2.3. Obtención del token para crear los nodos

Desde un equipo de la red se ejecuta el siguiente comando, el cual mediante ssh y la ip del Raspberry Pi 4 obtiene el token que se encuentra en el artefacto:

```
ssh 192.168.1.17 -l pi "sudo cat /var/lib/rancher/k3s/server/node-token"
```

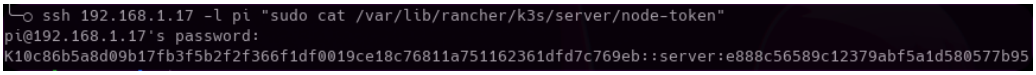


Figura 11.7: Token de k3s

O desde el Raspberry Pi 4 directamente se puede obtener el token con el comando:

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

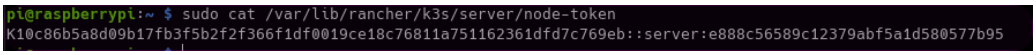


Figura 11.8: Token de k3s desde el Raspberry Pi 4

11.2.4. Desde los nodos

Con el comando:

```
vagrant ssh nodo1
```

Se accede al primer nodo.

En el nodo para enlazarlo al Raspberry Pi 4 se digital el siguiente comando:

```
curl -sfL https://get.k3s.io | K3S_URL=https://192.168.1.17:6443 K3S_TOKEN=
    K10c86b5a8d09b17fb3f5b2f2f366f1df0019ce18c76811a751162361dfd7c769eb::server:
 e888c56589c12379abf5a1d580577b95 sh -
```

Figura 11.9: Nodo1 k3s Raspberry Pi 4

En el cual K3S_URL corresponde a la dirección IP del Raspberry Pi 4 y K3S_TOKEN corresponde al token previamente obtenido.

En el nodo2 el proceso se repite:

Se accede al nodo con el comando:

```
vagrant ssh nodo2
```

Posteriormente se enlaza el nodo con el Raspberry Pi4:

```
curl -sfL https://get.k3s.io | K3S_URL=https://192.168.1.17:6443 K3S_TOKEN=
    K10c86b5a8d09b17fb3f5b2f2f366f1df0019ce18c76811a751162361dfd7c769eb::server:
 e888c56589c12379abf5a1d580577b95 sh -
```

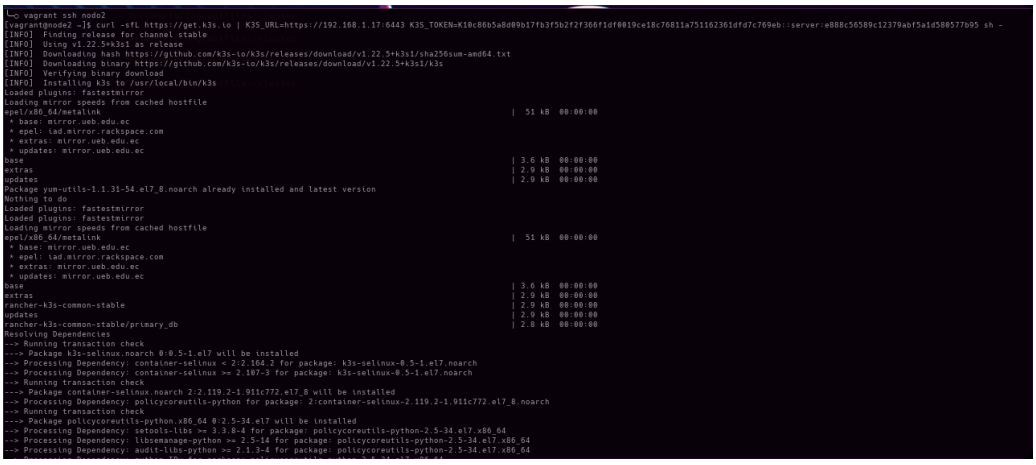


Figura 11.10: Nodo2 k3s Raspberry Pi 4

Y con ello el proceso finaliza.

11.3. Validación

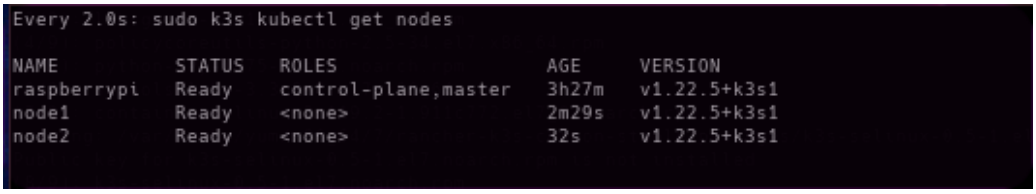


Figura 11.11: Clúster elaborado

ANEXO 12

Control del clúster de Kubernetes k3s externo desde la máquina local con Ubuntu 20.04

12.1. Desde el Raspberry Pi 4

```
$ sudo chmod 644 /etc/rancher/k3s/k3s.yaml
$ cp /etc/rancher/k3s/k3s.yaml .
$ nc -lvp 2222 < k3s.yaml
```

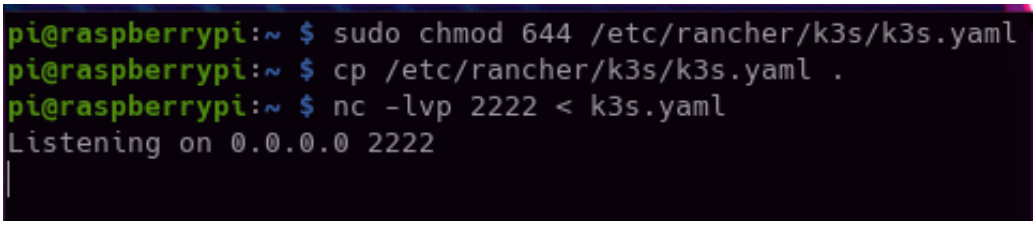


Figura 12.1: Pasos desde el Raspberry Pi 4

12.2. Desde la máquina local de linux

```
$ nc 192.168.1.17 2222 > k3s.yaml
$ cp k3s.yaml ~/.kube/config
$ vim ~/.kube/config
```

Figura 12.2: Recibir por netcat el fichero k3s.yaml

Figura 12.3: Copiar el fichero k3s.yaml al ~/.*kube*/*config*

Figura 12.4: Editar el fichero ~/.*kube*/*config*

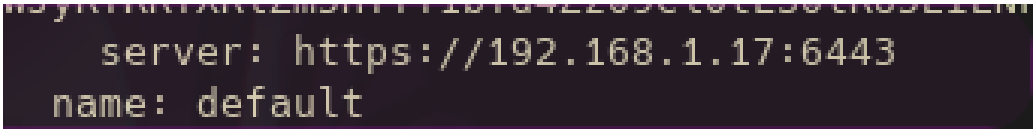


Figura 12.5: Agregar la IP del Raspberry Pi 4 a la sección server del fichero ~/.*kube*/*config*

12.3. Validación

```
kubectl get nodes
```

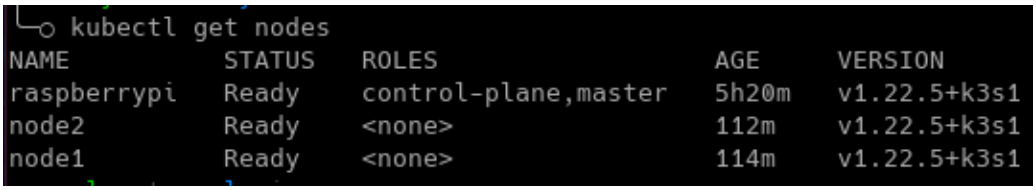


Figura 12.6: Obtención de los nodos del clúster desde la máquina local