ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UNA PLATAFORMA DE MONITOREO DE INSTANCIAS ODOO, UTILIZANDO LA HERRAMIENTA ZABBIX EN INFRAESTRUCTURA EN LA NUBE

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

RICARDO DANIEL RANGLES CÓRDOVA

DIRECTOR: ING. PABLO WILIAN HIDALGO LASCANO, M.Sc.

Quito, diciembre 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Ricardo Daniel Rangles Córdova, bajo mi supervisión.

PABLO WILIAN HIDALGO LASCANO
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Ricardo Daniel Rangles Córdova, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_____

RICARDO DANIEL RANGLES CÓRDOVA

DEDICATORIA

Este trabajo va dedicado a mi Dios por ser el que me ha puesto en este lugar para poder terminar mi carrera, a toda mi Familia por el inagotable apoyo y sobre todo el amor que me han brindado siempre, todos mis éxitos son y siempre serán para ellos.

Ricardo Daniel Rangles Córdova

AGRADECIMIENTO

Primero agradezco mucho a mis Padres Papu y Gigi por su apoyo incondicional y por ser un gran ejemplo de vida que espero poder llegar a tener algún día, les debo mi vida entera y todos mis triunfos serán siempre para ellos.

A mis hermanos Pato y Santy por siempre apoyarme, por su amor incondicional, por ser mis maestros, por aconsejarme y darme muchos ánimos para realizar todos los objetivos que quiero cumplir en mi vida, les agradezco por su tiempo, su paciencia y sobre todo por enseñarme cómo se debe vivir.

A mis sobrinos Nicole y Santy, por cada locura que hemos compartido, por el cariño y amor que me brindan cada vez que pasamos juntos.

A mi esposa Katy y a mi hija Pelusa por su amor incondicional que desde el día que Dios nos unió fue lo más hermoso que me ha pasado en la vida, ellas son mi inspiración para cada día ser un mejor hombre y todo mi amor y éxitos futuros serán siempre para ellas.

A mi primo Bid y mis cuñadas Daniela y Lupita que me han ayudado y me han apoyado para poder culminar mi carrera.

A mis abuelitos Papá Carlitos, Mamá Rosita y a mi hermano Eddy que desde el cielo estoy seguro están apoyándome y dándome fuerzas para seguir adelante en todo lo que me proponga en mi vida.

A mis tíos Ñaño Hernán y Ñaña Ceci, Ñaño Héctor y Ñaña Mariani por su apoyo y por ser parte de mi camino para conseguir mi objetivo de culminar mis estudios.

A mi director de Tesis M.Sc. Pablo Hidalgo, por apoyarme desde el primer día que conversamos, por su tiempo y sobre todo su guía para poder culminar este proyecto.

A mi amigo Andrés Calle por su ayuda, apoyo y su aprobación para poder realizar este proyecto con éxito.

A todos mis familiares y a mis amigos que me apoyaron de una u otra manera para finalizar con éxito este proyecto y esta etapa de mi vida.

A todos ellos dedico mi proyecto y les estaré eternamente agradecido. Los amo a todos.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

En la actualidad existen empresas que llevan varios años trabajando y brindando ayuda a sus clientes en la gestión de sus negocios mediante la plataforma Odoo; para que dicha ayuda sea eficiente se requiere conocer a mayor detalle el estado de los servidores y las instancias existentes que brindan a cada cliente.

Se propone el desarrollo de una plataforma Zabbix para monitorear con mayor número de métricas cada servidor y tomar decisiones para mejorar el área de infraestructura actual de las empresas; se requiere también controlar el monitoreo de las instancias cuando se cree y elimine el enlace de monitoreo de una instancia. Parte del monitoreo permitirá verificar la disponibilidad de cada instancia, para que de existir inconvenientes, se recupere la disponibilidad del cliente y se tomen acciones preventivas y/o correctivas.

En el Capítulo I se explica el marco teórico para el desarrollo de todo el sistema de monitoreo. En el Capítulo II se expone el diseño y la implementación del sistema de monitoreo. En el Capítulo III se indican las pruebas y resultados del funcionamiento del sistema de monitoreo. En el Capítulo IV se presentan las conclusiones y recomendaciones obtenidas en el desarrollo del proyecto.

En los Anexos se detalla la entrevista a los gerentes de la empresa, usuarios y contraseñas de para el uso de Zabbix, los códigos del proyecto y el manual de instrucciones y configuración del proyecto en general.

PALABRAS CLAVE: Plataforma de monitoreo, Módulo de interconexión, Odoo, Zabbix, instancia, disponibilidad.

ABSTRACT

Currently there are companies that have been working for several years and providing help to their clients in the management of their businesses through the Odoo platform; In order for this help to be efficient, it's necessary to know in greater detail the status of the servers and the existing instances that they provide to each client.

The development of a Zabbix platform is proposed to monitor each server with a greater number of metrics and make decisions to improve the current infrastructure area of the companies; it's also required to control the monitoring of the instances when creating and removing the monitoring link of an instance. Part of the monitoring will allow to verify the availability of each instance, so that there are problems, the client's availability is recovered and preventive and / or corrective actions are taken.

Chapter I explains the theoretical framework for the development of the entire monitoring system. Chapter II describes the design and implementation of the monitoring system. In Chapter III the tests and results of the operation of the monitoring system are indicated. In Chapter IV the conclusions and recommendations obtained in the development of the project are presented.

The Annexes detail the interview with the company managers, users and passwords for the use of Zabbix, the project codes and the instructions manual and project configuration in general.

KEY WORDS: Monitoring platform, Interconnection module, Odoo, Zabbix, instance, availability.

CAPÍTULO I

CAPÍTULO I

1. INTRODUCCIÓN

En este capítulo se presenta una recopilación de la información necesaria para llevar a cabo el desarrollo de la plataforma de monitoreo Zabbix y del módulo de interconexión Zabbix - Odoo. Además, se exponen objetivos del proyecto, el alcance que tiene, detallando todos los componentes y funcionalidades del mismo, con el fin de entender cómo se desarrolló el proyecto.

En los siguientes subtemas, se exponen los conceptos acerca de las herramientas y el software utilizados.

1.1. OBJETIVOS

El objetivo general de este Trabajo de Titulación es:

- Desarrollar una plataforma de monitoreo de instancias Odoo, utilizando la herramienta Zabbix en infraestructura en la nube.

Los objetivos específicos de este Trabajo de Titulación son:

- Analizar las herramientas y conceptos teóricos a utilizar en el desarrollo de la plataforma de monitoreo.

- Diseñar todo el software necesario para dar solución al monitoreo simple que tienen las empresas que trabajan con Odoo.

- Implementar todas las funcionalidades diseñadas previamente.

- Analizar la funcionalidad de la plataforma en base a las pruebas realizadas.

1.2. ALCANCE

Este proyecto desarrolla una plataforma de monitoreo completo mediante la herramienta Zabbix, trabajando con un sistema de gestión de empresas llamado Odoo (*On Demand Open Object*); para ello se investiga el monitoreo de servidores en la nube, conceptos, requerimientos y configuraciones.

Se investiga cómo detectar la disponibilidad vía web de cada instancia contenerizada de Odoo, para brindar un mejor servicio a los clientes; además se investiga cómo diseñar y programar un módulo en la plataforma Odoo.

La idea de implementar un sistema de monitoreo basado en Zabbix se debe a la necesidad de la empresa TRESCLOUD CIA. LTDA. de conocer con mayor detalle el estado de sus servidores en la nube de manera rápida y sencilla, mediante un tablero en donde se indiquen las métricas identificadas como necesarias, con el fin de conocer el estado y uso de recursos de los servidores que tienen instancias contenerizadas de Odoo.

Bajo esta necesidad, la metodología será mediante la investigación, diseño, implementación y pruebas con el fin de delimitar las métricas requeridas y las formas de presentación de la información, así como alarmas y notificaciones estándar; una vez obtenida esta información se procederá a realizar la implementación de la plataforma de monitoreo, dejando en la empresa un sistema de monitoreo utilizable y escalable.

Cabe recalcar que se tendrá un módulo de interconexión entre ambas plataformas que permitirá la creación y eliminación de clientes a ser monitoreados. Además, el módulo de interconexión tendrá la funcionalidad de avisar cuando una instancia Odoo cayó o tuvo problemas durante su monitoreo en Zabbix.

1.3. MARCO TEÓRICO

En esta sección se explican las herramientas y tecnologías software utilizadas para el diseño y desarrollo, tanto de la plataforma de monitoreo como del módulo de interconexión Odoo - Zabbix.

1.3.1. ADMINISTRACIÓN DE RED

Administrar redes radica en la organización, la toma de decisiones, control y sobre todo la supervisión de una red, con el fin de mantener un funcionamiento correcto, todo con la ayuda de herramientas de red, aplicaciones y dispositivos [1].

1.3.1.1. Objetivos de la Administración de Red

Los objetivos que realiza la administración de una red son los siguientes [1]:

1. Otorgar las herramientas tanto manuales como automáticas sobre la administración de red, con el fin de controlar la misma en caso de fallas.
2. Establecer tácticas de administración para mejorar la infraestructura existente, es decir en la mejora del rendimiento tanto de los servicios como de las aplicaciones.
3. Tener conocimiento sobre un posible crecimiento de la red, debido a la evolución de la tecnología.

1.3.2. MONITOREO DE RED

Un administrador de red tiene varias tareas dentro de una red, pero uno de los roles fundamentales es sin duda monitorear permanentemente los servicios en una red, en busca de fallas o problemas que puedan tener, con el fin de notificar a los administradores de la red con alarmas mediante correo electrónico o mensajes de texto [1], [2], [3].

La funcionalidad de una persona que monitorea una red consiste en tomar datos de consumo de recursos, memoria, rendimiento, estados de conectividad, el tráfico tanto de entrada como de salida y otros más. El objetivo final es que los problemas que se encuentren se solucionen de manera rápida y efectiva sin que el cliente lo perciba [3].

En la Tabla 1.1 se mencionan los elementos que tiene un sistema de monitoreo.

Tabla 1.1. Elementos de un sistema de monitoreo [4]

Elemento	Descripción
La Estación de Gestión (NMS)	Es la interfaz entre el administrador de red, el usuario y el sistema de gestión; tiene una base de datos con la información de las bases de datos de todas las entidades que son gestionadas.
Agente	Es un elemento activo que responde a todas las solicitudes desde la NMS, entregando de manera sincrónica o asincrónica información importante y no solicitada. El agente se encuentra localizado en cada dispositivo gestionado.
Base de Información (MIB)	Se presentan como una recolección de objetos que permiten gestionar los recursos de la red.

1.3.2.1. Objetivos del Monitoreo de Red

Los objetivos del monitoreo de red son los siguientes [2]:

- Guardar toda la información de las bases de información de la gestión para su futuro análisis.
- Determinar la información que va a ser monitoreada.
- Obtener conclusiones para la resolución de problemas específicos o para la mejora de la utilización de la red.

Los eventos típicos que suelen ser monitoreados con frecuencia son [2]:

- Errores en el inicio de aplicaciones.
- Aviso de arranque de algunas aplicaciones.
- Registro de entradas y salidas de usuarios en la red.
- Registro del estado de finalización de los procesos que son ejecutados.

1.3.2.2. Tipos del Monitoreo de Red

Los tipos de monitoreo son los siguientes:

- ➢ Monitoreo Activo
- ➢ Monitoreo Pasivo

En la Tabla 1.2 se menciona la definición de cada tipo de monitoreo.

Tabla 1.2. Tipos de Monitoreo [5], [6]

Tipo de Monitoreo	Definición
Monitoreo Activo	Funciona enviando paquetes de prueba en toda la red o también enviando paquetes de algunas aplicaciones para medir los tiempos de respuesta. Este tipo de monitoreo agrega tráfico a la red debido al constante envío de paquetes; se utiliza para medir el rendimiento en una red como forma habitual.
Monitoreo Pasivo	Se basa en la recolección y análisis del tráfico que circula en la red. Para ello se utilizan herramientas como ruteadores, *sniffers* y máquinas con todo el software necesario para el análisis del tráfico. Este tipo de monitoreo no agrega tráfico en la red como el monitoreo activo y suele ser más utilizado para determinar todo el tráfico de la red y medir su uso.

1.3.3. VIRTUALIZACIÓN

En la actualidad la virtualización de máquinas abarca varios campos de estudio, pero de forma general se refiere a la abstracción de los recursos de una computadora [10].

La virtualización básicamente consiste en crear a través de software algún recurso tecnológico, con el fin de que este software tenga la función de simular la existencia de este recurso tecnológico [11].

1.3.3.1. Tipos de Virtualización

En la Tabla 1.3 se mencionan los tipos de virtualización.

Tabla 1.3. Tipos de Virtualización [12]

Tipos de Virtualización	Definición
Virtualización de Plataformas	Se aplica para consolidación de servidores, que no es más que particionar un servidor físico de manera que albergue varios servidores dedicados que ejecutan su propio sistema operativo (S.O) y también sus servicios.
Virtualización de Recursos	En esta virtualización el recurso que se obtiene es individual de una máquina, puede ser la conexión de red o el almacenamiento principal o secundario.
Virtualización de Aplicaciones	Es un proceso que hace creer a una aplicación que está interactuando directamente con un sistema operativo, pero en realidad interactúa con una capa de virtualización insertada entre la aplicación y el sistema operativo.
Virtualización de Escritorio	En esta virtualización el escritorio del usuario se manipula de forma remota, este se encuentra fuera de la máquina física.

1.3.3.2. Ventajas y Desventajas de la Virtualización

En la Tabla 1.4 se detallan algunas ventajas y desventajas de la virtualización.

Tabla 1.4. Ventajas y Desventajas de la Virtualización [12]

Ventajas	Desventajas
Reducción de los costos de espacio	Falta de visibilidad en la solución de problemas en entornos virtuales.
Reducción de costos de hardware	Problemas con los administradores de virtualización.
Aislamiento en caso de fallos generales	Costos de despliegue demasiado altos.

1.3.4. VIRTUALBOX

VirtualBox es una herramienta potente de virtualización de máquinas, tanto para uso empresarial como doméstico; además, no solo es una herramienta que ofrece varias funcionalidades de alto rendimiento para clientes empresariales, sino que también es la única solución profesional que está disponible gratuitamente como software de código abierto bajo los términos de la GNU *General Public License* (GPL) versión 2 [14].

Actualmente VirtualBox se ejecuta en *hosts* como Windows, Linux y Solaris; y es compatible con una gran cantidad de sistemas operativos como Windows (Vista, Windows 7, Windows 8, Windows 10), Linux (2.4, 2.6, 4.x), y Solaris.

VirtualBox es un esfuerzo comunitario respaldado por una empresa dedicada, mientras que Oracle garantiza que el producto siempre cumpla con criterios de calidad profesional [14].

En la Figura 1.1 se presenta la interfaz gráfica de VirtualBox con sus componentes.

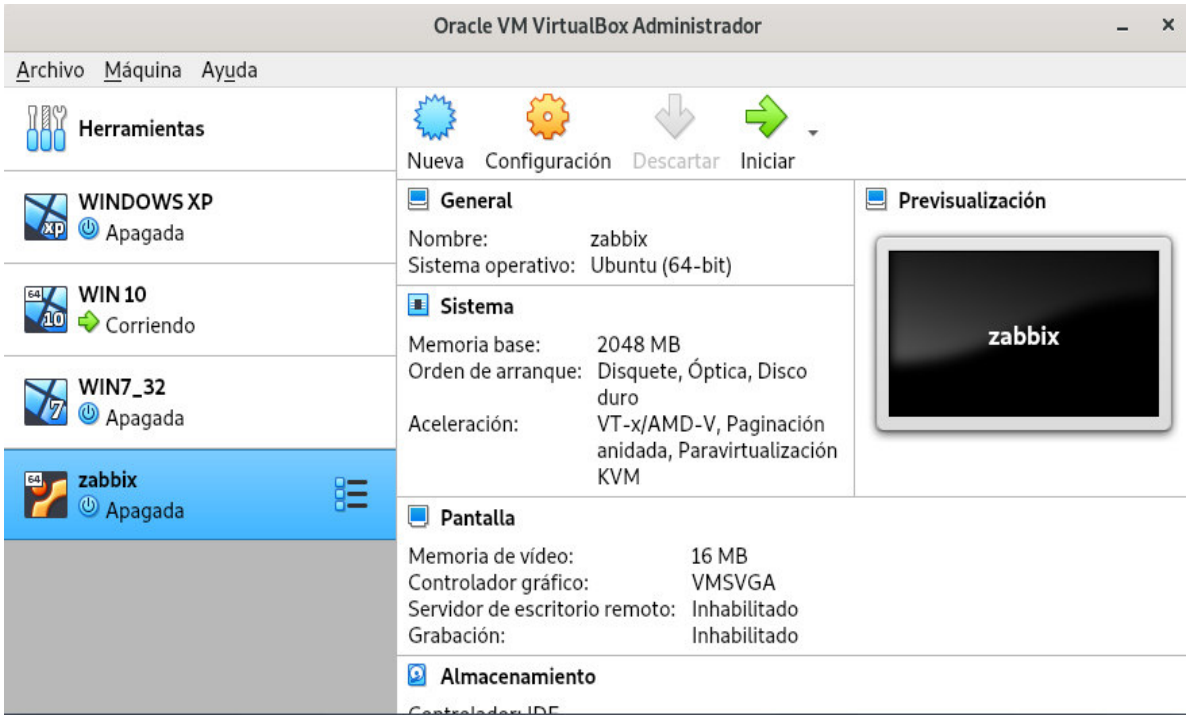


Figura 1.1. Interfaz gráfica de VirtualBox [14]

1.3.5. DOCKER

Docker es una plataforma de virtualización que trabaja a nivel del sistema operativo, la cual permite crear una aplicación y la empaqueta con todas sus dependencias y librerías dentro de un contenedor; al contenerizar este será capaz de ser ejecutado en cualquier otra máquina que tenga una capa de gestión para este tipo de contenedores.

Una de las ventajas principales es que puede ejecutar un desarrollo en cualquier máquina, siempre y cuando tenga las herramientas necesarias para utilizarlo, además que es muy potente y es de código abierto. Cuando se trabaja con Docker no es necesario realizar pruebas en cada entorno en donde se vaya a utilizar la aplicación y tampoco se necesitan hacer ajustes o reconfiguraciones [55], [57].

En la Figura 1.2 se presenta la arquitectura de Docker con todos sus elementos principales.

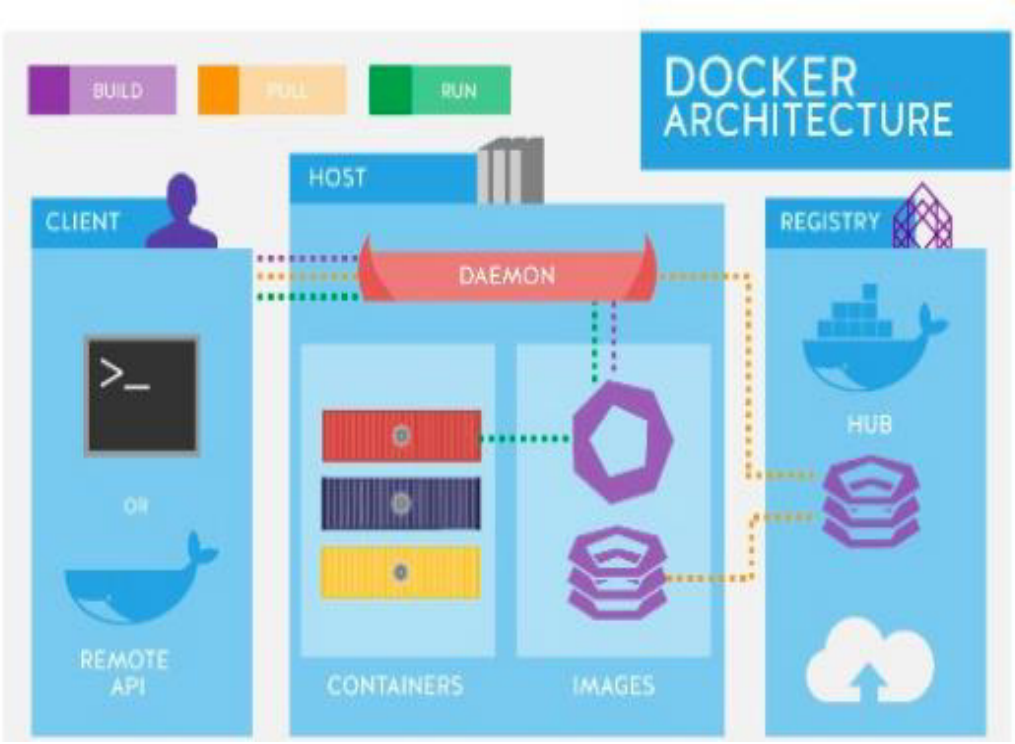


Figura 1.2. Arquitectura de Docker [56]

1.3.5.1. Componentes de Docker

En la Tabla 1.5 se mencionan los componentes principales de Docker [55].

Tabla 1.5. Componentes de Docker [55]

Componentes	Descripción
Docker Daemon	También llamado Docker Engine, es una capa que se encuentra entre el contenedor y el *kernel* de Linux. Es el entorno de tiempo de ejecución persistente que administra todos los contenedores de las aplicaciones y es independiente del S.O.
Dockerfile	Es un documento que se utiliza para crear las imágenes del contenedor. Es un documento de texto que almacena toda la configuración y los comandos necesarios para poder crear una imagen del contenedor.
Interfaz por línea de comandos	Tiene una amplia cantidad de órdenes directas para utilizar la herramienta Docker.

1.3.5.2. Docker Compose

Es una herramienta que hace más fácil el uso de Docker, ya que gracias a Docker Compose se pueden crear distintos contenedores y en cada uno de ellos crear diferentes servicios. Algunos de los comandos que más se utilizan son [58]:

- version: Indica la versión de *docker-compose.*
- build: Comando que construye las imágenes de los servicios que son indicados en el *docker-compose file.*
- up: Crea y levanta los servicios indicados en el *docker-compose file*.
- down: Detiene todos los servicios indicados en el *docker-compose file*, destruyendo los contenedores.
- stop: Solo detiene los servicios y no elimina los contenedores.
- start: Levanta los servicios pero no los crea.
- restart: Reinicia todos los servicios.

1.3.6. VISUAL STUDIO CODE

Visual Studio Code (VSC) es una herramienta poderosa para el desarrollo de software, gracias a la simplicidad de un editor de código fuente, así como a la finalización y depuración de código denominado IntelliSense, la cual tiene como característica importante el completar el código e incluir una lista de características como palabra completa, información de parámetros, lista de miembros e información rápida [15], [16].

Para el desarrollador de software estas características ayudan a la obtención de más información sobre el código que se está utilizando, también a agregar llamadas de propiedades y métodos con teclas especiales y así como realizar un seguimiento de los parámetros que se está escribiendo.

Esta herramienta es compatible con Windows, Linux y MacOS; además incluye soporte para JavaScript y Node.js. Este editor de código admite gran número de extensiones para lenguajes de programación como C++, Java, XML, Python, PHP, C# entre otros. VSC a diferencia de otros editores de código presenta al usuario una interfaz simple e intuitiva [15].

Esta interfaz de usuario consta de cinco áreas fundamentales como se observa en la Figura 1.3 [17].

1. Barra de Actividades (A) - Área que permite cambiar de vistas.
2. Barra Lateral (B) - Área que muestra las carpetas y los archivos que el usuario tiene.
3. Editor de Código (C) - Área donde se realiza la edición de archivos.
4. Paneles (D) - Área donde se muestra la depuración, errores y advertencias.
5. Barra de Estado (E) - Indica la información sobre el proyecto y archivos que están abiertos.

Cuando el usuario utiliza Visual Studio Code y la ejecuta, esta herramienta mantiene el estado que el usuario dejó por última vez.

Más información sobre las herramienta de programación que se utilizó en este proyecto, se la puede obtener ingresando al siguiente enlace: *https://code.visualstudio.com/docs.*

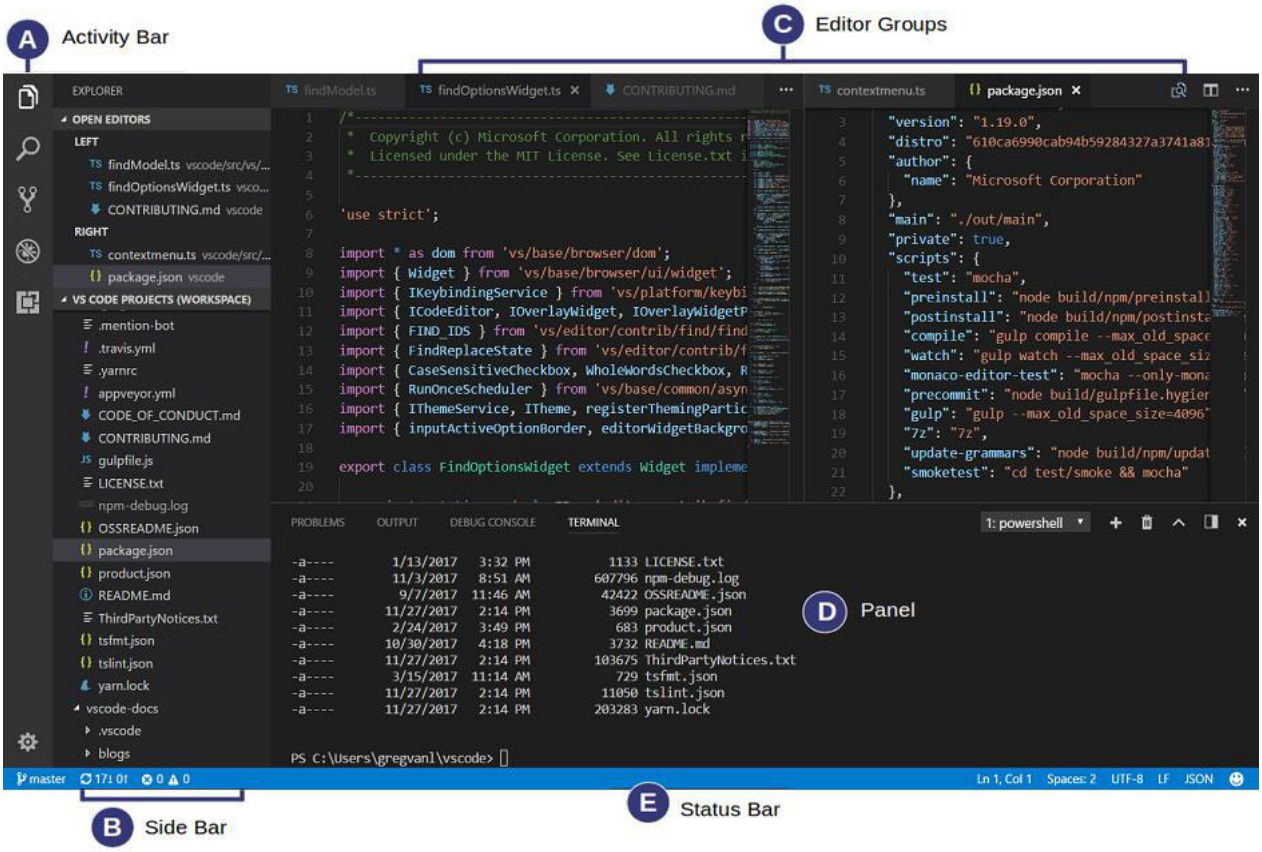


Figura 1.3. Interfaz gráfica de Visual Studio Code [17]

1.3.7. LENGUAJE DE SCRIPTING BASH

Bash es el intérprete del lenguaje de comandos o Shell para el sistema operativo GNU; Bash es el Shell predeterminado. Al igual que otros programas de GNU, Bash es muy portátil. En la actualidad se ejecuta en casi todas las versiones de Unix y en algunos otros sistemas operativos con su soporte independiente. Además, Bash puede ejecutar la mayoría de los *scripts* sh sin modificaciones.

BASH incluye una sintaxis algo compleja, pero fácil de aprender; maneja una serie de órdenes internas que funcionan de forma similar a la línea de comandos. Un programa puede dividirse en secciones cortas, fáciles de depurar, permitiendo realizar prototipos de aplicaciones más complejas.

1.3.7.1. Características principales de Bash

Las principales características de Bash son las siguientes [19]:

1. Edición del historial de mandatos ejecutados.
2. Lenguaje de programación de alto nivel, que incluye varios tipos de variables, valores, funciones entre otras.
3. Usa un “Shell” para controlar el entorno del usuario.
4. Ejecución interactiva de mandatos, aceptando tanto entradas desde teclado o desde ficheros.

1.3.7.2. Edición y ejecución de un script en Bash

Un *script* en Bash es un fichero de texto normal que tiene una serie de bloques de código que se encuentra formado por líneas de comando que son ejecutadas en secuencia; para ello el usuario debe tener los permisos necesarios para modificarlo en el caso de ser necesario [19].

Los argumentos para el intérprete pueden ser de uno o más argumentos opcionales que siguen al nombre del intérprete en la primera línea del archivo de secuencia de comandos, seguido del nombre del archivo de secuencia de comandos y por último el resto de los argumentos proporcionados a la secuencia de comandos.

La mayoría de *scripts* en bash comienzan con #! / bin / bash, ya que esto confirma que Bash se usará para interpretar el *script*, incluso si se ejecuta bajo otra capa [18].

En la Figura 1.4 se muestra un ejemplo de cómo debe comenzar un *script*, es decir con la marca #! para especificar el camino completo y los parámetros del intérprete de mandatos que ejecutará el programa.

```
#!/bin/bash
# ejemplo1: informe de la capacidad de la cuenta

echo "Usuario: $USER"
echo "Capacidad de la cuenta:"
du -hs $HOME            # suma total del directorio del usuario
```

Figura 1.4. Comienzo de un script en Bash [19]

1.3.7.3. Recomendaciones de programación en Bash

Un *script* hecho en Bash puede requerir ser modificado, actualizado o mejorado en un futuro, por lo que para el usuario que lo programa es importante ser precavido y tener en consideración ciertas recomendaciones para programar un *script* en Bash. Estas medidas son las siguientes [19]:

- Deben estar agregados comentarios y ayudas sobre los bloques o comandos que sean importantes, además de tener ayuda para la ejecución del programa.
- El código debe estar bien legible, esto incluye sangrías y espacios para que el código esté bien separado.
- Depurar el código para evitar errores.
- Utilizar funciones y estructuras de programación para evitar código redundante.
- Escribir los nombres de las variables, funciones y programas en una manera coherente; los nombres no deben ser ni muy largos ni muy cortos.

Más información sobre el lenguaje de programación Bash se la pueda obtener ingresando al siguiente enlace: *http://www.informatica.us.es/~ramon/articulos/Programacion-BASH*.

1.3.8. LENGUAJE DE PROGRAMACIÓN PYTHON

El lenguaje de programación Python fue creado por Guido van Rossum a inicios de los 90. Se trata de un lenguaje interpretado o de *script* que tiene un tipado dinámico, orientado a objetos y multiplataforma [20].

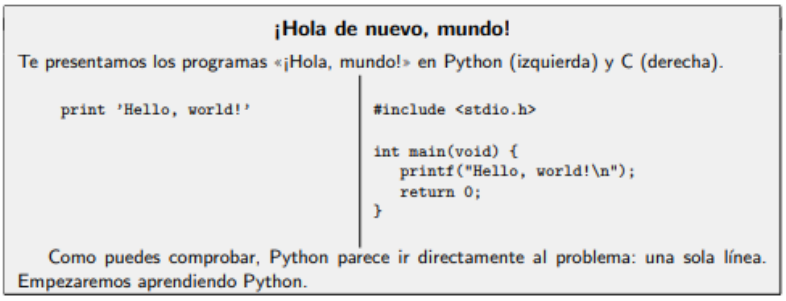
1.3.8.1. Lenguaje interpretado o de Script

Un lenguaje interpretado o de *script* es el que se suele ejecutar utilizando un programa como intermediario llamado intérprete; en vez de compilar el código a lenguaje de máquina, este puede entender y ejecutar directo en la máquina.

Una de las ventajas de los lenguajes compilados es que la ejecución es más dinámica y rápida, pero aun así los lenguajes interpretados son más manejables.

Python se le puede considerar como un lenguaje semi interpretado ya que maneja varias características de lenguajes compilados. Tanto en Python como en Java, el código es traducido a un código fuente llamado *bytecode*; al ejecutar por primera vez éste, se generan los archivos .pyc o. pyo que son ejecutados en varias ocasiones [20].

En la Figura 1.5 se indica un ejemplo de un programa en Python en comparación con un programa en C.



¡Hola de nuevo, mundo!

Te presentamos los programas «¡Hola, mundo!» en Python (izquierda) y C (derecha).

```
print 'Hello, world!'
```

```
#include <stdio.h>

int main(void) {
   printf("Hello, world!\n");
   return 0;
}
```

Como puedes comprobar, Python parece ir directamente al problema: una sola línea. Empezaremos aprendiendo Python.

Figura 1.5. Programa "Hola mundo" en Python comparado al programa en lenguaje C [23]

1.3.8.2. Características principales de Python

En la Tabla 1.6 se mencionan las características principales del lenguaje Python.

Tabla 1.6. Características principales de Python [21], [22]

Características	Definición
Tipado Dinámico	No necesariamente se declara el tipo de dato que va a contener una variable, sino que el tipo se determina en tiempo de ejecución según el tipo del valor al que se asigne.
Multiplataforma	Python está disponible para múltiples plataformas (Linux, Windows, UNIX, Mac OS, Solaris, entre otras), por lo que si no se utilizan las respectivas librerías para cada plataforma el programa que se codifique con Python no podrá ejecutarse.
Orientado a Objetos	Python tiene la capacidad de hacer que los conceptos del mundo real se trasladen a clases y objetos en el programa; la ejecución del programa consiste en una serie de interacciones entre objetos.
Python es de código abierto	Todos los derechos de este programa son reservados por el Instituto Python, pero es de código abierto por lo tanto, no existe limitación en el uso, cambio y distribución del mismo.

En la tabla 1.7 se mencionan todos los métodos que fueron utilizados en Python para el desarrollo del proyecto.

Tabla 1.7. Tipos de métodos [24]

Tipo de Método	Definición
Método de Transformación	Las cadenas son inmutables, por lo tanto todos los métodos que se van a mencionar no actúan sobre el objeto original sino que retornan uno nuevo. Las funciones strip(), lstrip() y rstrip() remueven los espacios en blanco que se preceden o suceden a la cadena.
Método de Separación	El método de separación de una cadena según un carácter separado suele ser el split(), donde este separador por defecto son espacios en blanco y saltos de línea, aunque también puede ser utilizado para separar texto y obtener lo más relevante del mismo.

1.3.8.3. Colecciones

En la tabla 1.8 se mencionan las colecciones que fueron utilizadas en Python para el desarrollo del proyecto.

Tabla 1.8 Tipos de colecciones [20], [25], [26]

Tipo de Colecciones	Definición
Listas	Las listas son un tipo de colección ordenada, equivalentes a los *arrays* o vectores que son conocidos en otros lenguajes de programación.
Tuplas	Las tuplas son secuencias igual que las cadenas y son utilizadas con la misma notación de las cadenas, se diferencian con los [].
Diccionarios	Los diccionarios son estructuras de datos que permite almacenar cualquier tipo de valor entero, cadenas, funciones y listas; son colecciones que relaciona una clave y un valor.

1.3.8.4. Funciones

Una función es un bloque de código con un nombre asociado que recibe desde cero a muchos argumentos como entrada; a éste sigue una secuencia de sentencias, las cuales ejecutan una operación deseada y devuelve un valor o alguna acción. Este bloque de código puede ser llamado cuando se lo necesite [27].

1.3.8.5. Excepciones

Las excepciones son errores que son detectados por Python durante la ejecución del programa. Cuando el programa o intérprete se encuentra con una situación excepcional, como el intentar dividir un número entre 0 o al intentar acceder a un archivo que no existe, se genera un error y lanza una excepción que informa al usuario de que algo sucedió con el programa.

En Python se utiliza una construcción *try-except* para capturar y tratar las excepciones. El bloque *try* define un fragmento de código en el que se cree puede producirse una excepción, mientras que el bloque *except* permite indicar la solución de la excepción, donde se imprime un mensaje de error para que el usuario conozca lo que está sucediendo en el programa [28].

1.3.8.6. Recomendaciones de programación en Python

Para no tener problemas al momento de programar o desarrollar un software con lenguaje Python se recomiendan las siguientes consideraciones [32]:

- Usar excepciones basadas en clases.
- Usar los métodos de *string* y no los módulos *string.*
- No escribir *strings* literales que hagan uso de espacios en blanco al final de una línea.

Para más información sobre el lenguaje de programación Python puede ingresar al siguiente enlace: *https://docs.python.org/3/*.

1.3.9. TECNOLOGÍA XML (EXTENSIBLE MARKUP LANGUAGE)

Esta tecnología hace referencia a un conjunto de módulos, en donde éstos otorgan servicios que funcionan para las demandas más frecuentes de los usuarios; estos módulos sirven para estructurar, intercambiar y almacenar información. El núcleo de la tecnología XML es su propio lenguaje XML [29].

1.3.9.1. Lenguaje XML

El lenguaje extensible de etiquetas es un lenguaje de marcas que, a diferencia de HTML *(HyperText Markup Language)*, su función principal no es mostrar los datos, es describirlos de una manera eficaz. Un lenguaje de etiquetas es el que permite codificar un documento, de tal manera que unido al texto se agrupan marcas que contiene información adicional acerca de la estructura del texto [29].

1.3.9.2. Ventajas del lenguaje XML

Las principales ventajas de trabajar con lenguaje XML son [30], [31]:

- Metalenguaje extensible, por lo que se puede etiquetar en cualquier momento sin complicaciones.
- Estructura fácil de entender y procesar, ayudando a la compatibilidad entre aplicaciones.
- No se necesita de una licencia para usar la tecnología XML.

1.3.9.3. Sintaxis de XML

El archivo XML es especial porque está compuesto por un número de etiquetas en forma de árbol. Todos los elementos del árbol están vinculados entre sí a través de las ramas que se crean. A diferencia de un archivo HTML en XML cualquier etiqueta abierta debe estar cerrada.

Para más información sobre la tecnología XML puede ingresar al siguiente enlace: *https://www.w3.org/XML/.*

1.3.10. SMSGLOBAL

SMSGlobal es un proveedor internacional de servicios de SMS móvil con una plataforma web de SMS personalizada y con precios competitivos. Ofrece soluciones de comunicación innovadoras y fáciles de usar; maneja una variedad de soluciones de mensajería de texto para todo tipo de empresa.

En la figura 1.6 se puede apreciar la interfaz de SMSGLOBAL [33].

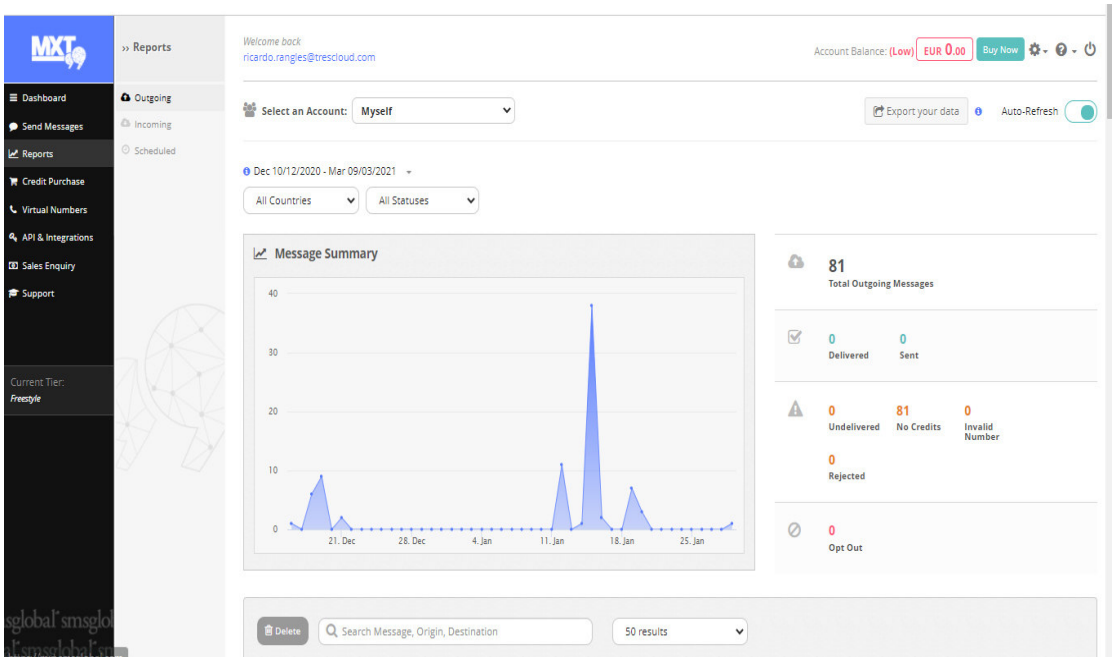


Figura 1.6. Interfaz SMSGlobal

1.3.11. ZABBIX

Zabbix es una solución de monitoreo de redes, que está diseñada para registrar y monitorear el estado de varios servicios de red, servidores, hardware de red y aplicaciones. Es de código abierto, de clase empresarial y es gratuito; está escrito y distribuido bajo la GPL *(General Public License)* versión 2, esto significa que su código fuente se distribuye libremente y está disponible para el público en general [34].

Este software de monitoreo utiliza un mecanismo de notificación flexible que permite a los usuarios configurar alertas basadas en correo electrónico y mensajes de SMS para prácticamente cualquier evento. Esto permite una reacción rápida a los problemas del servidor.

Si se configura correctamente, Zabbix puede desempeñar un papel importante en la supervisión de la infraestructura de TI (Tecnología de la Información). Esto es igualmente cierto para las organizaciones pequeñas con pocos servidores y para las grandes empresas con una multitud de servidores [34].

1.3.11.1. Características de Zabbix

Zabbix permite controlar servidores, dispositivos de red y aplicaciones, junto con reportes de datos estadísticos de rendimiento. Dentro de las características más destacadas de Zabbix se tienen [35], [36]:

- Alto Rendimiento.
- Monitoreo centralizado.
- La instalación de agentes se realiza en cualquier S.O.
- Alta capacidad de análisis de servicios prestados.
- Notificaciones que permiten una fácil integración de sistemas.
- Permite el uso externo de *scripts.*
- Construcción de gráficos y opciones de visualización.

1.3.11.2. Funciones principales de Zabbix

Zabbix es una solución de monitoreo de red altamente integrada que ofrece una multiplicidad de funciones en un solo paquete. A continuación se especifican algunas funciones de Zabbix [37]:

- Recopilación de datos.
- Alertas configurables.
- Gráficos en tiempo real.
- Capacidades de monitoreo web.
- Almacenamiento de datos históricos.
- Fácil configuración.
- API de Zabbix para interconectar Zabbix con Odoo.

1.3.11.3. Ventajas de Zabbix

A continuación, se mencionan algunas ventajas de utilizar Zabbix dentro del monitoreo de una red [36]:

- Capacidad de encontrar diferentes dispositivos de red como servidores, impresoras y periféricos a través de varios protocolos como SNMP (*Simple Network Management Protocol*).
- Posee un sistema de administración centralizada desde un monitor web donde se encuentran en una misma interfaz todos los dispositivos de la red.
- Posee un sistema de manejo de usuarios junto con una autenticación mediante contraseñas y usuarios; para esto cada usuario posee su perfil con acceso propio.
- Existen diferentes maneras de alertar un evento, las más utilizadas son a través de correo electrónico y mensajes SMS, los cuales son enviados automáticamente cuando algún dispositivo sufre algún problema.

1.3.11.4. Arquitectura de Zabbix

Zabbix posee varios elementos para el funcionamiento del sistema de monitoreo de una red. En la Tabla 1.9 se muestran los elementos que forman parte de la arquitectura de Zabbix [39].

Tabla 1.9 Elementos del software Zabbix [36]

Elementos	Descripción
Agente Zabbix	Componente propio de Zabbix basado en el protocolo SNMP que permite monitorear en tiempo real los recursos de la red.
Key	Son etiquetas utilizadas para organizar el tipo de información que se va a analizar.
Triggers	Son módulos o disparadores creados por el administrador de red para evaluar cada dato recopilado de acuerdo con un ítem y a un *key*.
Ítems	Son módulos que se encargan de recoger la información del host.
Host	Son los equipos registrados en la consola de la administración, los cuales son los dispositivos monitoreados.
Servidor recolector Zabbix	Es el servidor principal donde se encuentra instalada la consola de administración y la base de datos central de Zabbix, en donde se recopila la información de los dispositivos gestionados en Zabbix.

En la Figura 1.7 se muestra la arquitectura que maneja Zabbix.

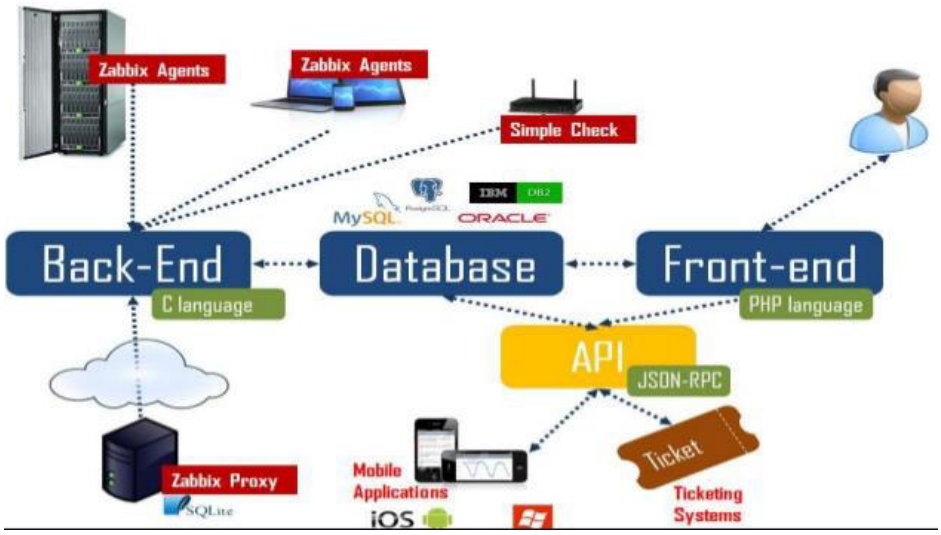


Figura 1.7. Arquitectura de Zabbix [38]

1.3.11.5. Tipos de monitoreo en Zabbix

En Zabbix existen dos maneras para monitorear un *host*; la primera es mediante el monitoreo activo en donde el agente toma el rol de activo, esto quiere decir toma la iniciativa de comunicar la información al servidor Zabbix. Como ventajas de este tipo de monitoreo se tienen la disminución de carga del servidor Zabbix y el mejor uso del ancho de banda al poder enviar más de un valor por conexión.

Por otro lado se encuentra el monitoreo pasivo, éste en cambio hace que el agente se comporte como pasivo y por tanto va a esperar a que le soliciten la información; este tipo de monitoreo es el más fácil de implementar [40].

En la Figura 1.8 se muestra cómo funcionan ambos monitoreos en Zabbix.

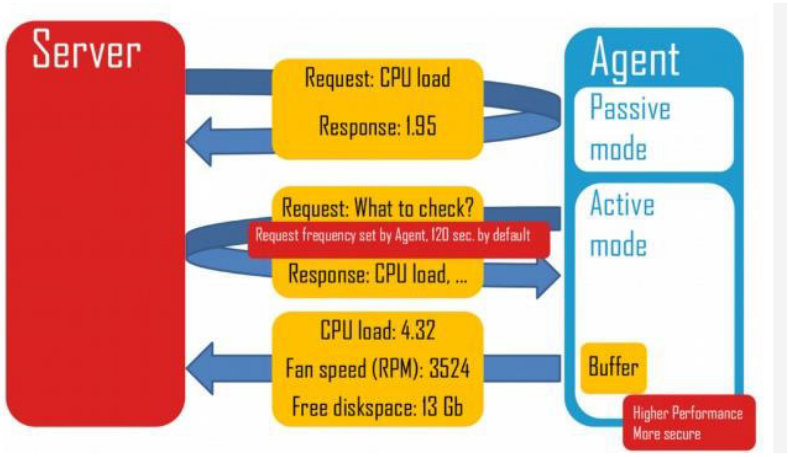


Figura 1.8. Monitoreo activo y pasivo en Zabbix [38]

1.3.11.6. Configuraciones de Zabbix

En la Tabla 1.10 se mencionan las configuraciones de Zabbix que fueron utilizadas en el proyecto.

Tabla 1.10 Configuraciones de Zabbix [41], [42], [43], [44]

Configuración de Zabbix	Función
Host	Los *hosts* que más se utilizan en Zabbix son los dispositivos que desean ser monitoreados, tales como servidores, estaciones de trabajo, conmutadores, entre otros. La creación de *hosts* es una de las primeras tareas de monitoreo en Zabbix. Los *hosts* están organizados dentro de los host *groups*.
Triggers	Los *triggers* o disparadores son expresiones lógicas que calculan datos recopilados por elementos y representan el estado actual del sistema.
Notificaciones sobre Eventos	Al momento de configurar algunos elementos y *triggers,* se debe tomar en cuenta algunas acciones. Una de ellas es recibir una notificación si ha sucedido algo importante, como por ejemplo un problema en un *host*. Además, cuando surgen estos problemas, es necesario informar al usuario lo que está sucediendo.
User	Todos los usuarios de Zabbix acceden a la aplicación Zabbix a través de la interfaz basada en web. A cada usuario se le asigna un nombre de inicio de sesión y una contraseña únicos. Todas las contraseñas de los usuarios están cifradas y almacenadas en la base de datos de Zabbix.
Artículos de monitoreo web	Algunos elementos nuevos se agregan automáticamente para monitorear cuando se crean escenarios web. Al momento de que se crea un escenario, Zabbix agrega automáticamente todos los campos para monitorear, estos campos están vinculados a la aplicación seleccionada.

1.3.11.7. Cifrado

Para el cifrado, Zabbix admite comunicaciones cifradas entre sus componentes. Además admite cifrado basado en certificados y en claves previamente compartidas. El cifrado se puede configurar para las conexiones entre el servidor Zabbix, el agente Zabbix y la base de datos de Zabbix desde la interfaz de Zabbix y el servidor proxy [45].

Las conexiones en Zabbix se pueden usar [45]:

- Sin cifrado (predeterminado).
- Con cifrado basado en certificado RSA (*Rivest, Shamir, Adleman*).
- Con cifrado basado en PSK. (*Pre- Shared Key*).

En la Tabla 1.11 se muestra un ejemplo de configuración de cifrado para cada *host.*

Ejemplo	Conexiones al host	Conexiones permitidas del host	Conexiones rechazadas del host
NONE	Sin cifrar	Sin cifrar	Cifrado, certificado y cifrado basado en PSK
CERT NONE PSK CERT	Cifrado, basado en certificado	Cifrado, basado en certificado	Sin cifrar y cifrado basado en PSK
PSK NONE PSK CERT	Cifrado, basado en PSK	Cifrado, basado en PSK	No cifrado y cifrado basado en certificado
PSK NONE PSK CERT	Cifrado, basado en PSK	Sin cifrar y cifrado basado en PSK	Cifrado basado en certificado
CERT NONE PSK CERT	Cifrado, basado en certificado	Sin cifrar, PSK o cifrado basado en certificado	-

Tabla 1.11. Configuraciones de cifrado en Zabbix [45]

1.3.11.8. API de Zabbix

La API de Zabbix permite recuperar y modificar mediante programación la configuración de Zabbix y proporciona acceso a datos históricos. Básicamente se usa ampliamente para [46]:

- Crear nuevas aplicaciones para trabajar con Zabbix.
- Integrar Zabbix con software de terceros.
- Automatizar las tareas rutinarias.

La API de Zabbix es una API basada en web y se envía como parte de la interfaz web. Para esto se utiliza la librería Pyzabbix para ocupar los métodos de la API de Zabbix [47].

En la Tabla 1.12 se mencionan los métodos de la API de Zabbix que fueron utilizados para el proyecto.

Tabla 1.12. Métodos de la API Zabbix utilizados [45], [48], [49], [50]

Métodos API de Zabbix	Función
trigger.create()	Permite crear nuevos disparadores.
httptest.create()	Permite crear nuevos escenarios web, al momento de crear un escenario web se creará automáticamente un conjunto de elementos de supervisión web.
httptest.delete()	Permite eliminar escenarios web.
application.create()	Permite crear una nueva aplicación en Zabbix.
httptest.get()	Permite recuperar escenarios web de acuerdo con los parámetros dados.
host.get()	Permite recuperar hosts de acuerdo con los parámetros dados.

Para más información sobre el software de monitoreo de redes Zabbix puede ingresar al siguiente enlace: *https://www.zabbix.com/documentation/current/manual* .

1.3.12. ODOO (ON DEMAND OPEN OBJECT)

Odoo conocido antes como OpenERP es un sistema de planificación de recursos empresariales (ERP) y también de administración de la relación con clientes (CRM); de código abierto, fue desarrollado por la empresa belga OpenERP. En el año 2014 cuando se presentó la versión 8 decidieron cambiar el nombre con la finalidad de realizar un homenaje a su licencia libre y filosofía de código abierto [51], [52].

Odoo cuenta con una infinidad de módulos específicos para cada actividad como ventas, gestión de proyectos, contabilidad, inventarios, fabricación, comercio en línea, tienda *online*, recursos humanos entre otros.

Además añade en su mayoría de módulos, herramientas de análisis y generación de informes lo que le hace a este software un sistema muy poderoso para uso profesional, ya que integra muchos procesos de la empresa y también permite mantener todo el control de lo que ocurre en la empresa, haciendo que el entorno de trabajo sea fácil y correcto [51].

1.3.12.1. Características de Odoo

Odoo al ser un sistema completo de gestión de empresas posee algunas características que lo resaltan y lo hacen único [51]:

- Sistema sólido y estable.
- Multiplataforma.
- Acceso a información de manera consistente.
- Modular y escalable.
- Soluciones verticales.
- Integración con otros programas.

1.3.12.2. Funciones principales de Odoo

Odoo se ha dado paso dentro del mercado mundial, tanto así que es considerado como la herramienta de gestión empresarial más potente al poseer más de 500 módulos para dar soporte a cada área de una empresa. En la Tabla 1.13 se mencionan las principales funciones de Odoo.

Tabla 1.13. Funciones principales de Odoo [51]

Funciones Principales de Odoo	Definición
Completo	Odoo trae módulos base por defecto que permiten gestionar una empresa de manera predeterminada o estándar en todas las áreas.
Potente	Odoo cuenta con análisis y generación de información que ayudan a la gestión y visualización de la información.
Flexible	Odoo permite las modificaciones y adaptaciones de código para acoplarse a las necesidades de las empresas de forma ágil.
Libre	Odoo es un sistema de código abierto, basado en estándares abiertos y desarrollados en plataformas libres para cumplir las 4 libertades del software libre.
Accesible	Odoo al ser de código libre se distribuye bajo la licencia GPL, por lo que no se incurre en gastos de adquisición.

1.3.12.3. Ventajas de Odoo

Algunas ventajas de utilizar Odoo en una empresa para su debida gestión son [53]:

- Maneja una solución integral, ya que Odoo satisface todas las necesidades en una misma solución.
- Gracias a que es una plataforma *Open Source*, la versión *community* de Odoo además de contar con los módulos estándar, ofrece gran flexibilidad en cuanto a personalizar los módulos acorde a las necesidades de una empresa.
- El software es gratuito y no cuenta con ningún costo por licenciamiento, no existen renovaciones y sus actualizaciones son gratuitas.
- Odoo permite seguir creciendo según las necesidades de la empresa, si una empresa necesita nuevos módulos o usuarios, éste puede hacerlo sin limitaciones.

1.3.12.4. Arquitectura de Odoo

En la Tabla 1.14 se describen las distintas partes que componen la arquitectura de Odoo.

Tabla 1.14. Definición de la arquitectura de Odoo [52]

Capa	Descripción
Datos	Es la capa más baja en donde se almacena y se da persistencia a los datos mediante la tecnología PostgreSQL. Otros archivos binarios como adjuntos, documentos o imágenes se almacenan en los directorios del sistema. La capa de Datos proporciona información a la capa lógica.
Lógica	Es la capa que contiene todos los programas que son ejecutados; también recibe y envía peticiones del usuario, así como solicita datos a la base de datos. Es la capa que comunica los datos con la interfaz que ve el usuario.
Presentación	Es la capa de más alto nivel, es la interfaz gráfica que ve el usuario por pantalla. Es intuitiva y sencilla, y se comunica únicamente con la capa Lógica de forma ágil.

En la Figura 1.9 se muestra de mejor manera la arquitectura de Odoo.

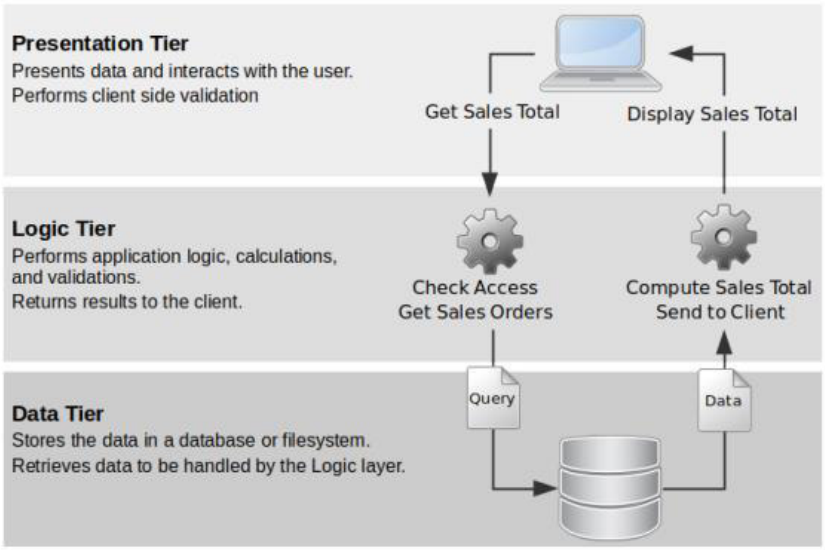


Figura 1.9. Arquitectura de Odoo [38]

1.3.12.5. Configuraciones de Odoo

1.3.12.5.1. API de Odoo

Odoo generalmente se extiende internamente a través de módulos, pero muchas de sus características, así como sus datos también están disponibles desde el exterior para análisis externos o integración con varias herramientas [54].

Para el desarrollo del proyecto se utilizó la librería ERP peek que es una herramienta versátil para navegar por datos de Odoo. La biblioteca ERP peek se comunica con cualquier servidor Odoo desde la versión 5.0 en adelante, utilizando la interfaz XML-RPC estándar o la nueva interfaz JSON-RPC.

En la Tabla 1.15 se indican los métodos de la API de Odoo que fueron utilizados para el proyecto.

Tabla 1.15. Métodos de la API Odoo utilizados [54]

Métodos API de Odoo	Función
erppeek.Client()	Este es el objeto de nivel superior en donde el servidor es la URL de la instancia, como http: // localhost: 8069. Si el servidor es un paquete Python Odoo, se usa para conectarse al servidor local. La base de datos es el nombre de la base de datos y el usuario debe existir en la tabla *res.users.* Si no se proporciona la contraseña, se le pedirá al iniciar sesión al usuario.
Client.execute()	El método de argumento es el nombre de un método osv.osv o un método disponible en este obj. Se permiten parámetros de método. Si es necesario, los argumentos de las palabras clave se recopilan en kwargs.

Para más información sobre el software de gestión empresarial Odoo puede ingresar al siguiente enlace: *https://www.odoo.com/documentation/14.0/* .

CAPÍTULO II

CAPÍTULO II

2. METODOLOGÍA

En este capítulo se pondrá en práctica todo lo estudiado en el Capítulo I para diseñar la plataforma de monitoreo en Zabbix y el módulo de interconexión Odoo - Zabbix con sus respectivas funcionalidades.

2.1. VISIÓN DEL PROYECTO

Desarrollo de un módulo de interconexión Odoo - Zabbix para la mejora de la plataforma de monitoreo de la empresa TRESCLOUD CIA LTDA.

2.2. ENTREVISTA

Para generar las historias de usuario, se realizó una misma entrevista a 3 gerentes de la empresa TRESCLOUD CIA LTDA. El modelo de la entrevista se encuentra en el Anexo A.

2.3. HISTORIAS DE USUARIOS

Con el propósito de estructurar claramente los requerimientos de los usuarios, se generan las historias de usuario en base a la información recabada en las entrevistas del punto anterior. Las historias de usuario se muestran en la Tabla 2.1.

Tabla 2.1. Historias de Usuarios

ID	Pregunta de referencia	Título	Descripción
HU_01	Pregunta 1	Comunicación entre sistemas Odoo y Zabbix	El usuario necesita que ambos sistemas puedan trabajar conjuntamente para monitorear su infraestructura.
HU_02	Pregunta 2	Monitoreo de métricas	El usuario desea que se monitoree todas las métricas posibles para tomar acciones correctivas en caso de fallo en su infraestructura.
HU_03	Preguntas 3 y 5	Notificaciones vía SMS	El usuario requiere ser notificado por vía SMS cuando un cliente o servidor Odoo sufra algún problema.
HU_04	Pregunta 5	Notificación de fallo de un cliente Odoo	El usuario desea ser notificado cuando un cliente Odoo se cayó o tuvo algún problema.

2.4. REQUERIMIENTOS

Los requerimientos funcionales del proyecto se han identificado basándose en la visión del mismo, y en las historias de usuario generadas en el punto anterior.

2.4.1. REQUERIMIENTOS NO FUNCIONALES

A continuación, se mencionan los requerimientos no funcionales del proyecto, cabe mencionar que estos requerimientos no son obtenidos de las historias de usuario realizadas antes.

- Utilizar la arquitectura Amazon Web Services.
- Utilizar el lenguaje de programación Python y la tecnología XML para crear el módulo de interconexión Odoo - Zabbix.
- Utilizar el lenguaje de programación Bash para permitir el envío de notificaciones vía SMS desde Zabbix.
- Utilizar el lenguaje de programación Python para notificar si un cliente cayó o tuvo problemas.
- Utilizar el *framework* Visual Studio Code para crear el módulo de interconexión Odoo - Zabbix.

2.5. FASE DE PLANEACIÓN

Considerando la arquitectura que se va a utilizar (Web Services), a continuación se describe el sistema de monitoreo propuesto.

El sistema de monitoreo a desarrollar cuenta con un módulo de interconexión entre Odoo y Zabbix, el cual utiliza el *framework* de Odoo para su creación; este módulo se comunica mediante la API de Zabbix para utilizar todos los métodos necesarios y crear el módulo. Este módulo estará disponible para instalarlo como un módulo más de Odoo para las empresas que lo deseen instalar; en este caso para la empresa TRESCLOUD CIA LTDA. se lo instalará y se lo utilizará para poder modificarlo de acuerdo a la necesidad.

Además, dentro del módulo de interconexión Odoo - Zabbix existirán varias funcionalidades para que el monitoreo de los clientes en Odoo sea más dinámico; estas funcionalidades son: Creación y eliminación de clientes al sistema de monitoreo Zabbix, envío de notificaciones al usuario tanto por vía *mail* como SMS, notificación de caída de un cliente Odoo en caso de tener una falla. Para esta última funcionalidad se utilizará la API de Odoo para utilizar todos los métodos necesarios y realizar esta función.

2.6. RESPALDO Y MANEJO DE VERSIONES

Con el propósito de guardar toda la información como respaldo y también realizar el control de cambios de lo programado, se subió todo el código a un repositorio propio de la empresa TRESCLOUD CIA LTDA. en GitHub. En la Figura 2.1 se muestra un ejemplo de una de las ramas del repositorio que se generó, en este caso, es la rama *2021-05-21-RR/Implementacion_modulo_interconexion_Trescloud.*

La creación de todas las ramas para el desarrollo del módulo de interconexión se lo hizo mediante el *framework* de Visual Studio Code.

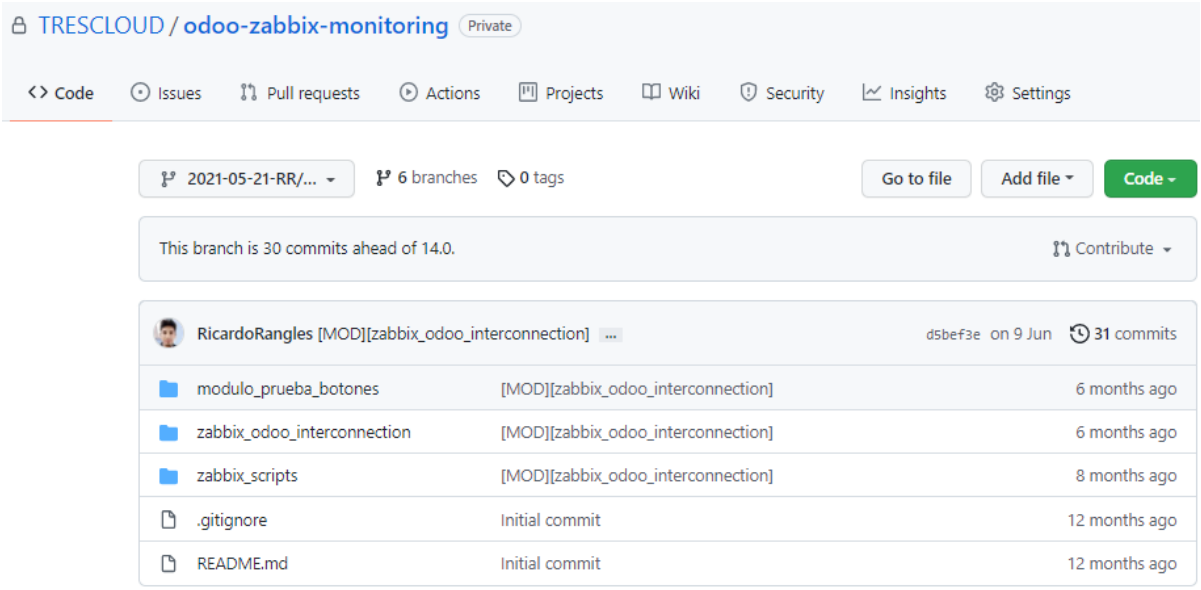


Figura 2.1. Repositorio levantado en GitHub para el módulo de interconexión

2.7. FASE DE DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE MONITOREO

2.7.1. DISEÑO DEL MÓDULO DE INTERCONEXIÓN ODOO – ZABBIX

Primero se debe definir qué clases van a intervenir en el desarrollo del proyecto. Inicialmente se creó la clase *LibreriaZabbix*, la cual tiene los siguientes campos:

- AGENT: Servirá para conectarse con el servidor Zabbix.
- company_id: Servirá para buscar el primer elemento en cada campo de res.company.

Además, esta clase contará con varios métodos que serán utilizados para las funcionalidades que complementan el módulo de interconexión Odoo - Zabbix. Estos métodos son los siguientes:

- *create_monitor_client:* Método que servirá para crear clientes dentro del servidor Zabbix.
- *remove_monitor_client:* Método que servirá para eliminar clientes dentro del servidor Zabbix.
- *verify_zabbix_application:* Método que servirá para verificar si el servidor Zabbix tiene creada la aplicación “Website”, esto con el fin de la creación de un cliente en Zabbix.

- *restart_instance_odoo:* Método que servirá para notificar que un cliente Odoo ha caído por algún problema.

Para la conexión al servidor de Zabbix se trabajó con la clase heredada *ResCompany*, la cual cuenta con los siguientes campos:

- host_id: Servirá como el id del servidor Zabbix a conectarse.
- url: Servirá como la dirección url del servidor Zabbix a conectarse.
- user: Credencial que servirá para tener acceso al servidor Zabbix.
- password: Credencial que servirá para tener acceso al servidor Zabbix.

Es necesario indicar que la empresa tiene una clase llamada *OdooDeployment* a la cual solo tiene acceso la empresa, es decir los desarrolladores de este módulo; por ese motivo no se puede realizar ninguna implementación ni prueba en producción, por lo tanto este desarrollo de monitoreo junto con el módulo de interconexión quedará como guía para que la propia empresa utilice para su propia necesidad.

Las relaciones entre estas clases se indican en el diagrama de clases que se encuentra en la Figura 2.2.

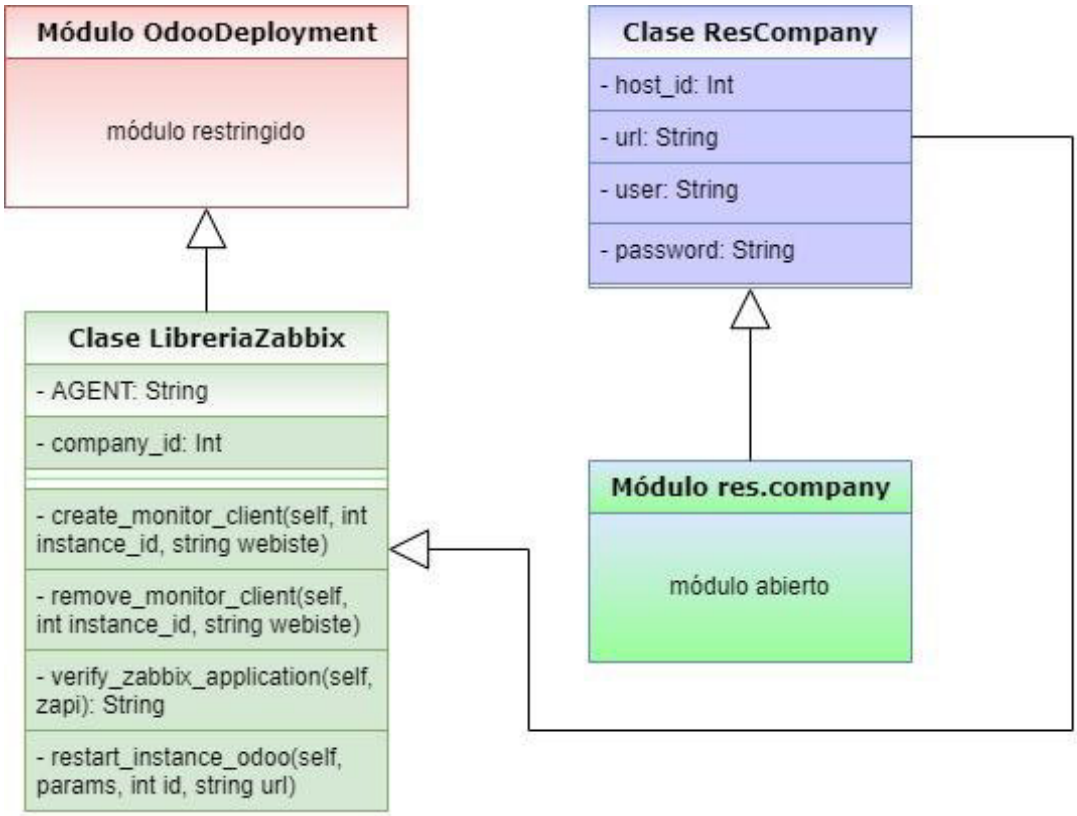


Figura 2.2. Diagrama de Clases

2.7.2. IMPLEMENTACIÓN DEL MÓDULO DE INTERCONEXIÓN ODOO – ZABBIX

La implementación del módulo de interconexión permite tener mayor facilidad para gestionar los clientes Odoo con la ayuda del sistema de monitoreo Zabbix.

2.7.2.1. Instalación del Servidor Zabbix

Primero se necesita instalar el servidor Zabbix dentro de un servidor de prueba que está corriendo en Ubuntu 18.04.5 LTS, todo con el fin de tener el sistema de monitoreo; con este sistema se va a poder monitorear otro servidor de prueba con el *framework* de Odoo permitiendo el desarrollo del módulo de interconexión con sus pruebas de funcionamiento.

Para que funcione la instalación de Zabbix en el servidor de prueba y poder replicar de mejor manera, en producción se utilizó Docker Compose; para esto se instaló Docker y Docker Compose en el servidor de pruebas, además se creó el archivo *docker-compose_v3_ubuntu_pgsql_latest trescloud.yaml* que en su contenido describe las configuraciones requeridas por cada imagen Docker que se está usando para cada componente de Zabbix que se desplegará al levantar el servidor completo de Zabbix; este archivo creado y modificado se encuentra en un repositorio personal luego de realizar un *Fork* del repositorio original de Zabbix.

En la Figura 2.3 se muestra el repositorio donde se encuentran los archivos para la instalación de Zabbix.

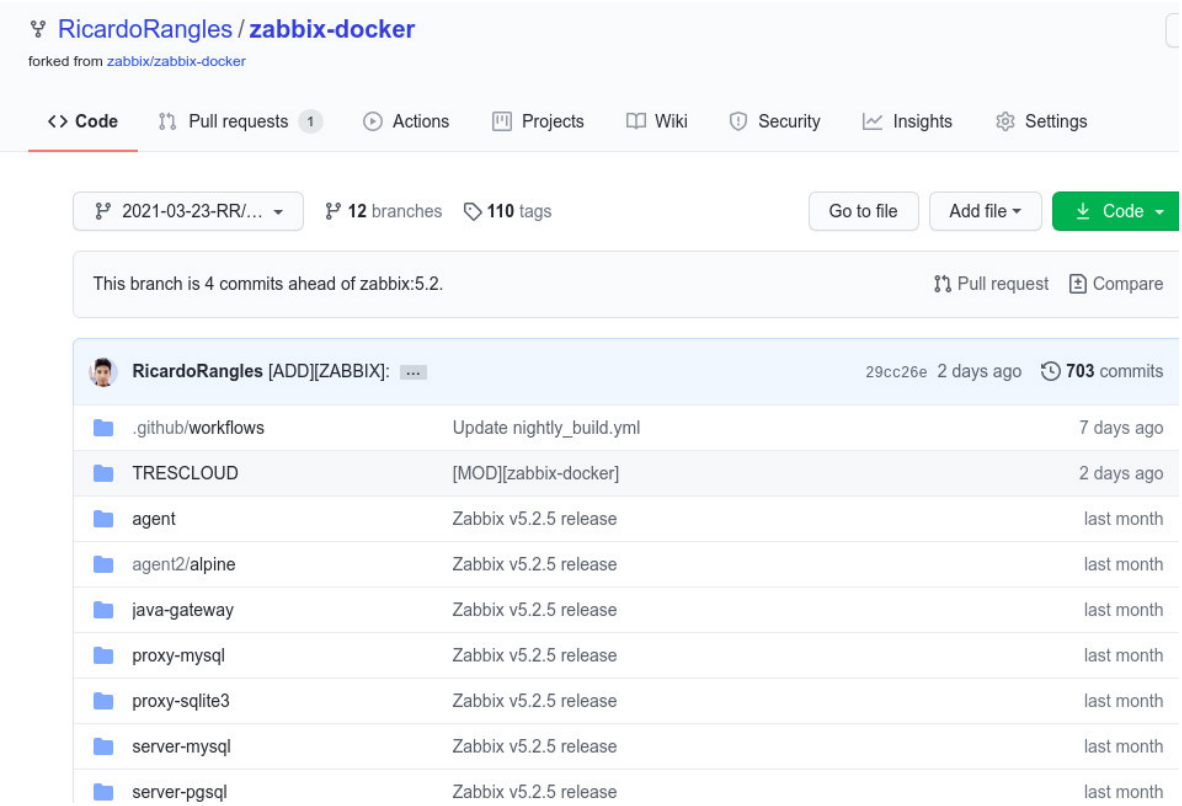


Figura 2.3. Repositorio zabbix-docker "realizado un Fork" en la cuenta personal Ricardo Rangles de Github

Dentro de este archivo. *yaml* se encuentran todos los componentes o servicios que Zabbix soporta para su uso, tanto como su base de datos, su servidor *web* y su *proxy*. Para el presente caso se van a utilizar ciertos servicios para que funcione Zabbix; los servicios o contenedores que se van a mantener en la configuración son:

- zabbix-server.
- zabbix-web-nginx-pgsql.
- zabbix-agent.
- zabbix-java-gateway.
- zabbix-snmptraps.
- postgres-server.

Una vez identificados los servicios que se van a utilizar para el funcionamiento de Zabbix, los demás servicios se los comenta para no tener ningún inconveniente al momento de levantar el servidor Zabbix en Docker que se encuentra en el servidor de prueba.

Se hizo una comparación entre la versión 5.0 de Zabbix y su última versión 5.2; entre las diferencias más notorias es que en la última versión Zabbix añadió la pestaña de *user roles* donde se pueden asignar a los usuarios del sistema y mantener los permisos específicos para cada función. Por lo tanto se instalará la última versión disponible al momento de realizar este análisis que es Zabbix 5.2 para estar más actualizado con el software.

El método de seguridad utilizado entre el agente y el servidor Zabbix será PSK (*Pre-Shared Key*) debido a que es una de las seguridades más viables para el uso de la empresa por su facilidad de uso; además se trabajará para la implementación del monitoreo de los servidores Odoo con monitoreo pasivo, ya que la empresa trabaja más con clientes que tienen salida a Internet que con clientes que necesiten una VPN para trabajar en línea, para este caso se necesitaría utilizar monitoreo activo entre Zabbix y el agente instalado en cada servidor Odoo.

Para la empresa TRESCLOUD CIA LTDA. es necesario conocer qué valores son críticos para saber si existe un problema con sus servidores Odoo, esto con la finalidad de avisar al administrador de que se puede estar suscitando un problema con los servidores de la empresa.

Los valores críticos a considerar son:

- Carga del sistema.
- Utilización del CPU: Indica el uso de la CPU, para conocer cuan ocupado está el procesador con la gestión de programas y procesos en el momento actual.
- Uso del CPU: Cuanta más capacidad de rendimiento tenga la CPU más rápido funciona el ordenador.
- Utilización de memoria: Indica el uso de memoria cuando se ejecutan los programas en la máquina en el momento actual.
- Uso de memoria: Proporciona el almacenamiento de información de la máquina.

En la Figura 2.4 se muestra el servidor Zabbix una vez instalado correctamente.

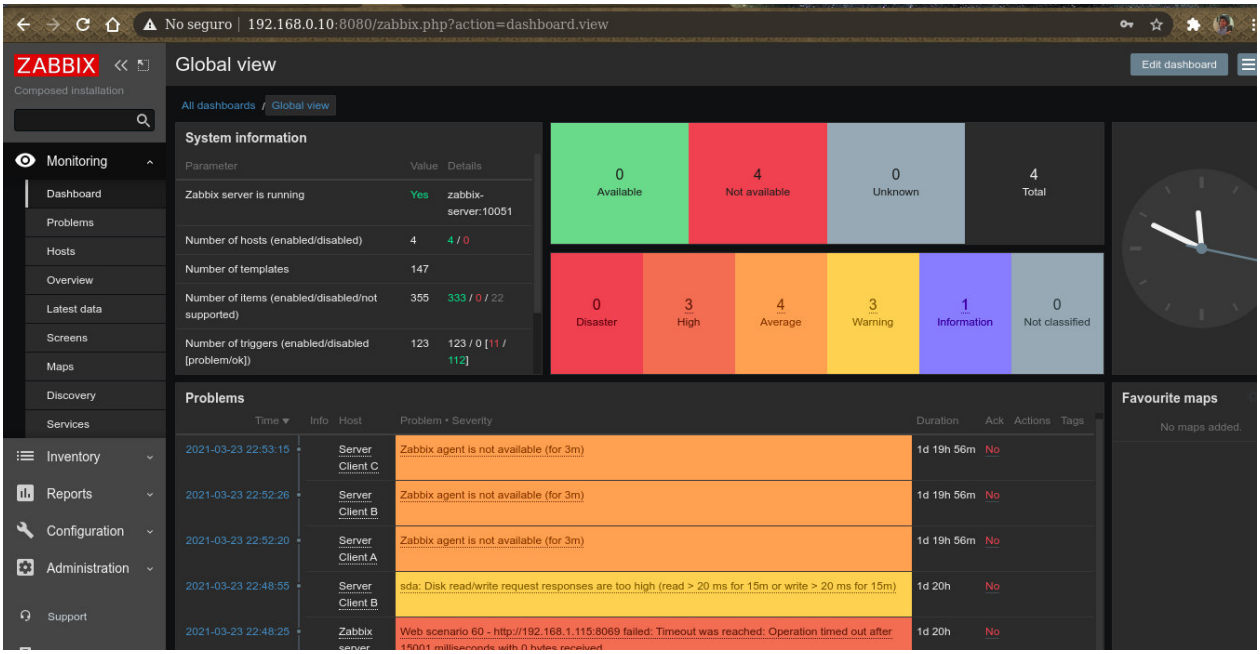


Figura 2.4 Servidor de Prueba con Zabbix v5.2

2.7.2.2. Instalación del Cliente Odoo

Para poder desarrollar el módulo de interconexión se necesita instalar Odoo, en este caso se instalará localmente, con el fin de desarrollar el módulo de interconexión y hacer todas las pruebas para el funcionamiento correcto del módulo. La instalación de Odoo se hará en una Laptop Dell Inspiron 3437 i5 y se utilizará la última versión 14 de Odoo.

Cabe mencionar que este servidor de Odoo servirá para ser utilizado en la implementación final del sistema de monitoreo y sus debidas pruebas, ya que en este servidor se instalará tanto el módulo de interconexión Odoo - Zabbix como el módulo denominado “modulo_prueba_ botones”.

En la Figura 2.5 se muestra el servidor Odoo una vez instalado correctamente.

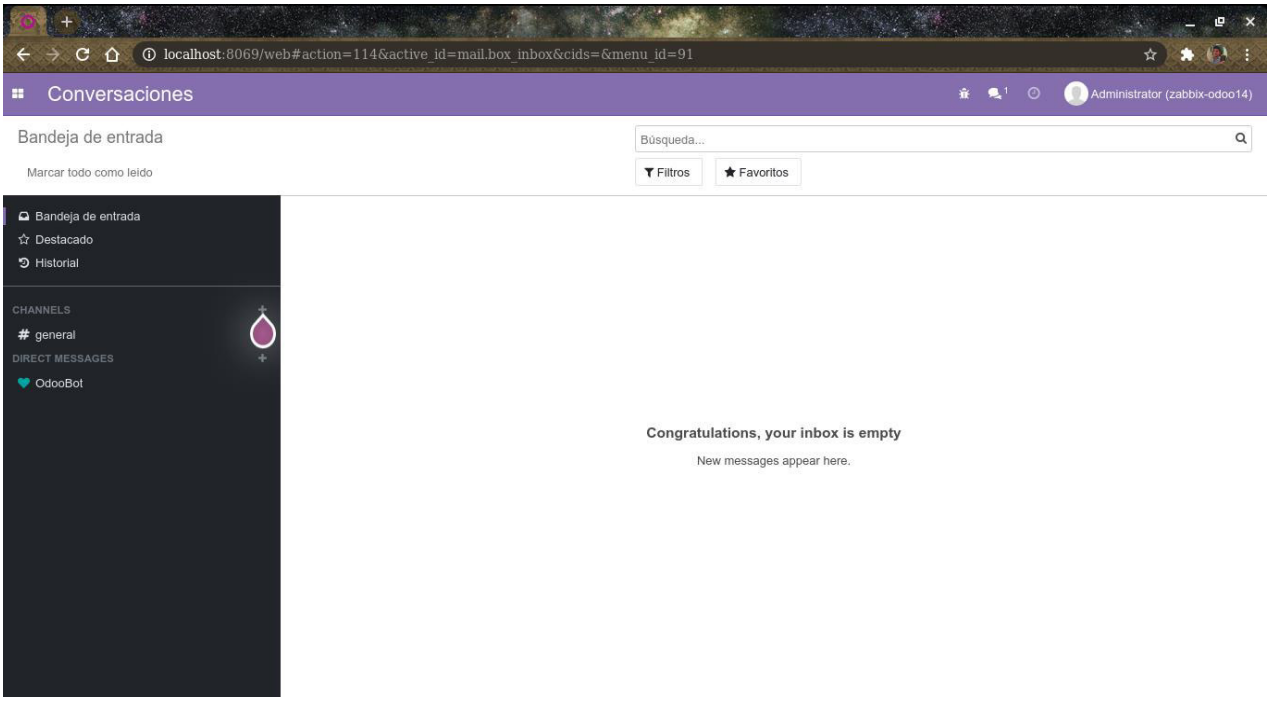


Figura 2.5 Servidor de Prueba con Odoo v14

2.7.2.3. Instalación de Visual Studio Code

Para definir el ambiente de pruebas se tiene el servidor Zabbix junto con el servidor Odoo; ahora se utilizará el editor de código fuente Visual Studio Code para codificar y depurar todo el módulo de interconexión, se utiliza VSC debido a que se puede utilizar varios lenguajes de programación como Python y C ++. En este caso se instala localmente para trabajar con Odoo que también se encuentra instalado localmente. Con esto se tendrá listo el ambiente de pruebas para crear el módulo de interconexión.

En la Figura 2.6 se muestra la interfaz de Visual Studio Code una vez instalado correctamente.

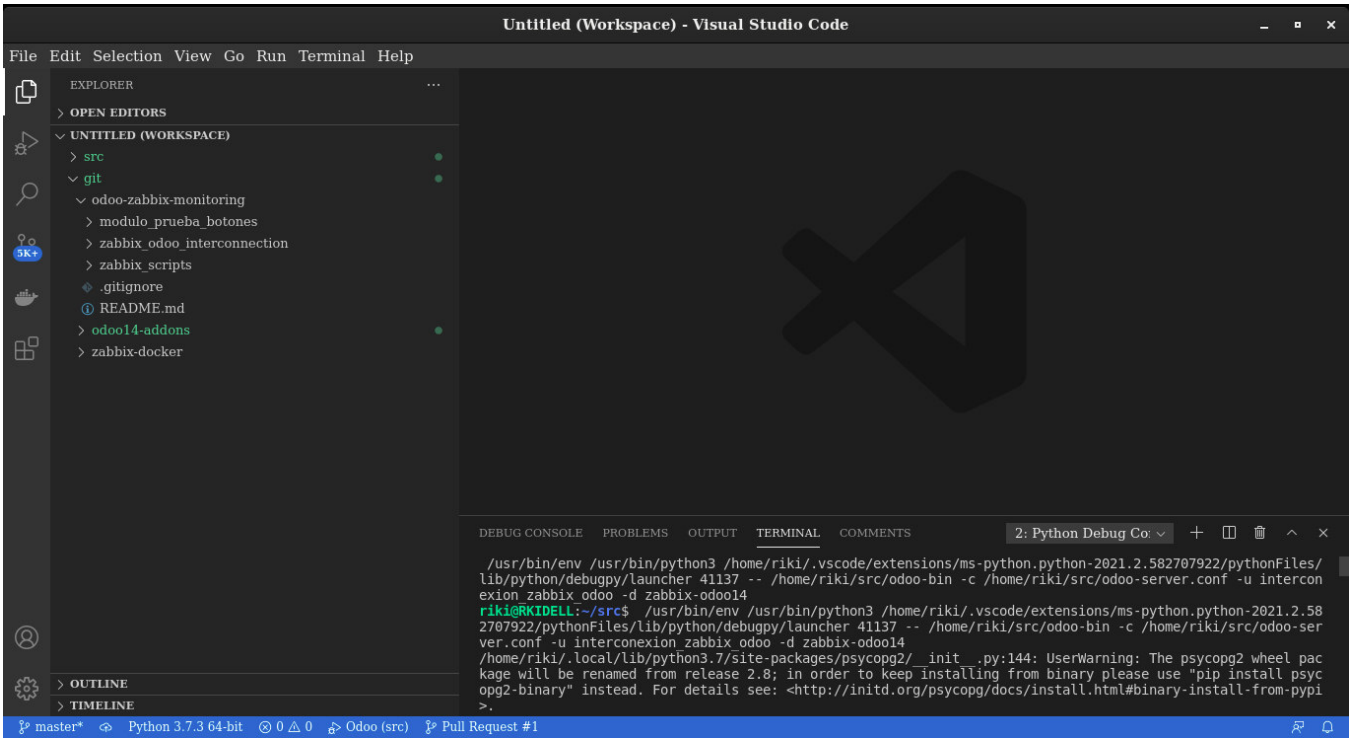


Figura 2.6 Instalación de Visual Studio Code

Una vez instalado Visual Studio Code se debe copiar la carpeta *src* dentro del *Workspace* de VSC, esta carpeta se creó al instalar Odoo y se encuentra en donde se haya instalado Odoo. Después de copiar la carpeta *src* se debe configurar el archivo *launch.json* que se encuentra en la pestaña *.vscode*, esto sirve para que “corra” el servidor Odoo dentro del editor VSC.

En la Figura 2.7 se muestra la configuración del archivo *launch.json* que permite iniciar el servidor Odoo en modo depuración, todo esto dentro del VSC.

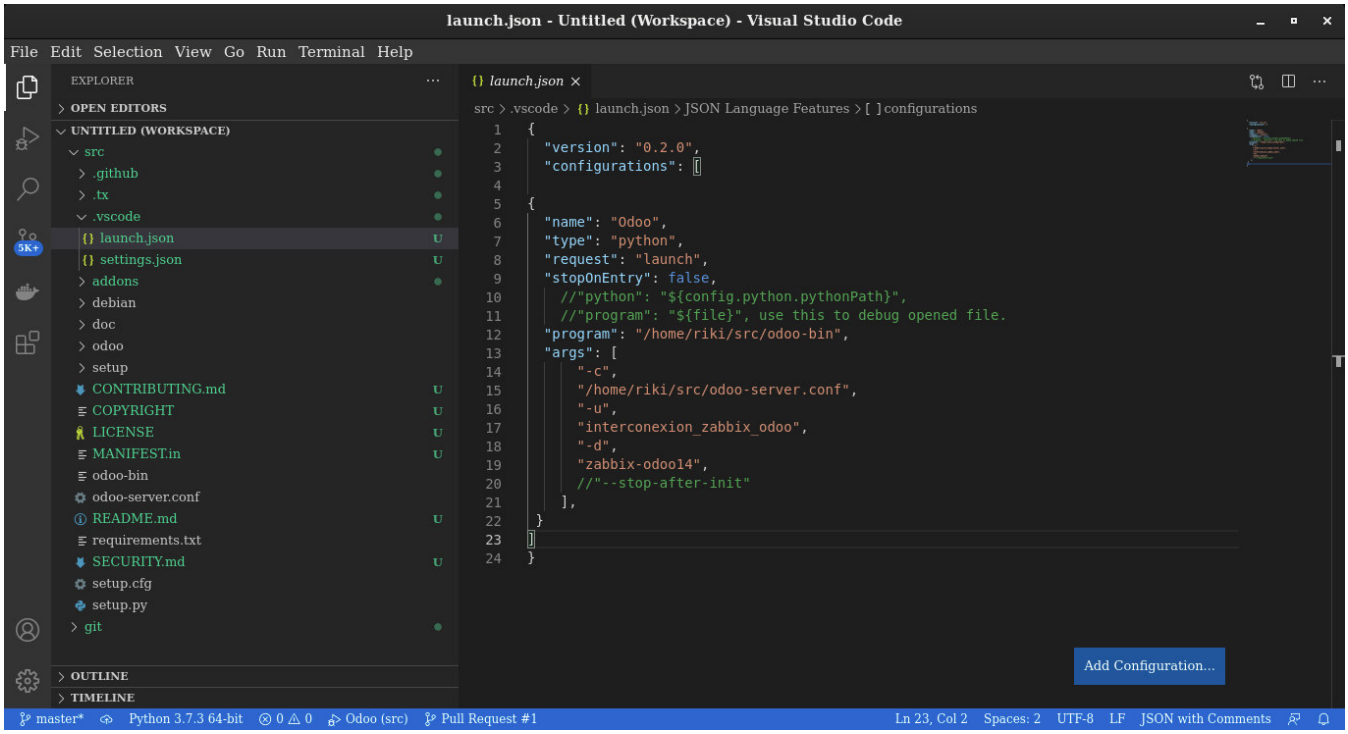


Figura 2.7 Configuración del archivo *launch.json*

2.7.2.4. Creación del módulo de Interconexión Odoo – Zabbix

Para crear el módulo de interconexión Odoo - Zabbix primero se crea una carpeta llamada *git* en VSC, para luego clonar el repositorio *odoo-zabbix-monitoring* que la empresa TRESCLOUD CIA LTDA. ha creado para este proyecto.

Una vez clonado en VSC el repositorio mencionado, se crea un módulo basado en Odoo llamado *zabbix_odoo_interconnection;* en este módulo se van a realizar todas las codificaciones necesarias para el funcionamiento del módulo de interconexión.

Al crearse el módulo se genera un archivo llamado *_manifest_.py*, el cual indica todo lo relacionado del módulo que se creó previamente.

En la Figura 2.8 se muestra la codificación del archivo *_manifest_.py.*

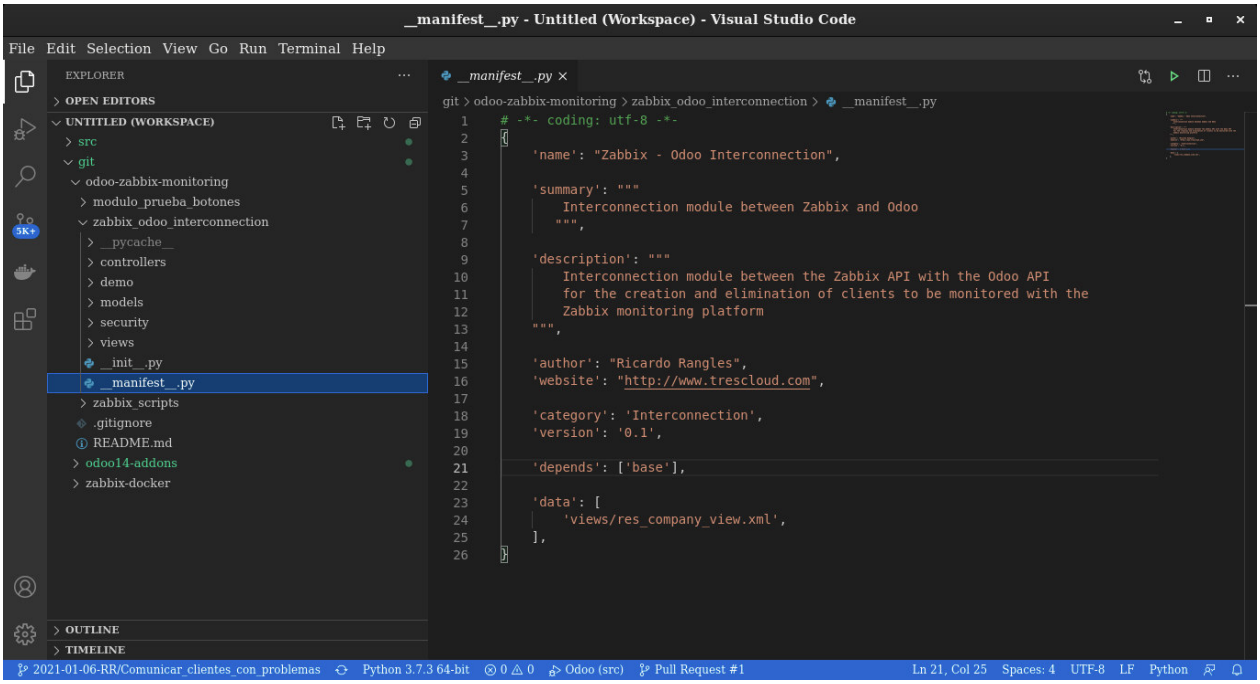


Figura 2.8 Archivo *_manifest_.py* de zabbix_odoo_interconnection

Para empezar a codificar el módulo de interconexión se crea el archivo *zabbix_odoo_library.py* dentro de la carpeta *models*, en la cual se crean todos los archivos necesarios para generar el módulo de interconexión.

Se empieza por codificar la función de agregar clientes Odoo al sistema de monitoreo. Esta función es propia del módulo de interconexión y va a servir para agregar clientes Odoo al sistema de monitoreo Zabbix.

En el Código 2.1 se muestra una parte de la codificación de la función de creación de clientes en el archivo *zabbix_odoo_library.py.*

```
22      def create_monitor_client(self, instance_id, website):
23          """
24          This function is to create a client to monitor in Zabbix using the button
25          "Add client for monitoring" created in the contact form of Odoo v14
26          """
27          #Unique code of the client instance
28          instance_id = str(instance_id)
29          #Value of the first item to search in each field of res.company
30          company_id = 1
31
32          #Validation in case the host_id variable is null
33          if not self.env["res.company"].browse(company_id).host_id:
34              raise ValidationError("The host_id of your Zabbix server has not been configured. Go to Settings -> Companies -> Select
35          #Variable host_id indicates the id of the server where the instances are located
36          host_id = self.env["res.company"].browse(company_id).host_id
37
38          #Validation in case the url variable is null
39          if not self.env["res.company"].browse(company_id).url:
40              raise ValidationError("The url of your Zabbix server has not been configured. Go to Settings -> Companies -> Select you
41          #Variable url indicates the URL of the Zabbix server where it is going to connect
42          url = self.env["res.company"].browse(company_id).url
43
44          #Validation in case the user variable is null
45          if not self.env["res.company"].browse(company_id).user:
46              raise ValidationError("The user of your Zabbix server has not been configured. Go to Settings -> Companies -> Select yo
47          #Variable user indicates the user to login to the Zabbix server
48          user = self.env["res.company"].browse(company_id).user
```

Código 2.1 Fragmento de la función de creación de clientes

Para agregar un cliente Odoo al sistema de monitoreo Zabbix se usan las librerías de Odoo y la librería pyzabbix que sirve para usar los métodos de la API de Zabbix, lo cual se puede visualizar en el Código 2.2. Las librerías que se muestran en Código 2.2 sirven para todas las funcionalidades que tiene el módulo de interconexión.

```
#Library that imports the Odoo methods necessary for this module
from odoo import models, fields, api

#Library that imports the necessary Zabbix methods for module interconnection
from pyzabbix.api import ZabbixAPI

#Library that imports notifications and errors to the user
from odoo.exceptions import ValidationError

#library that allows working with dates and times
from datetime import datetime
```

Código 2.2 Fragmento de la implementación de librerías de Odoo y Zabbix

Para que se pueda agregar un cliente Odoo al monitoreo de Zabbix se utiliza el método httptest.create() como se indica en el Código 2.3; este método crea un cliente dentro del sistema Zabbix con la url que el usuario añadió al momento de crear un contacto en el sistema Odoo.

```
zapi.httptest.create(
    name= instance_id + " - " + website,
    hostid=host_id,
    applicationid = application_id,
    agent=AGENT,
    steps=[
        {
            "name": "Homepage",
            "url": website,
            "status_codes": "200",
            "no": 1
        },
    ]
)
```

Código 2.3 Fragmento del uso del método httptest.create()

A continuación, se codifica la función de eliminar clientes Odoo del sistema de monitoreo (ver Código 2.4). Esta función también es propia del módulo de interconexión y va a servir para eliminar clientes Odoo del sistema de monitoreo Zabbix.

```
102     def remove_monitor_client(self, instance_id, website):
103         """
104         This function is to delete a client to monitor in Zabbix using the button
105         "Delete client for monitoring" created in the contact form of Odoo v14
106         """
107         instance_id = str(instance_id)
108         company_id = 1
109 
110         if not self.env["res.company"].browse(company_id).host_id:
111             raise ValidationError("The host_id of your Zabbix server has not been configured. Go to Settings -> Companies -> Select
112         host_id = self.env["res.company"].browse(company_id).host_id
113 
114         if not self.env["res.company"].browse(company_id).url:
115             raise ValidationError("The url of your Zabbix server has not been configured. Go to Settings -> Companies -> Select you
116         url = self.env["res.company"].browse(company_id).url
117 
118         if not self.env["res.company"].browse(company_id).user:
119             raise ValidationError("The user of your Zabbix server has not been configured. Go to Settings -> Companies -> Select yo
120         user = self.env["res.company"].browse(company_id).user
121 
122         if not self.env["res.company"].browse(company_id).password:
123             raise ValidationError("The password of your Zabbix server has not been configured. Go to Settings -> Companies -> Selec
124         password = self.env["res.company"].browse(company_id).password
125 
```

Código 2.4 Fragmento de la función de eliminación de clientes para el monitoreo

Así como se utiliza el método httptest.create() para agregar clientes Odoo al sistema de monitoreo en Zabbix, para eliminar clientes Odoo se utiliza el método httptest.delete(). En el Código 2.5 se muestra la forma de utilizar el método dentro del módulo de interconexión.

```
for web in web_list:
    if instance_id in web['name']:
        web_found = True
        #It's checked if the unique id and the URL are equal to the website name
        if instance_id + " - " + website == web['name']:
            zapi.httptest.delete(
                httptestid=web['httptestid']
            )
            #The function returns 1 when the URL has been correctly removed in Zabbix
            return 1

if not web_found:
    #The function returns 0 when the client ID with the URL is not being monitored in Zabbix
    return 0
```

Código 2.5 Fragmento del uso del método httptest.delete()

A continuación, se codifica la función llamada *verify_zabbix_application;* esta función lo que hace es verificar si la aplicación “Website” existe en el sistema de monitoreo Zabbix (ver Código 2.6), esto con el fin de que se pueda agregar un cliente Odoo al monitoreo Zabbix ya que es un parámetro necesario para éste.

```
def verify_zabbix_application(self, zapi):
    """
    This function is to check if the Website application exists in the Zabbix monitoring system
    """
    application_list = zapi.application.get(search={"name":"Website"})
    if application_list:
        applicationid = application_list[0]["applicationid"]
    else:
        applicationid = zapi.application.create(
            name="Website"
        )
    return applicationid
```

Código 2.6 Fragmento de la función de verificación de aplicación en Zabbix

Una vez terminadas todas las funciones del módulo de interconexión, se crea otro archivo dentro de la misma carpeta *models* llamado *res_company.py* (ver Código 2.7)*;* este archivo lo

que hace es heredar de la clase ResCompany y definir los campos que servirán para conectarse con el servidor Zabbix.

```
8     #Odoo res.company module inheritance
9     _inherit = 'res.company'
10
11    #Columns
12    host_id = fields.Char(string="Host ID")
13    url = fields.Char(string="URL")
14    user = fields.Char(string="User")
15    password = fields.Char(string="Password")
```

Código 2.7 Fragmento de la creación de los campos para conectarse a Zabbix

Cabe mencionar que los nombres de los archivos utilizados para este módulo deben estar en el archivo *_init_.py* que se encuentra dentro de la carpeta *models* del módulo (ver Figura 2.9), caso contrario no funcionará y no se encontrará en Odoo para instalar dicho módulo.

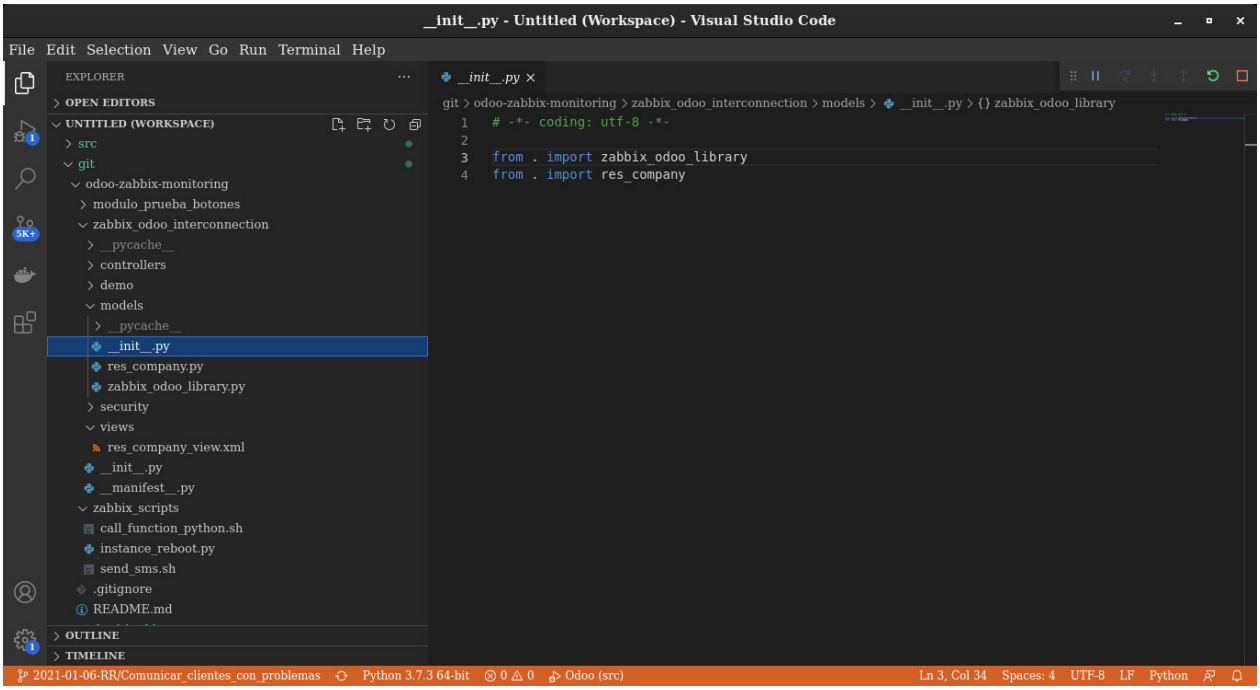


Figura 2.9 Archivo __init__.py del módulo zabbix_odoo_interconnection

Para que se pueda ver en la interfaz de Odoo la nueva pestaña para ingresar los datos de conexión al servidor Zabbix, dentro de la carpeta *views* se crea el archivo *res_company_view.xml* (ver Código 2.8*)*, con ello se diseña la pestaña, con lo que al momento

de reiniciar el servidor de Odoo se puede apreciar que se ha creado una nueva pestaña en el formulario de ResCompany.

```
 9          <xpath expr="//notebook" position="inside">
10              <page string="Zabbix" name='zabbix'>
11                  <group>
12                      <group name="group_zabbix" string="Zabbix">
13                          <field name="host_id"/>
14                          <field name="url"/>
15                          <field name="user"/>
16                          <field name="password"/>
17                      </group>
18                  </group>
19              </page>
20          </xpath>
```

Código 2.8 Fragmento de la creación de la pestaña Zabbix en ResCompany

En la Figura 2.10 se muestra la pestaña Zabbix una vez creada en vista de Odoo.

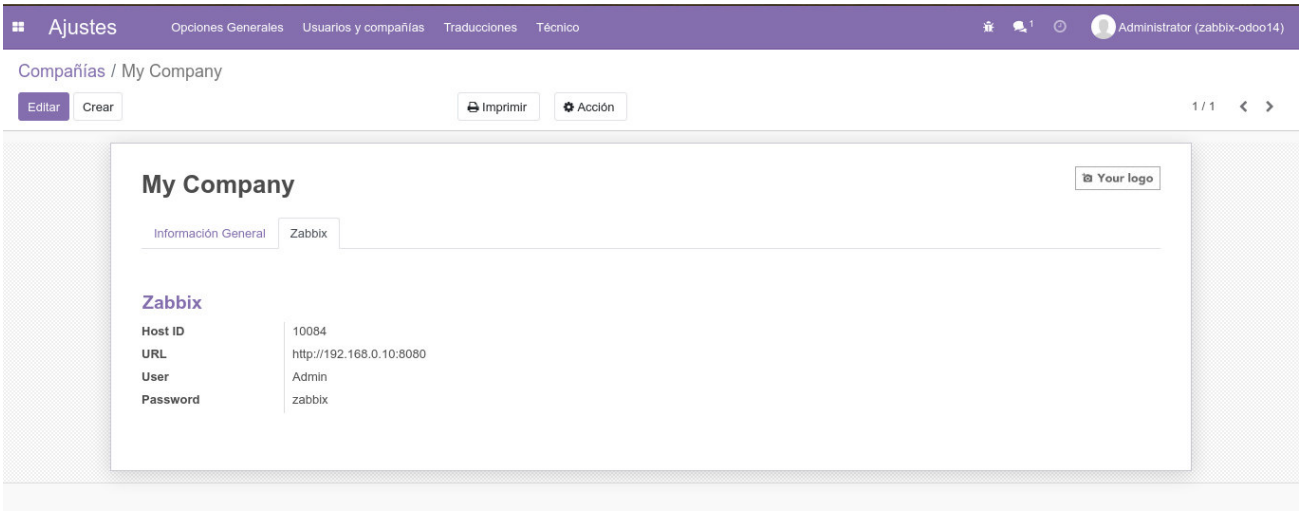


Figura 2.10 Vista de la pestaña Zabbix en Odoo

Una vez finalizado el proceso, lo que se debe hacer es instalar el módulo de interconexión en el servidor de prueba de Odoo previamente creado, para ello se utiliza la interfaz de Odoo, con esto ya se puede tener el módulo y todas las funcionalidades del mismo para realizar las pruebas de funcionamiento.

Para su instalación se debe ir a la pestaña de *Aplicaciones* y buscar por el nombre del módulo *zabbix - odoo interconnection* y luego instalarlo.

En la Figura 2.11 se muestra cómo se instala el módulo zabbix_odoo_interconnection.

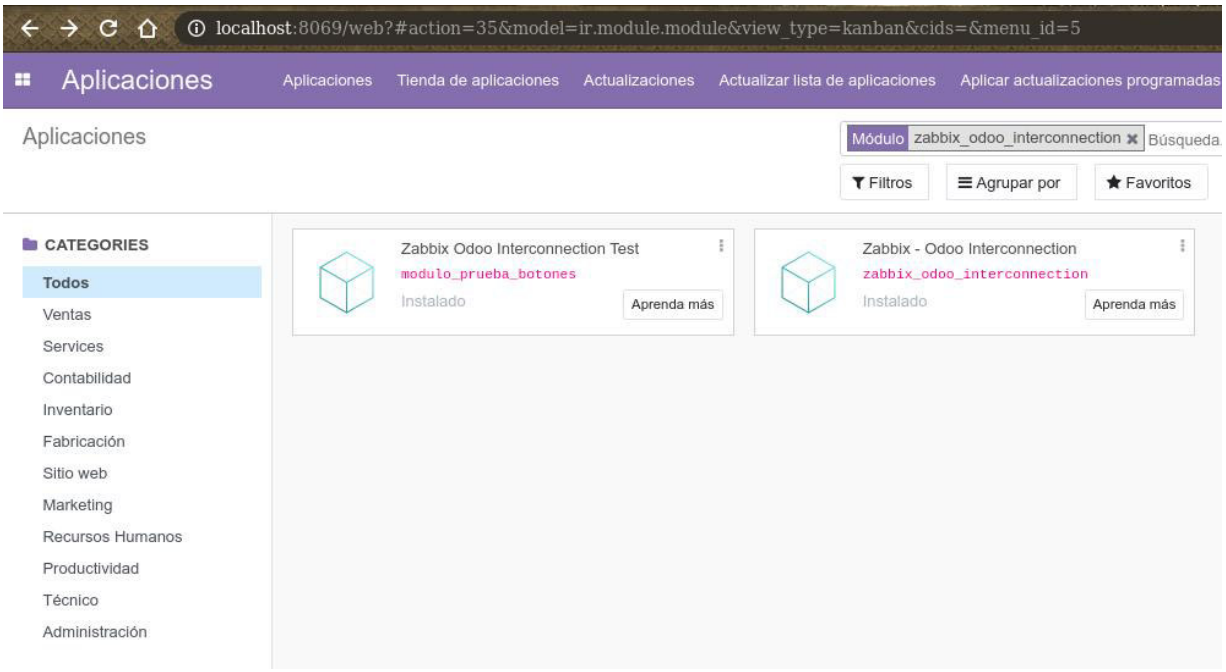


Figura 2.11 Instalación del módulo zabbix_odoo_interconnection en Odoo

En la Figura 2.12 se indican las características que tiene el módulo zabbix_odoo_interconnection.

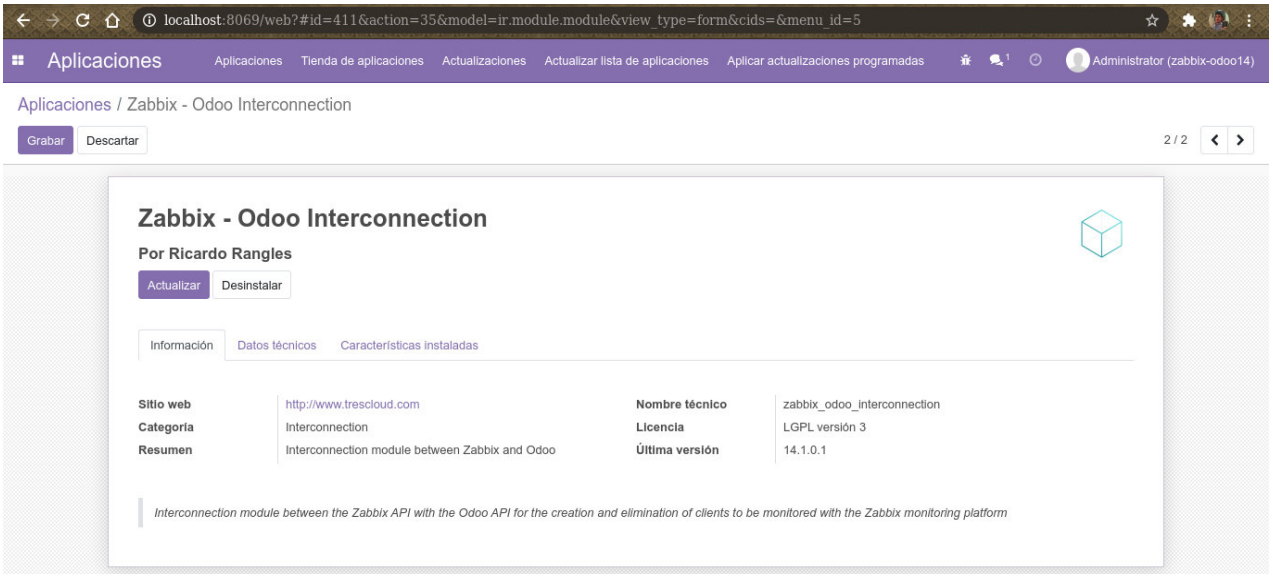


Figura 2.12 Especificaciones del módulo zabbix_odoo_interconnection

2.8. FASE DE DISEÑO E IMPLEMENTACIÓN DEL SCRIPT PARA EL ENVÍO DE NOTIFICACIONES POR VÍA SMS

2.8.1. DISEÑO DEL SCRIPT PARA EL ENVÍO DE NOTIFICACIONES POR VÍA SMS

Para el diseño del *script* sobre el envío de notificaciones mediante SMS, primero se deben entender los requerimientos que se necesitan para que funcione dentro del sistema de monitoreo Zabbix. Los requisitos para diseñar el *script son*:

- Servidor Zabbix activo.
- Uso del lenguaje de programación Bash.
- Editor para codificar el *script*.
- Tener una cuenta activa de un proveedor de SMS en línea.

Este *script* funciona de la siguiente manera: Cuando en el sistema de monitoreo Zabbix se genera un evento sobre algún cliente o un servidor Odoo, se generarán notificaciones que avisan al administrador correspondiente que algo ha sucedido con alguno de ellos; esto tiene como objetivo principal que se pueda avisar al usuario y solucionar el problema de inmediato, ya sea para reiniciar un cliente o un servidor Odoo.

Las notificaciones que están configuradas para el sistema de monitoreo en Zabbix son:

- Notificaciones vía correo electrónico.
- Notificaciones vía mensajes SMS.
- Notificaciones de que un cliente Odoo se cayó o tuvo problemas.

Estas notificaciones al configurarlas para los usuarios que necesiten ingresar al servidor Zabbix, tienen un parámetro denominado "*use if severity"* que indica cuándo se deben enviar las notificaciones, es decir en el caso de que el fallo de un cliente o un servidor Odoo sea de *"Not classified", "Information", "Warning", "Average", "High"* o del tipo *"Disaster"*. Para los usuarios que se han configurado en el sistema de monitoreo todas las notificaciones mencionadas se enviarán cuando el estado del fallo sea "*Average", "High"* y *"Disaster"*.

En la Figura 2.13 se muestra el diagrama de flujo para el *script* de envío de notificaciones mediante mensajes SMS.

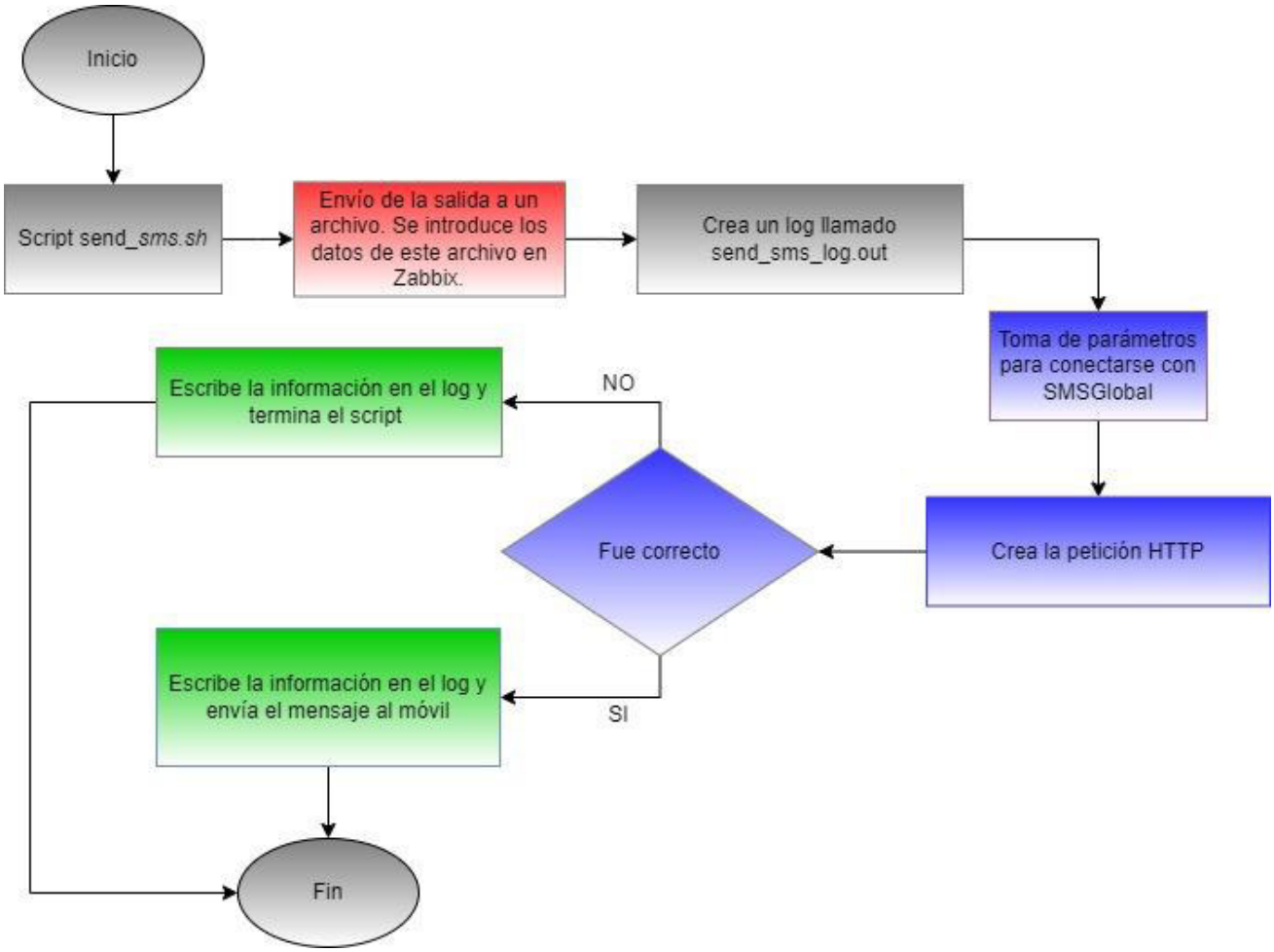


Figura 2.13 Diagrama de Flujo del script sms_send.sh

2.8.2. IMPLEMENTACIÓN DEL SCRIPT PARA EL ENVÍO DE NOTIFICACIONES POR VÍA SMS

2.8.2.1. Búsqueda de un proveedor de servicio SMS

Para la implementación del *script* que permite notificar al usuario por vía SMS desde Zabbix, que ha sucedido algún problema con un cliente o servidor Odoo, primero se realiza una

búsqueda de algún proveedor de servicios SMS que posibilite el envío de un mensaje SMS al número de teléfono del usuario que esté configurado dentro del servidor Zabbix.

Primero se encontró la aplicación Clickatell la cual cuenta con la funcionalidad de enviar mensajes SMS por lo que se creó una cuenta para verificar la funcionalidad.

Al momento de probar el mensaje de prueba, éste nunca llegó, por ende se procedió a seguir buscando una aplicación que permita enviar este tipo de notificaciones.

En la Figura 2.14 se muestra la interfaz de la aplicación Clickatell que fue utilizada para probar el envío de notificaciones por vía SMS.

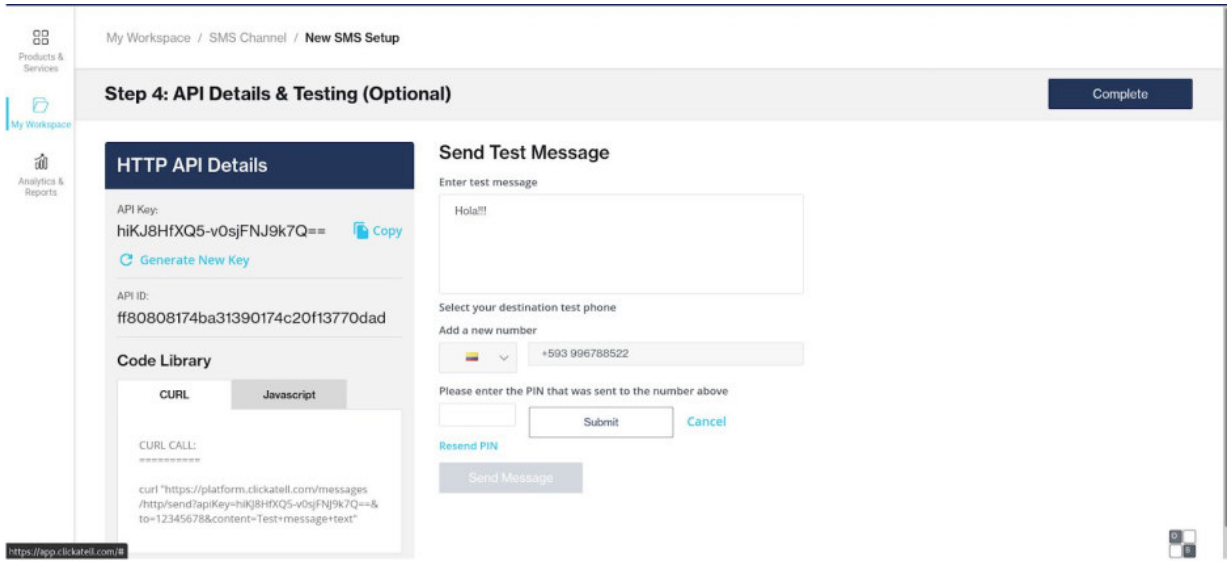


Figura 2.14 Prueba de envío de mensaje SMS

En la búsqueda de encontrar una aplicación que permita enviar SMS por vía Internet, se encontró “SMSGlobal” que es un proveedor de servicios de SMS con una plataforma web de SMS a medida y con precios competitivos, así como una variedad de soluciones de mensajería de texto para empresas, la cual según las configuraciones existentes y *plugins* para Zabbix permitió crear una cuenta para realizar la prueba y verificar su funcionamiento.

En la Figura 2.15 se muestra la interfaz de la aplicación SMSGlobal al crear una nueva cuenta.

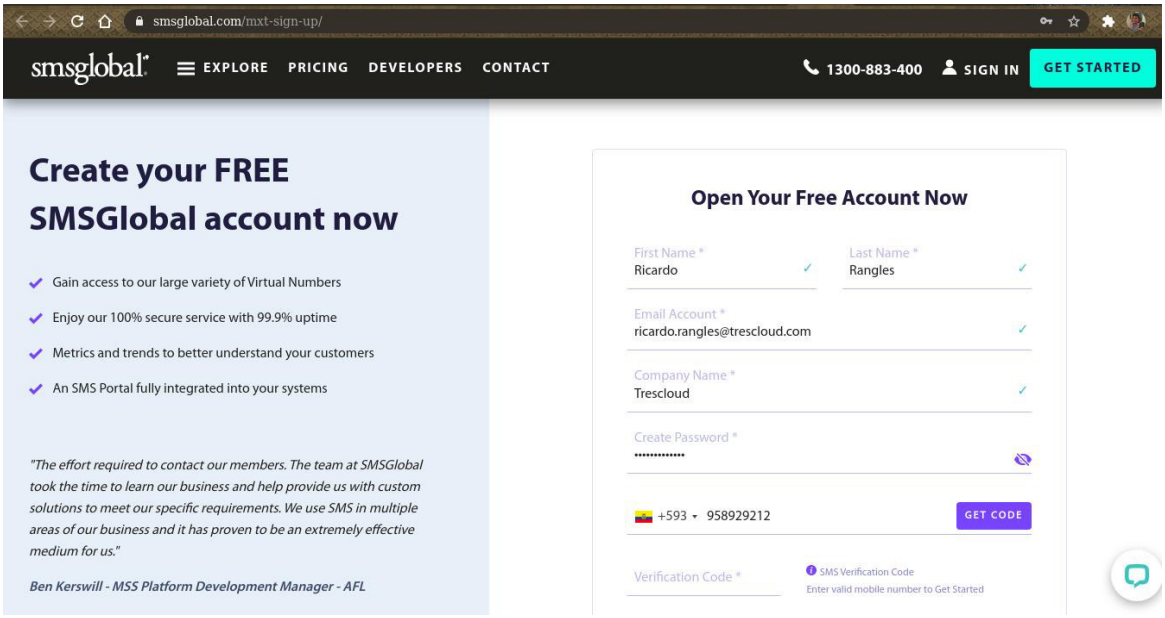


Figura 2.15 Creación de una cuenta en SMSGlobal

En la Figura 2.16 se muestra la cuenta creada de la aplicación SMS Global para el envío de notificaciones por vía SMS.

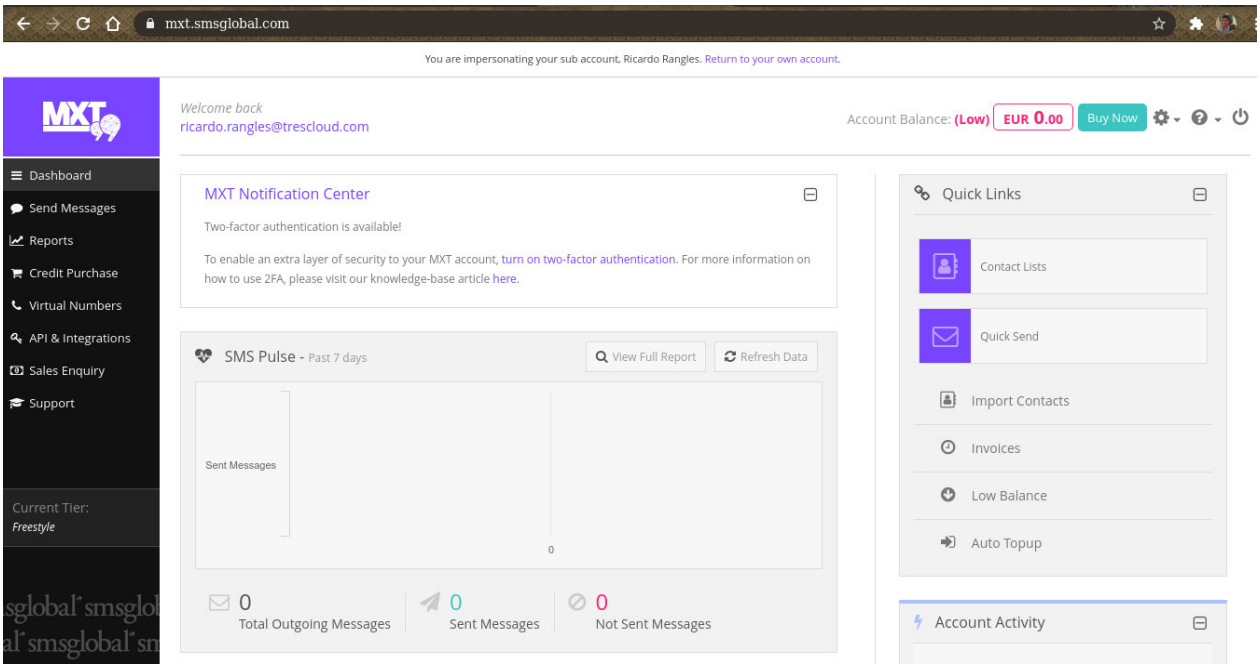


Figura 2.16 Cuenta creada en SMSGlobal

Al crear la cuenta en SMS Global se genera una llave denominada Master API key (ver Figura 2.17), la cual sirve para conectarse con esta aplicación para que envíe el mensaje de texto al número de teléfono que se encuentre configurado en el servidor de Zabbix.

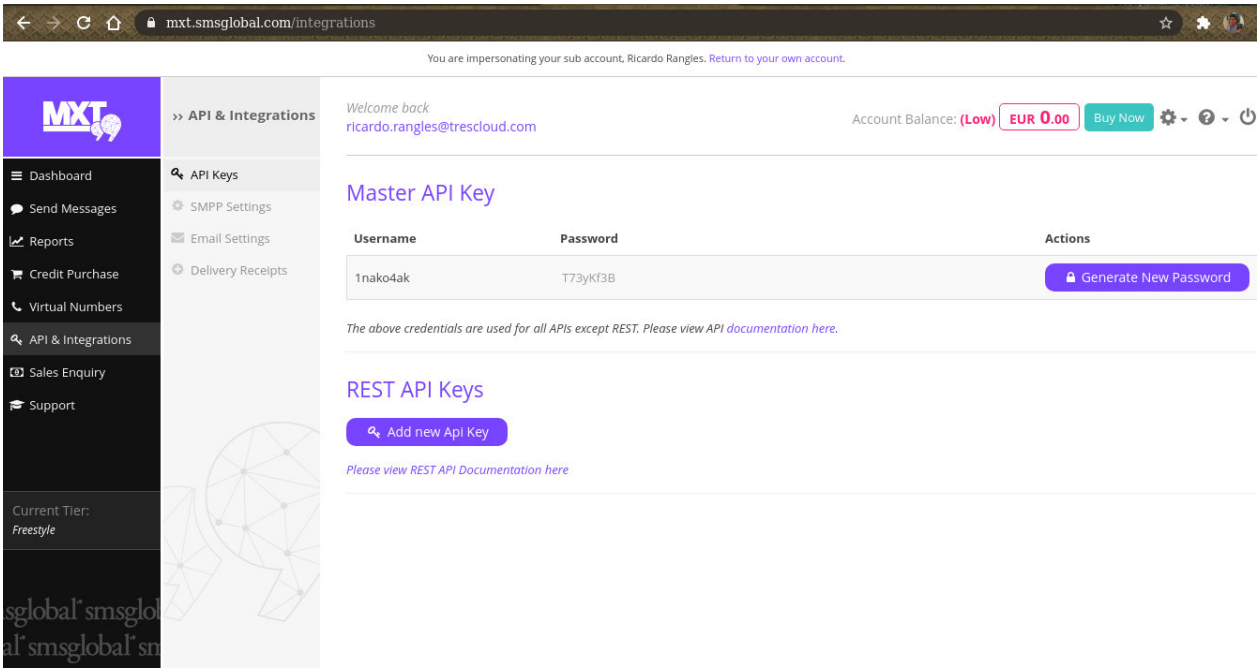


Figura 2.17 Master API key para conectarse con SMSGlobal

2.8.2.2. Creación del Script para el envío de notificaciones vía SMS

Para codificar el *script* primero se debe crear un archivo llamado *Dockerfile* (ver Figura 2.18) que se lo va a ocupar dentro del servidor Zabbix para que construya la imagen Docker e instale la librería *curl,* la cual es necesaria para realizar el envío SMS por medio de la aplicación SMSGlobal.

```
1   FROM zabbix/zabbix-server-pgsql:ubuntu-5.2-latest
2
3   RUN apt-get update && apt-get install -y curl \
```

Figura 2.18 Archivo Dockerfile para que funcione el script send_sms.sh

Para la programación del *script* primero se debe ubicar dentro del servidor de prueba Zabbix que se instaló previamente, luego según el *README* del *plugin* visto anteriormente el *script* debe estar en la siguiente ubicación:

/home/zadocker/zabbix-docker/zbx_env/usr/lib/zabbix/alertscripts

Una vez ubicado en esta dirección se crea el *script* para la conexión entre Zabbix y esta aplicación de mensajería SMS Global (ver Código 2.9); el *script* tiene por nombre *send_sms.sh* y se utiliza lenguaje de programación Bash.

```
# Your SMSGLOBAL credentials here
USERNAME=1nako4ak
PASSWORD=T73yKf3B
PHONE=${1}
MESSAGE=${2}
```

Código 2.9 Fragmento del uso de credenciales para conectarse con SMSGlobal

En el Código 2.10 se muestra el uso del comando *curl* para conectarse con la aplicación SMSGLOBAL.

```
curl -X POST "https://api.smsglobal.com/http-api.php?action=sendsms&user=$USERNAME&password=$PASSWORD&from=Test&to=$PHONE"
STATUS=$?
echo "SMS Text sent to $1"
echo "$MESSAGE"
exit $STATUS
```

Código 2.10 Uso del comando curl para conectarse a SMSGlobal

Una vez culminada la codificación del *script* deben darse los permisos necesarios para que funcione correctamente; a continuación, dentro del servidor Zabbix, se realiza la configuración para el envío de notificaciones por vía SMS. Para ello se creó un *media types* llamado SMS Global - SMS dentro de la interfaz del servidor Zabbix (ver Figura 2.19).

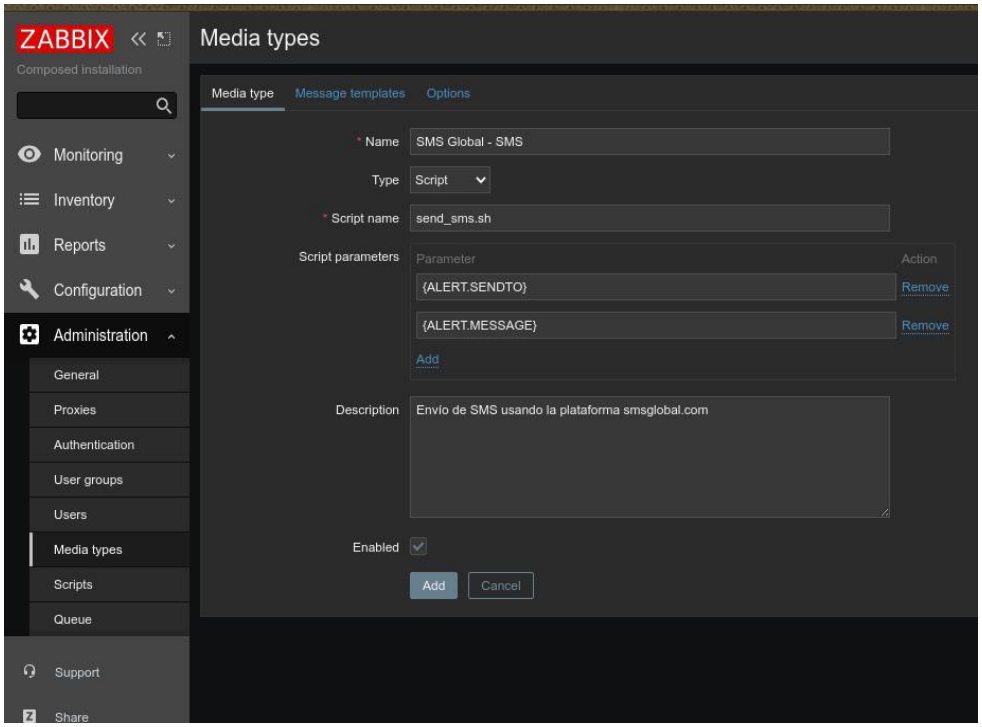


Figura 2.19 Configuración del media types SMS Global - SMS

Una vez creado este *media types* se realiza la prueba para verificar si el *script* funciona correctamente (Ver Figura 2.20).

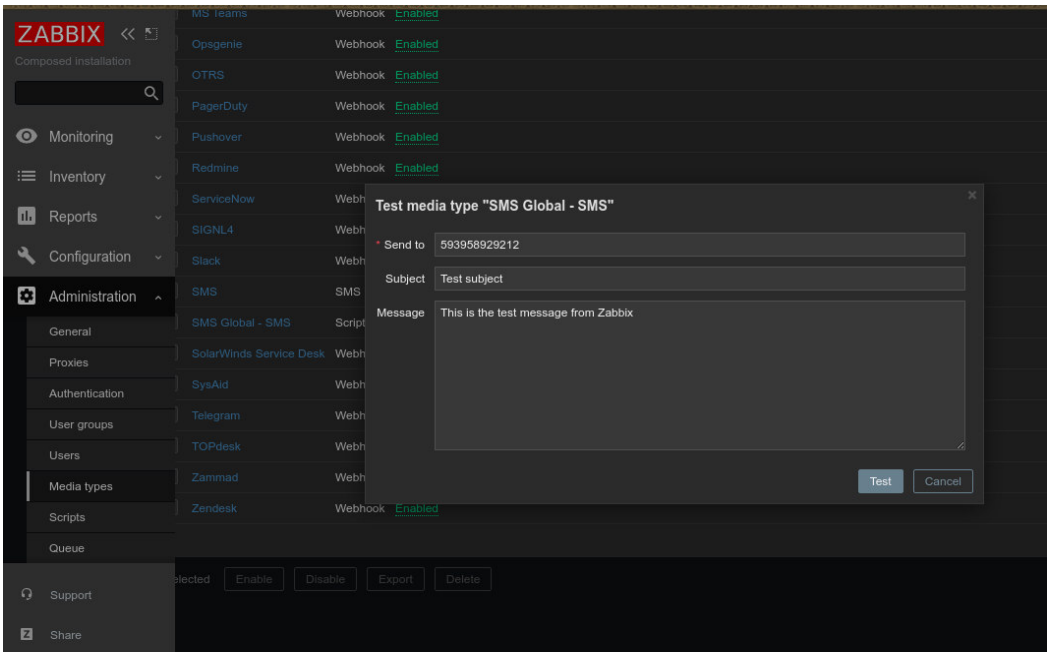


Figura 2.20 Verificación del script sms_send.sh en Zabbix

En la Figura 2.21 se muestra la verificación del *script* sms_send.sh en Zabbix.

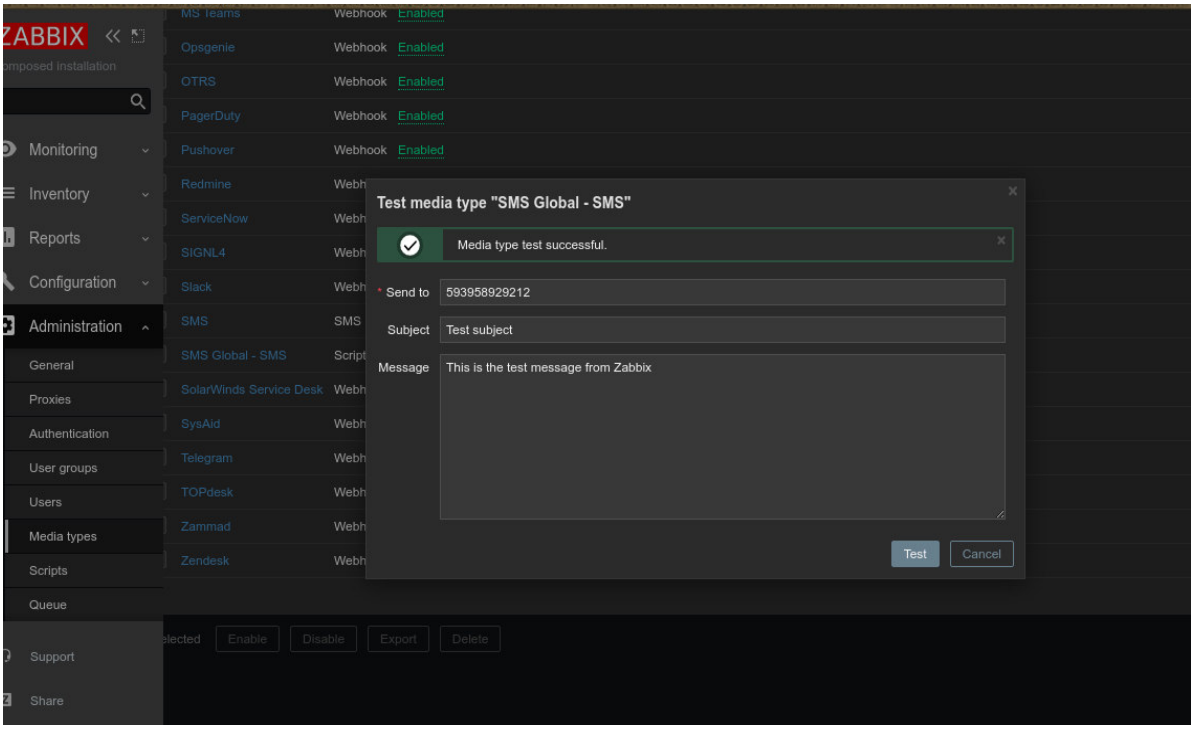


Figura 2.21 Verificación del correcto funcionamiento del script sms_send.sh en Zabbix

Al realizar el *test* se ve finalmente que resulta exitoso, ésta observando que en la aplicación SMSGlobal se ha intentado enviar el mensaje al número colocado en la configuración de Zabbix (Ver Figura 2.22), pero al no tener saldo en esta aplicación, ésta no puede enviar el mensaje a su destino. Una vez que se coloque saldo se podrá enviar sin ningún problema el mensaje a dicho destinatario.

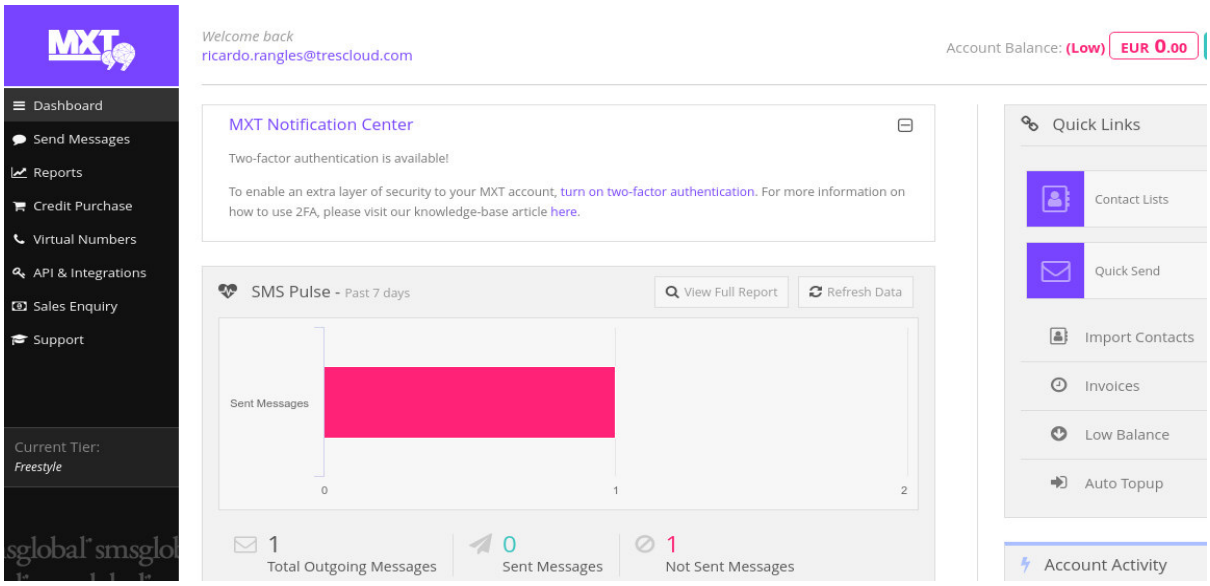


Figura 2.22 Envío de mensaje desde SMSGlobal al destinatario

Se debe tomar en cuenta que si no se dispone de saldo en la aplicación SMSGlobal no llegarán los mensajes al número que se configuró en el servidor Zabbix, en este caso si se prueba externamente saldrá error 88, que significa que no hay saldo suficiente; este error se puede observar en el *log* interno creado en el Docker con el fin de controlar lo que está sucediendo con el *script*, en este caso el *log* se encuentra en la ubicación: *tail -f /tmp/send_sms_log.out.* Para acceder al *log* primero se debe ingresar al contenedor que "corre" el servidor Zabbix, en la Figura 2.23 se muestra el error antes mencionado.

```
zadocker@zabbixdocker:~$ docker container exec -it zabbix-docker_zabbix-server_1 /bin/bash
zabbix@350cbd3b11ad:/var/lib/zabbix$ tail -f /tmp/send_sms_log.out
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    79  100    26  100    53     16     34  0:00:01  0:00:01 --:--:--    51
ERROR: 88 SMSGlobalMsgID:
/usr/lib/zabbix/alertscripts/send_sms.sh: 28: SMS Text sent to 593958929212: not found
This is the test message from Zabbix
```

Figura 2.23 Log para revisar el script sms_send.sh

2.9. FASE DE DISEÑO E IMPLEMENTACIÓN DEL SCRIPT PARA LA NOTIFICACIÓN DE FALLO DE UN CLIENTE ODOO

2.9.1. DISEÑO DEL SCRIPT PARA LA NOTIFICACIÓN DE FALLO DE UN CLIENTE ODOO

Para el diseño del *script* que permite notificar que un cliente Odoo ha caído o tiene problemas, primero se deben entender los requerimientos para que funcione dentro del sistema de monitoreo Zabbix y el sistema de gestión empresarial Odoo. A continuación se mencionan los requisitos para diseñar el *script*:

- Servidor Zabbix activo.
- Cliente Odoo.
- Uso del lenguaje de programación Python, XML y Bash.
- Editor para codificar el *script.*

Este *script* funciona de la siguiente manera: Cuando en el sistema de monitoreo Zabbix se genera el evento *"Problem name: Web scenario"* de cualquier cliente Odoo, se envía una notificación o aviso en el sistema Odoo al administrador correspondiente que algo ha sucedido con ese cliente Odoo, esto tiene como objetivo principal que se pueda avisar al usuario o administrador Odoo que ha pasado algo con algún cliente suyo y que se pueda solucionar el problema de inmediato para reiniciar el cliente Odoo.

Como se mencionó en el diseño del *script* para envío de notificaciones por vía SMS, dentro de las configuraciones de notificaciones en el sistema de monitoreo Zabbix para los usuarios se tendrá también la notificación de que un cliente Odoo cayó o tuvo problemas.

En la Figura 2.4 se muestra un diagrama de bloques para entender de mejor manera el *script* de notificación de fallo de un cliente Odoo.

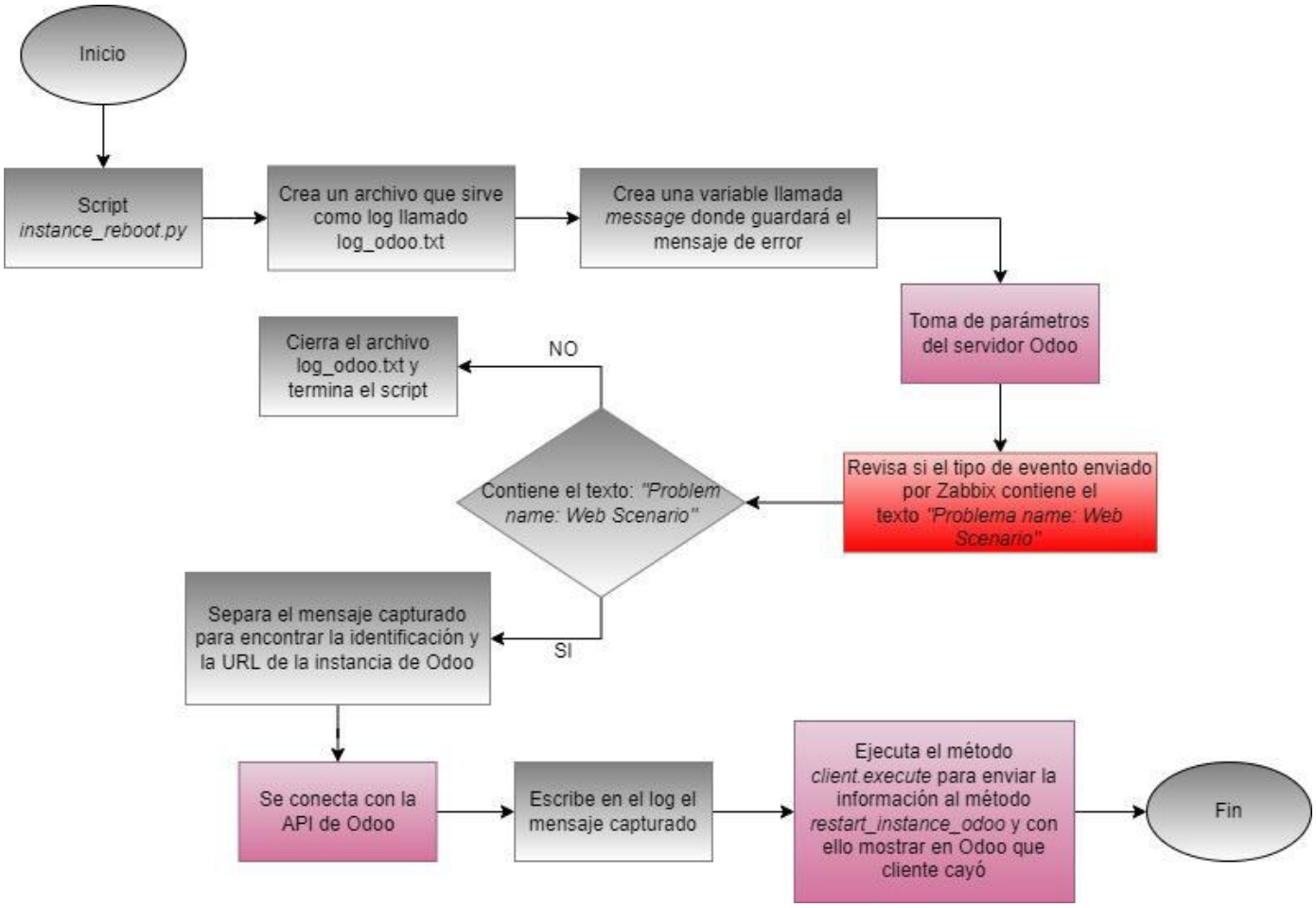


Figura 2.24 Diagrama de Bloques del script instance_reboot.py

2.9.2. IMPLEMENTACIÓN DEL SCRIPT PARA LA NOTIFICACIÓN DE FALLO DE UN CLIENTE ODOO

2.9.2.1. Creación del Script para la notificación de fallo de un cliente Odoo

Para codificar el *script* primero se modifica el archivo llamado *Dockerfile* que se lo ocupó previamente para el *script* de notificación vía SMS (ver Figura 2.25), esto con el fin de instalar *Python* y la librería *Erppeek,* los cuales son necesarios para realizar la notificación de que un cliente Odoo ha tenido algún problema.

```
FROM zabbix/zabbix-server-pgsql:ubuntu-5.2-latest

RUN apt-get update && apt-get install -y curl \
    python3-pip
RUN pip3 install erppeek
USER zabbix
```

Figura 2.25 Dockerfile modificado para que funcione el script instance_reboot.py

Se modifica el archivo Dockerfile para reconstruir la imagen Docker con los nuevos cambios de instalación de Python y Erpeek (ver Figura 2.26); realizado esto, se procede a crear el archivo *docker-compose_v3_ubuntu_pgsql_latest_trescloud.yaml* que construye la nueva imagen con los nuevos cambios en el servidor de Zabbix.

```
version: '3.5'
services:
 zabbix-server:
  image: zabbix-server-pgsql:ubuntu-5.2-latest-trescloud
  environment:
    - ZBX_CACHESIZE=64M
  logging:
    driver: syslog
    options:
#      syslog-address: "udp://XXX.papertrailapp.com:XXXX"
#      tag: "{{.Name}}/{{.ID}}"
      tag: "zabbix-server"
  ports:
   - "10051:10051"
  volumes:
   - /etc/localtime:/etc/localtime:ro
   - ./zbx_env/usr/lib/zabbix/alertscripts:/usr/lib/zabbix/alertscripts:ro
   - ./zbx_env/usr/lib/zabbix/externalscripts:/usr/lib/zabbix/externalscripts:ro
   - ./zbx_env/var/lib/zabbix/export:/var/lib/zabbix/export:rw
   - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
   - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
   - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
   - ./zbx_env/var/lib/zabbix/mibs:/var/lib/zabbix/mibs:ro
   - ./zbx_env/var/lib/zabbix/snmptraps:/var/lib/zabbix/snmptraps:ro
#   - ./.ZBX_DB_CA_FILE:/run/secrets/root-ca.pem:ro
#   - ./.ZBX_DB_CERT_FILE:/run/secrets/client-cert.pem:ro
#   - ./.ZBX_DB_KEY_FILE:/run/secrets/client-key.pem:ro
```

Figura 2.26 Parte de *docker-compose_v3_ubuntu_pgsql_latest_trescloud.yaml*

La creación del *script* se lo hace mediante el lenguaje de programación Python debido a que es más amigable y permite de manera más eficiente realizar la comunicación con la API de Odoo, esto con el fin de realizar la comunicación entre Zabbix y Odoo.

En el Código 2.11 se muestran las librerías utilizadas para la conexión entre Odoo y Zabbix.

```
1   # -*- coding: utf-8 -*-
2   import erppeek
3   import sys
4   import os
```

Código 2.11 Fragmento de las librerías para conectarse con el servidor Odoo

Una de las características para la implementación de este *script* es que se utiliza el API de Odoo para que se pueda enviar el fallo de un cliente desde Zabbix a Odoo (ver Código 2.12). Para ello se emplea el método *execute* (ver Código 2.13) que permite llamar al método *restart_instance_odoo* que se encuentra en el módulo de interconexión Odoo - Zabbix creado previamente.

```
#Odoo API connection
client = erppeek.Client(SERVER, DATABASE, USERNAME, PASSWORD, verbose=True)
```

Código 2.12 Fragmento del uso de la API de Odoo

```
#Method to execute the reset method of the zabbbix_odoo_library.py library
client.execute('zabbix.odoo.library','restart_instance_odoo',[],{},ID,URL)
```

Código 2.13 Fragmento del uso del método execute con la API de Odoo

Una vez culminada la codificación del *script*, dentro del servidor Zabbix se realiza la configuración de la creación de envío de notificación para un fallo de un cliente Odoo. Para ello se crea un *media types* llamado Reboot Odoo Instance dentro de la interfaz del servidor Zabbix (ver Figura 2.27).

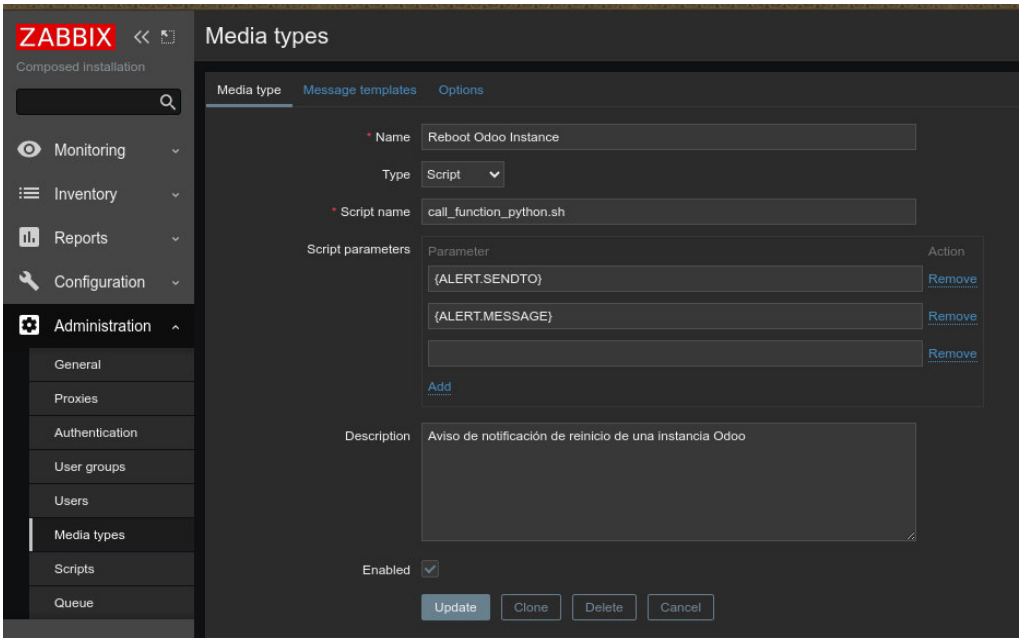


Figura 2.27 Configuración del media types Reboot Odoo Instance

Para que funcione el *script instance_reboot.py* hecho en Python se crea otro *script* en Bash llamado *call_function_python.sh* el cual sirve como intermediario para el *script* en Python, ya que Zabbix trabaja con *scripts* en Bash.

En el Código 2.14 se muestra una parte del código del *script call_function_python.sh*

```
#!/bin/sh
#
# Script to call the script made in python to restart an instance in odoo
#

#Write in log to check the values coming from Zabbix
echo "parametros que llegaron: $*" >>  /tmp/send_odoo_log.out
```

Código 2.14 Fragmento de los campos que llegan desde Zabbix al script de SMS

Una vez creado este *script* y configurado todo en el *media types* “Reboot Odoo Instance” se realiza la prueba para verificar si el *script* creado funciona correctamente.

Cuando un cliente Odoo cayó o tuvo problemas el servidor Zabbix notifica que ha sucedido algo con éste, por lo tanto envía todas las notificaciones configuradas previamente, entre ellas la de notificar que este cliente Odoo falló (ver Figura 2.28).

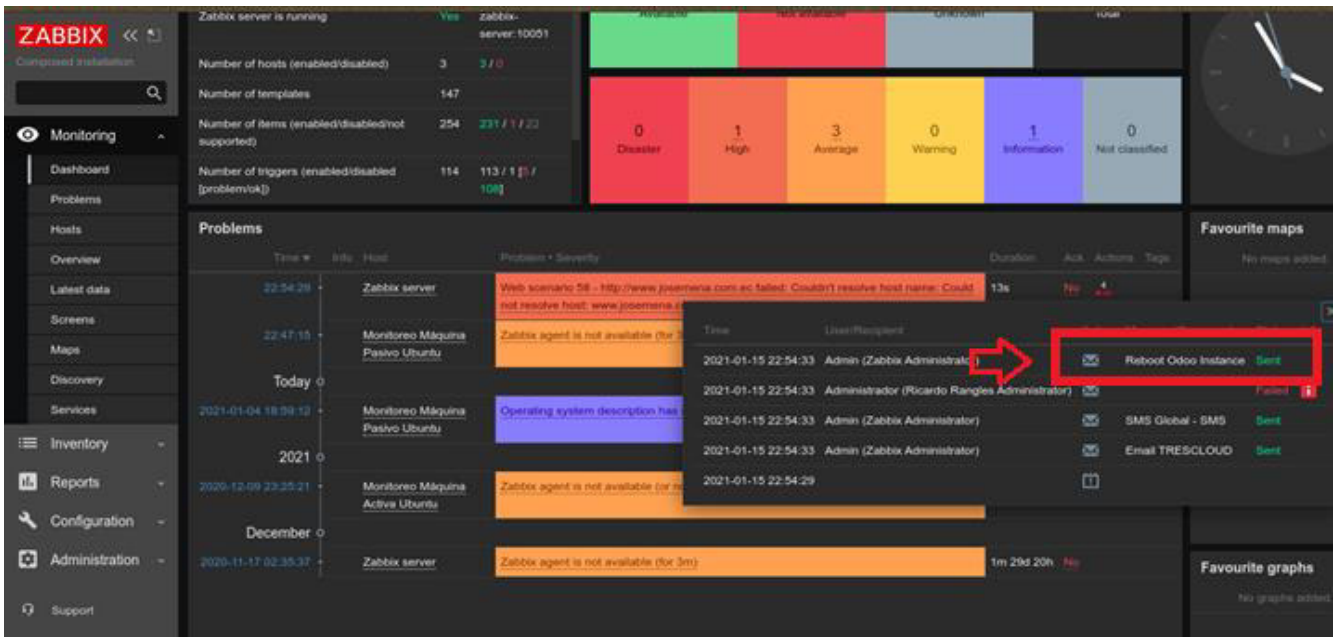


Figura 2.28 Notificación de fallo de un cliente Odoo en el servidor Zabbix

Cuando se envían todas las notificaciones sobre el fallo de un cliente Odoo, el *script call_function_python.sh* configurado en el *media types* Reboot Odoo Instance se ejecuta y en el servidor Odoo dentro del cliente Odoo que ha sufrido algún problema se notifica con fecha y hora que éste ha tenido algún fallo (ver Figura 2.29).

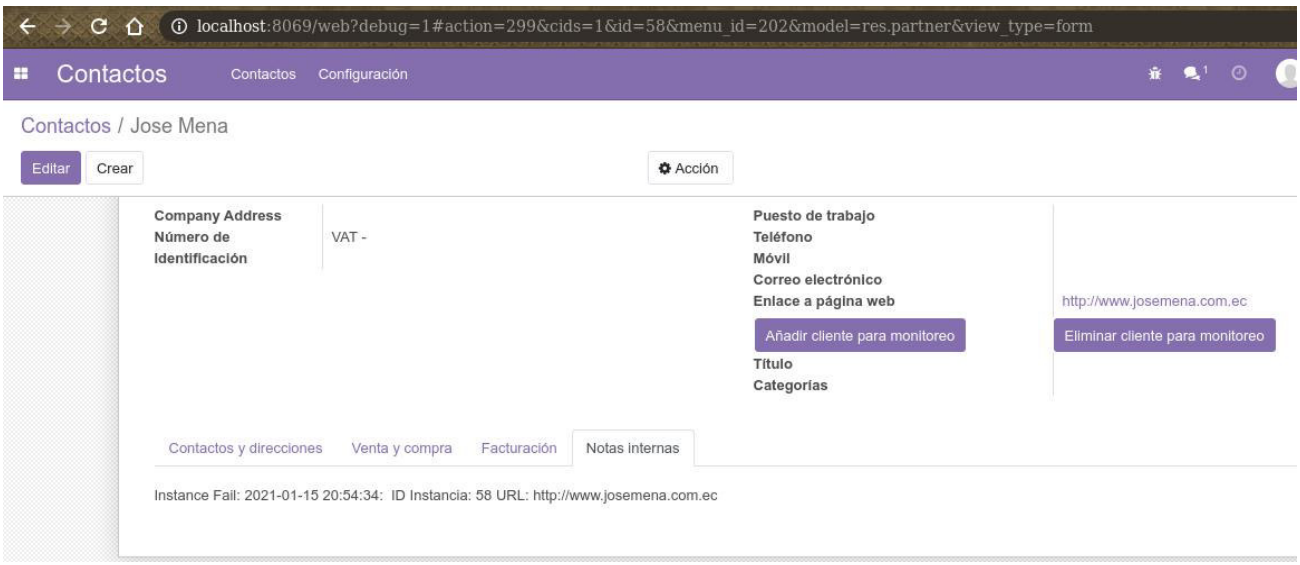


Figura 2.29 Notificación de fallo dentro del cliente Odoo fallido

Este *script* está pensado para notificar a clientes Odoo que sufren algún problema; es por este motivo que se codifica para ignorar las notificaciones provenientes de desperfectos o caídas de servidores y solo comunicarse en el caso de que la URL del cliente no responda. Para saber qué pasó con un servidor se informa mediante las notificaciones por vía *email* y SMS.

Queda abierto para futuras mejoras que el *script* notifique que un servidor Odoo cayó para que se pueda automáticamente realizar una acción.

Cabe recalcar que todas las notificaciones que se muestran en el servidor Zabbix pueden tener un desfase de 5 horas, esto se debe a que Linux se instala en UTC (*Universal Time Coordinated*) y el sistema operativo por debajo hace la transformación de UTC al uso horario de Ecuador si el usuario de Zabbix lo tiene así configurado. En el caso de que el usuario no esté adecuadamente configurado se tendrá este desfase.

2.10. IMPLEMENTACIÓN FINAL DEL SISTEMA DE MONITOREO

La implementación final del sistema de monitoreo está basada en tener 4 servidores Odoo en la nube junto con el servidor Zabbix también ubicado en la nube, todo esto con el fin de hacer todas las pruebas necesarias para indicar que el sistema de monitoreo y el módulo de interconexión Odoo - Zabbix funcionan correctamente. Cabe mencionar que para las pruebas de funcionamiento solo se monitorearán 3 de los 4 servidores Odoo.

2.10.1. INSTALACIÓN DEL SERVIDOR ZABBIX EN AWS

Inicialmente se instaló el servidor Zabbix en la nube de AWS (ver Figura 2.30); para instalar el software Zabbix se lo hizo de la misma manera que para instalar el servidor de prueba Zabbix. Más información acerca de esta instalación se la detalla en el Anexo D.

Figura 2.30 Servidor Zabbix en AWS

2.10.2. INSTALACIÓN DE LOS SERVIDORES ODOO EN AWS

Para tener el sistema completo se instalaron 4 servidores Odoo en la nube AWS, la creación de estos servidores en AWS está detallada en el Anexo D. Una vez creados los servidores, se les instaló Odoo de la misma manera que se lo hizo para implementar el servidor de prueba Odoo para hacer el módulo de interconexión Odoo - Zabbix.

Cabe mencionar que uno de estos 4 clientes Odoo servirá como un servidor Odoo especial en donde se va a configurar todo lo necesario para la comunicación entre Zabbix y Odoo de este sistema final; este servidor especial no será monitoreado y además tendrá instalado 2 módulos para las pruebas de funcionamiento, el módulo de interconexión Zabbix – Odoo y el módulo prueba botones.

En las Figuras 2.31, 2.32, 2.33 se muestran la creación de los 3 servidores Odoo que serán monitoreados en Zabbix.

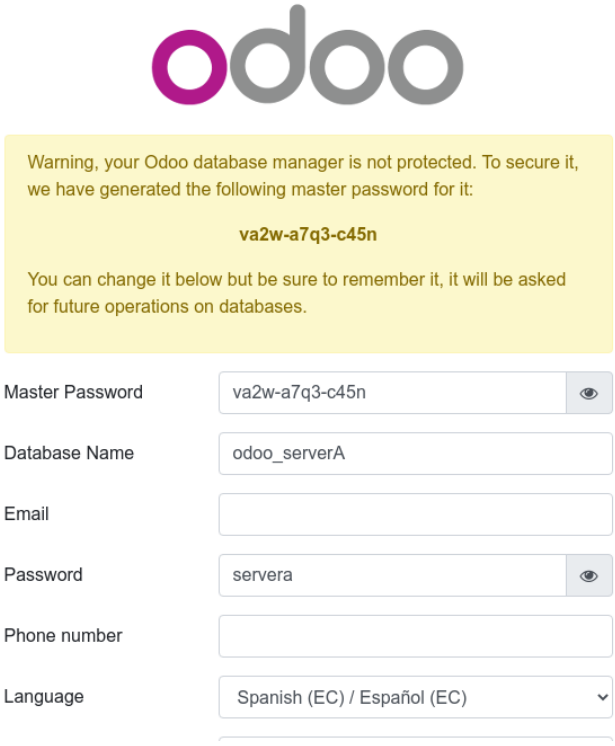


Figura 2.31 Servidor A de Odoo en AWS

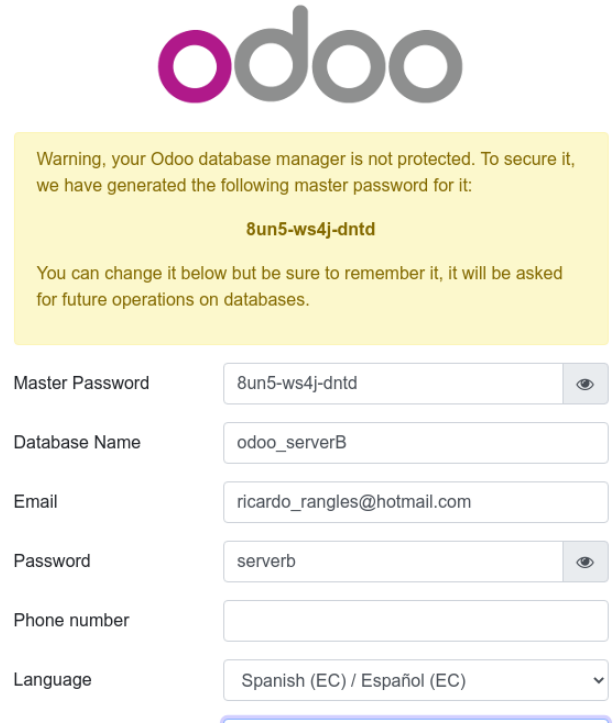


Figura 2.32 Servidor B de Odoo en AWS

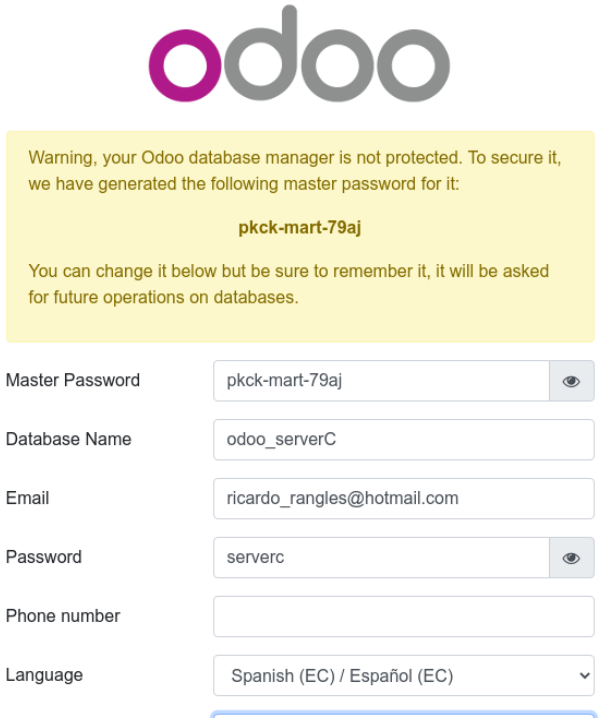


Figura 2.33 Servidor C de Odoo en AWS

En la Figura 2.34 se muestra la interfaz Odoo de los 3 servidores y el servidor especial.

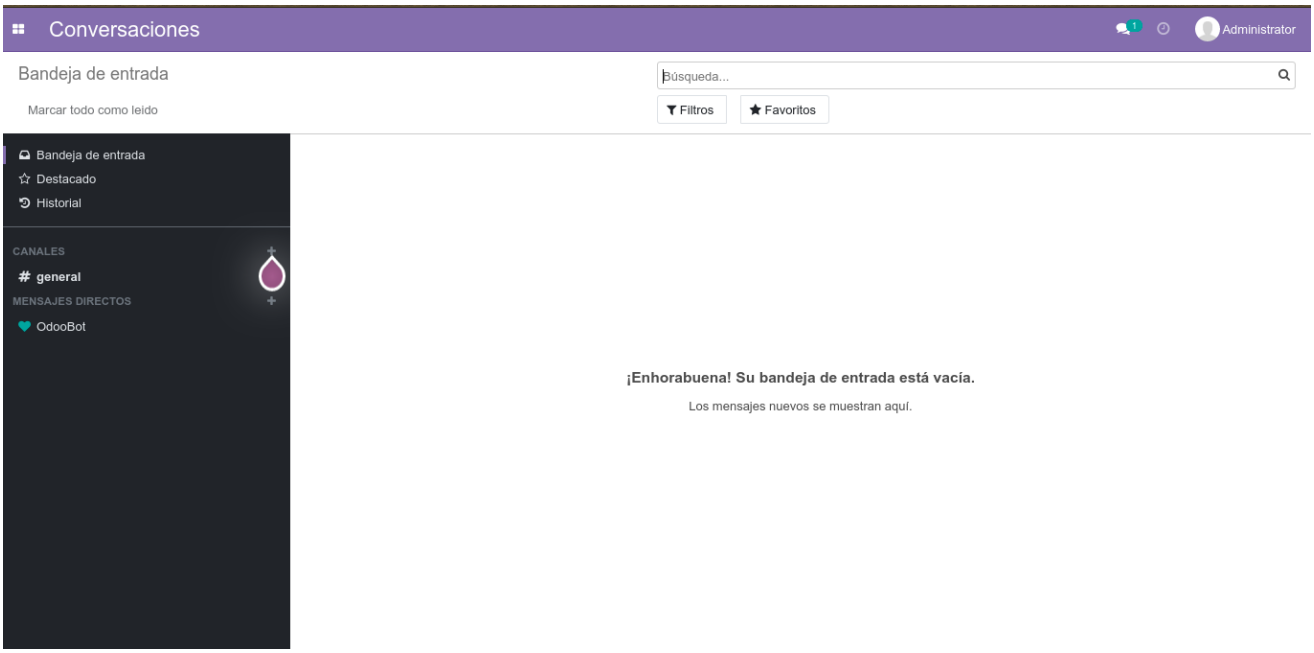


Figura 2.34 Interfaces de los servidores A,B, C y servidor especial de Odoo

Una vez instalado Odoo en los 3 clientes mencionados, en el servidor Odoo especial se crean los 3 clientes llamados *Cliente Odoo A, Cliente Odoo B y Cliente Odoo C* (ver Figura 2.35) los cuales serán monitoreados en el sistema de monitoreo Zabbix.

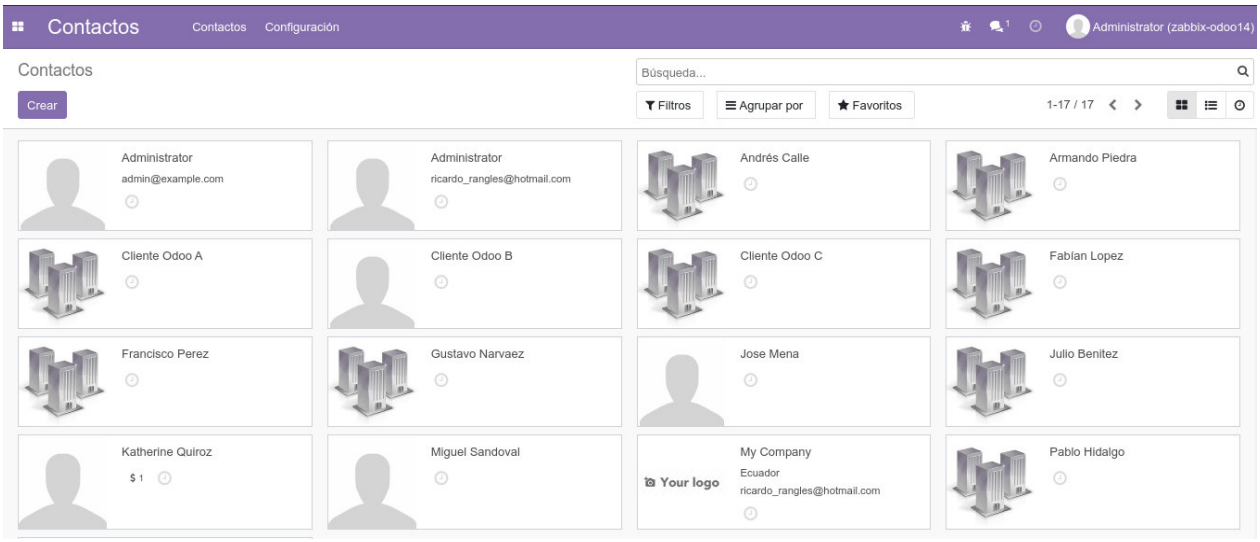


Figura 2.35 Creación de 3 clientes en Odoo para el monitoreo en Zabbix

La información sobre los usuarios y contraseñas que se manejan en el sistema de monitoreo que se acaba de implementar, se detalla en el Anexo B.

Una vez creados los servidores en la nube tanto de Zabbix como de Odoo, se realizan todas las pruebas de funcionamiento sobre el sistema de monitoreo y el módulo de interconexión Odoo - Zabbix, esto se lo revisa en el Capítulo III.

Más información de la codificación del módulo de interconexión Odoo – Zabbix se detalla en el Anexo C.

Para más información sobre cómo se realizó la implementación de todo el sistema de monitoreo en Zabbix, junto con la implementación del módulo de interconexión Odoo – Zabbix referirse al Anexo D.

CAPÍTULO III

CAPÍTULO III

3. RESULTADOS Y DISCUSIÓN

En el presente capítulo se exponen los resultados de las distintas pruebas aplicadas al sistema de monitoreo. Se realizan pruebas de funcionamiento de las configuraciones en la plataforma Zabbix, del módulo de interconexión Odoo - Zabbix, de las notificaciones que se envían al momento que un cliente o servidor Odoo tiene un problema y de las pruebas de funcionamiento de la notificación que un cliente Odoo cayó o tuvo problemas.

3.1. PRUEBAS DE FUNCIONAMIENTO DEL SISTEMA DE MONITOREO

3.1.1. PRUEBAS DE FUNCIONAMIENTO DEL MONITOREO DE UN CLIENTE ODOO EN LA PLATAFORMA ZABBIX

Las configuraciones que se hicieron en la plataforma Zabbix toma en cuenta solo las configuraciones necesarias para el desarrollo del proyecto, es decir no se realizaron cambios en varios parámetros que ofrece Zabbix para su funcionamiento.

En esta sección se presentan las pruebas de funcionamiento del monitoreo pasivo, que se encuentra configurado en cada cliente Odoo previamente creado en la nube AWS; también se presentan pruebas de simulación de carga al CPU en uno de los clientes Odoo.

En la Tabla 3.1 se indican todas las pruebas realizadas durante el monitoreo de un cliente Odoo en la plataforma Zabbix.

Tabla 3.1. Tipos de Pruebas en las configuraciones generales entre Odoo y Zabbix

Prueba	Definición
Prueba de Conectividad	Verifica la conexión entre el cliente Odoo con el servidor Zabbix.
Prueba de Obtención de Datos	Indica la obtención de las últimas métricas del cliente Odoo monitoreado.
Prueba de Carga del Sistema	Indica la carga del sistema del cliente Odoo monitoreado.
Prueba de Uso del CPU	Indica el uso del CPU del cliente Odoo monitoreado.
Prueba de Utilización de Memoria	Indica la utilización de memoria del cliente Odoo monitoreado.
Prueba de Disponibilidad de Memoria	Indica la disponibilidad de memoria que tiene el cliente Odoo monitoreado.
Prueba de Tráfico de Red	Indica el tráfico entrante y saliente de red que está ocupando el cliente Odoo monitoreado.

3.1.1.1. Pruebas de funcionamiento del Monitoreo Pasivo en los clientes Odoo en la nube

Para este tipo de pruebas primero se instala el agente de Zabbix en el servidor de cada cliente Odoo; una vez instalado se crea una llave PSK para poder tener una comunicación cifrada entre el cliente y el servidor Zabbix. A continuación se configura cada agente en modo pasivo seteando la IP del servidor de monitoreo Zabbix, para permitir la comunicación únicamente a este servidor. Todo este proceso se detalla en el Anexo D.

En la Figura 3.1 se muestra un ejemplo de configuración del agente pasivo de un cliente Odoo.

AGENTE ZABBIX SERVER A

Hostname=server-odoo-a

Server=192.168.0.10,127.0.0.1

TLSConnect=psk

TLSAccept=psk

TLSPSKIdentity=PSK 004

TLSPSKFile=/home/ubuntu/zabbix_agent/zabbix_agentd.psk

Figura 3.1 Ejemplo de configuración del agente pasivo de un cliente Odoo

Una vez configurado el agente, se configura el *host* en Zabbix para que lo pueda monitorear. Para la prueba de funcionamiento se toma en cuenta el cliente llamado *"Server Client A";* una vez configurado el *host* con todos sus parámetros, se verifica que estén comunicándose entre sí (ver Figura 3.2), es decir que Zabbix pueda tener toda la información de monitoreo de este *host*.

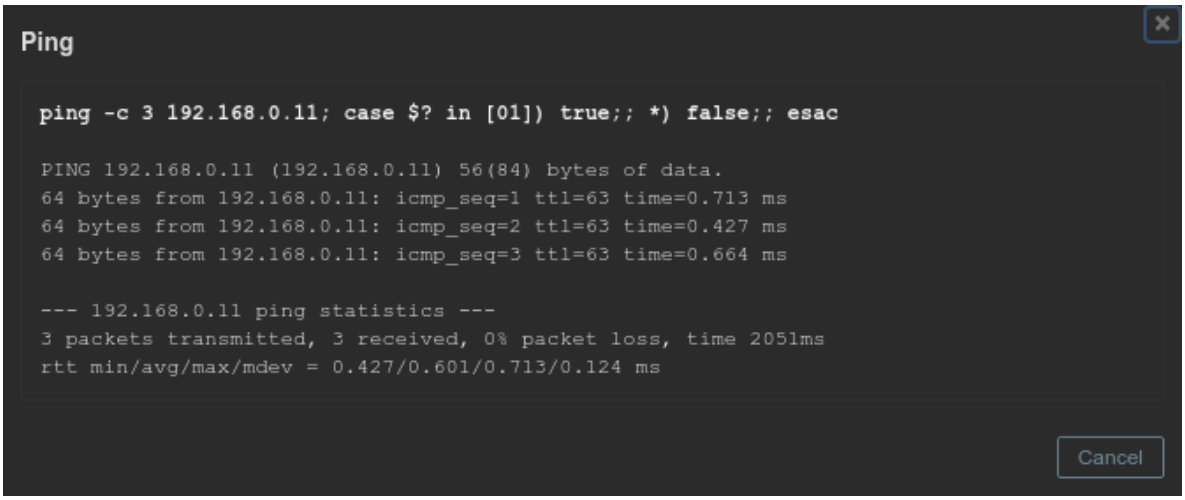


Figura 3.2 Prueba de conectividad entre Zabbix y Server Client A

En la Figura 3.3 se muestra el servidor *Server Client A* que está siendo monitoreado de manera correcta dentro de Zabbix.

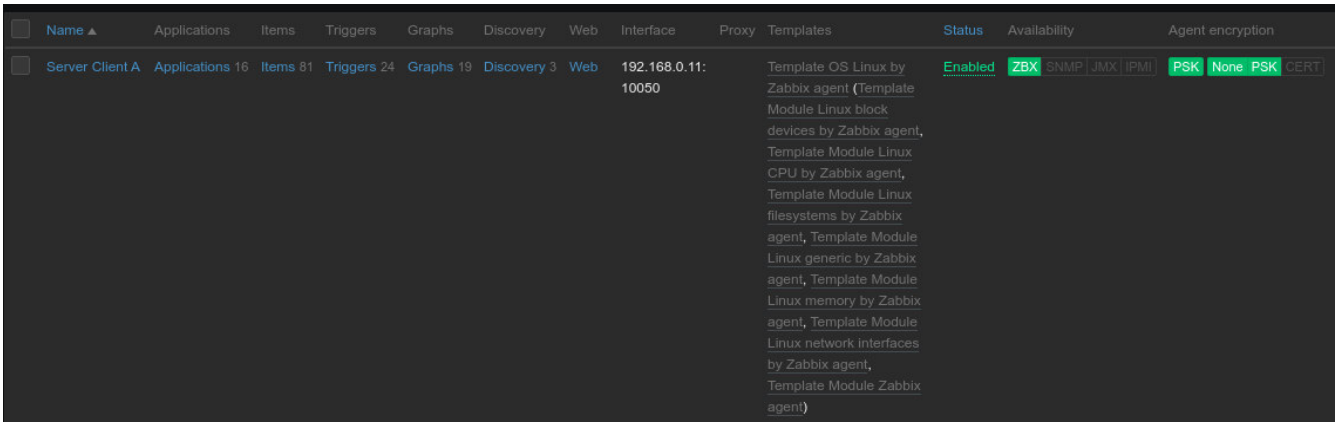


Figura 3.3 Prueba de funcionamiento de monitoreo pasivo en Server Client A

En la Figura 3.4 se indican los últimos datos obtenidos (métricas) del servidor *Server Client A* una vez iniciado el monitoreo del mismo.

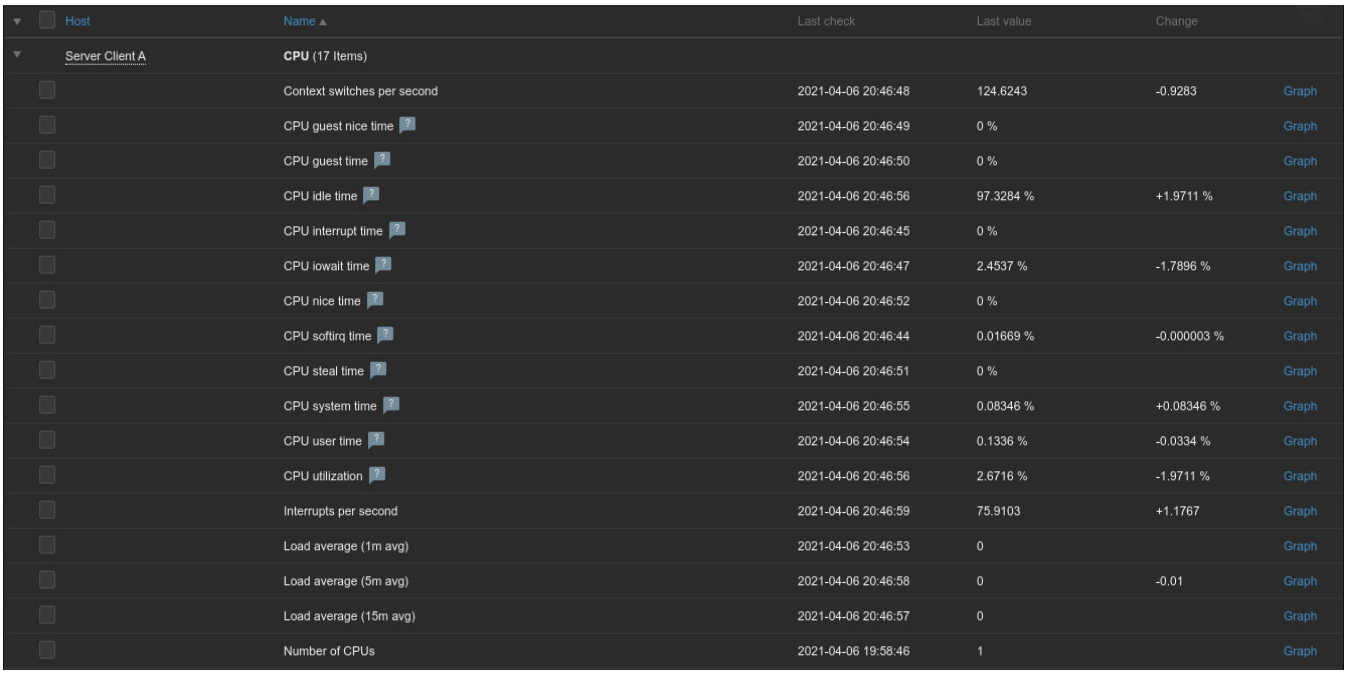


Figura 3.4 Últimos datos del cliente Server Client A

En la Figura 3.5 se indican los problemas que fueron detectados en el cliente *Server Client A* durante su monitoreo.

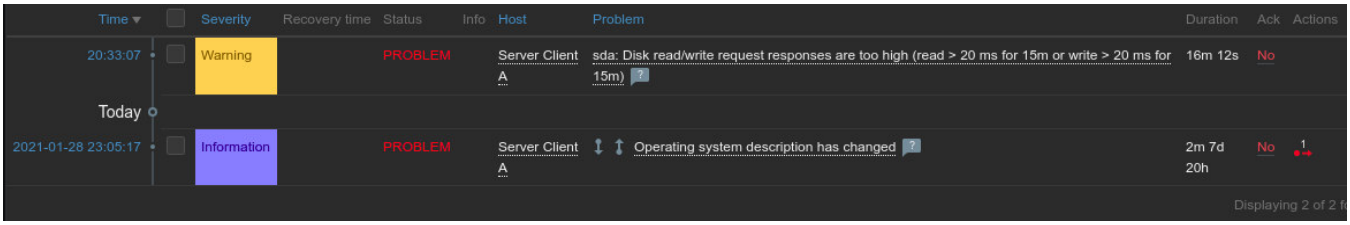


Figura 3.5 Problemas que detecta Zabbix del cliente Server Client A

En la Figura 3.6 se indica la carga promedio en 1, 5 y 15 minutos que tiene el cliente *Server Client A* monitoreado en un intervalo de tiempo de 1 hora.

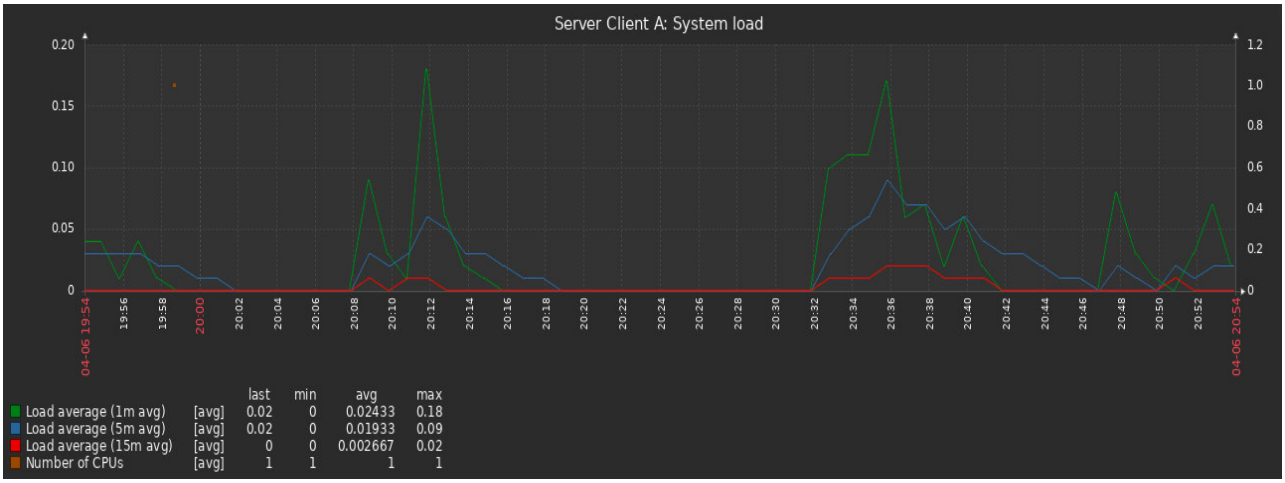


Figura 3.6 Gráfica de la métrica carga del sistema del cliente Server Client A

En la Figura 3.7 se presenta el uso del CPU del servidor monitoreado dividido en métricas propias de Linux, mostrando los valores último, mínimo, promedio y máximo en un rango de 1 hora.

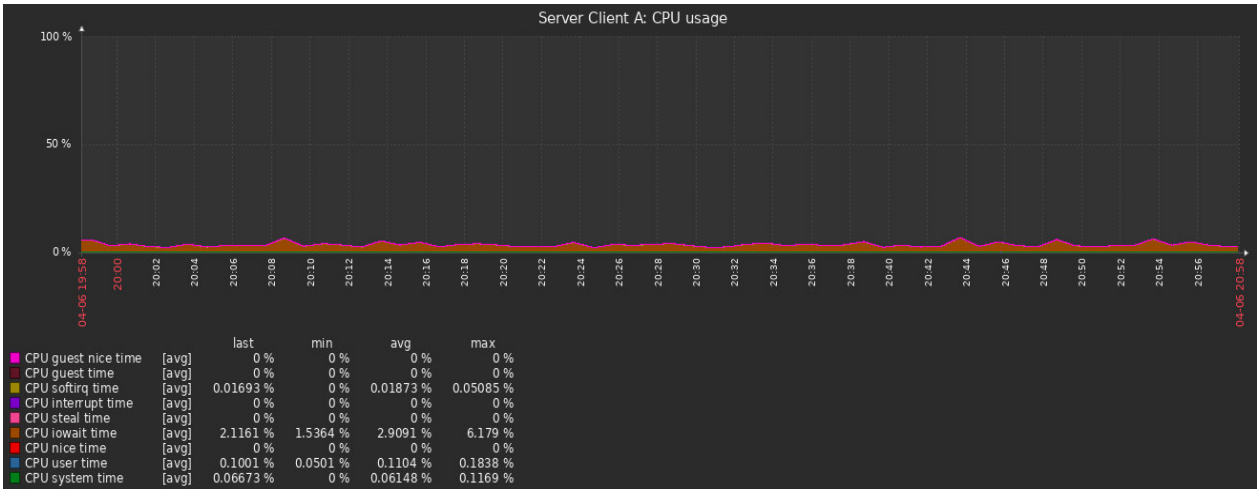


Figura 3.7 Gráfica de la métrica uso del CPU del cliente Server Client A

En la Figura 3.8 se muestra la disponibilidad de memoria del servidor monitoreado en el rango de 1 hora; en este caso se indica que la memoria disponible del cliente *Server Client A* es de 1.5 GB.

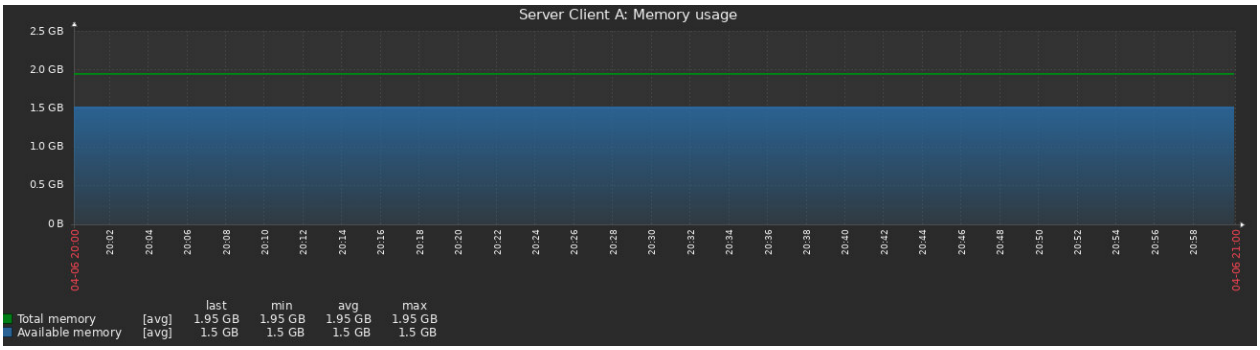


Figura 3.8 Gráfica de la métrica de disponibilidad de memoria del cliente Server Client A

En la Figura 3.9 se indica el tráfico de red que tiene el servidor monitoreado en la interfaz estándar en el rango de 1 hora.

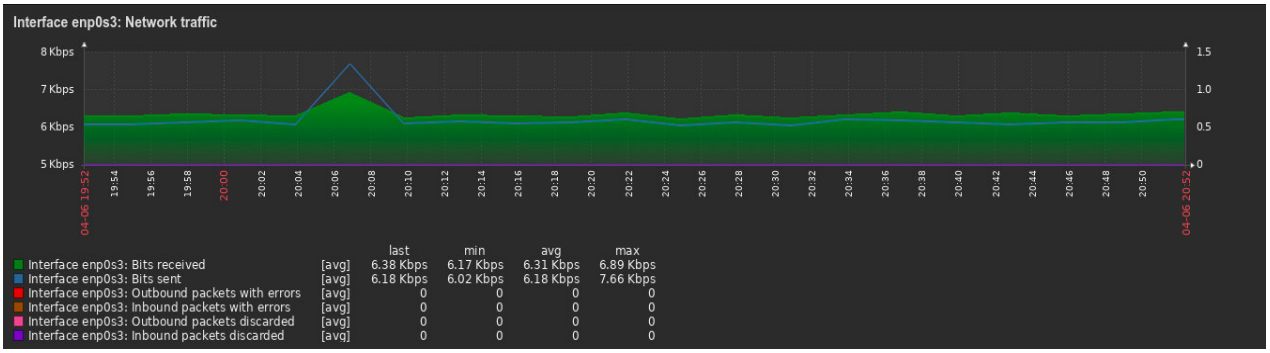


Figura 3.9 Gráfica del tráfico de la red del cliente Server Client

Más información sobre las configuraciones realizadas para el sistema de monitoreo puede encontrar en el Anexo D.

3.1.1.2. Prueba de funcionamiento: Simulación de carga de CPU sobre el servidor de un cliente Odoo en la nube

Esta prueba se la realiza para comprobar que el sistema de monitoreo Zabbix puede monitorear un cliente Odoo que trabaje "bajo presión". Para este caso se emplea al cliente *"Server Client A"* que permitirá hacer la prueba como un cliente Odoo real de la empresa TRESCLOUD CIA LTDA, con el fin de monitorear su actividad cuando se lo sobrecarga con procesos.

Para este escenario al cliente "*Server Client A"* se le sobrecarga de procesos al CPU durante 3 minutos.

Más información de cómo simular carga al CPU en un cliente Odoo se detalla en el Anexo D.

En la figura 3.10 se muestra cómo afecta la sobrecarga de procesos al servidor A de Odoo; el rango es de 24 horas.

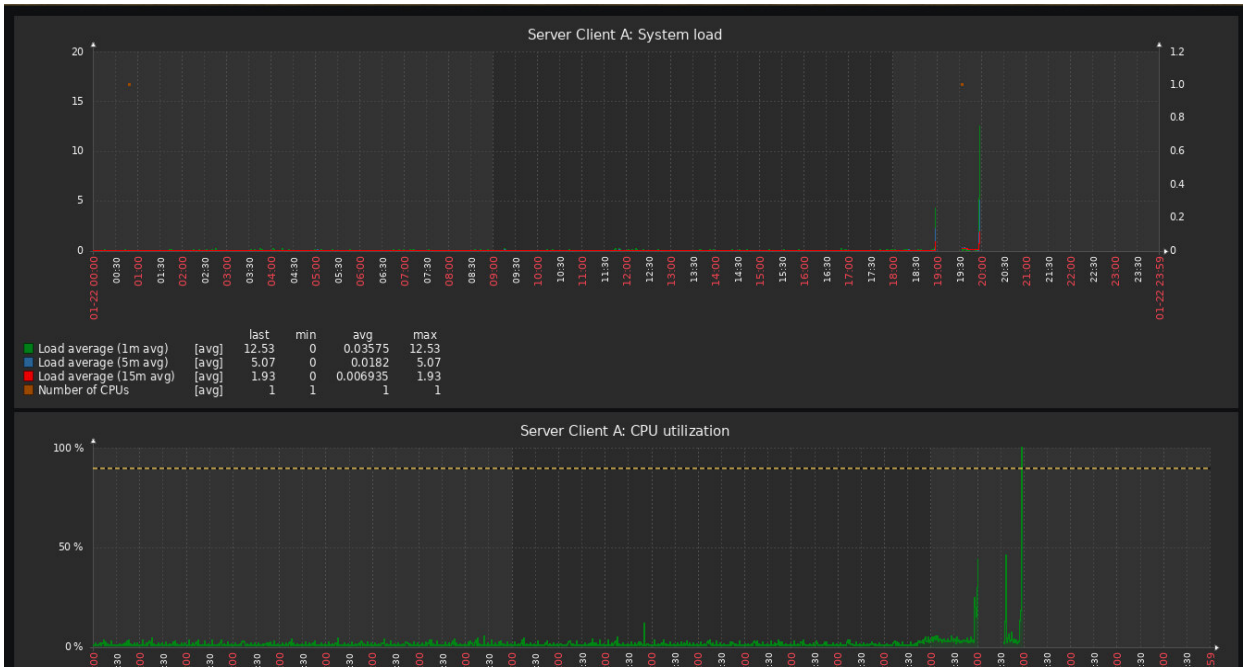


Figura 3.10 Métrica de carga del sistema y utilización del CPU del Client A

En la Figura 3.11 se indica el uso de CPU del Cliente A en un rango de 24 horas.

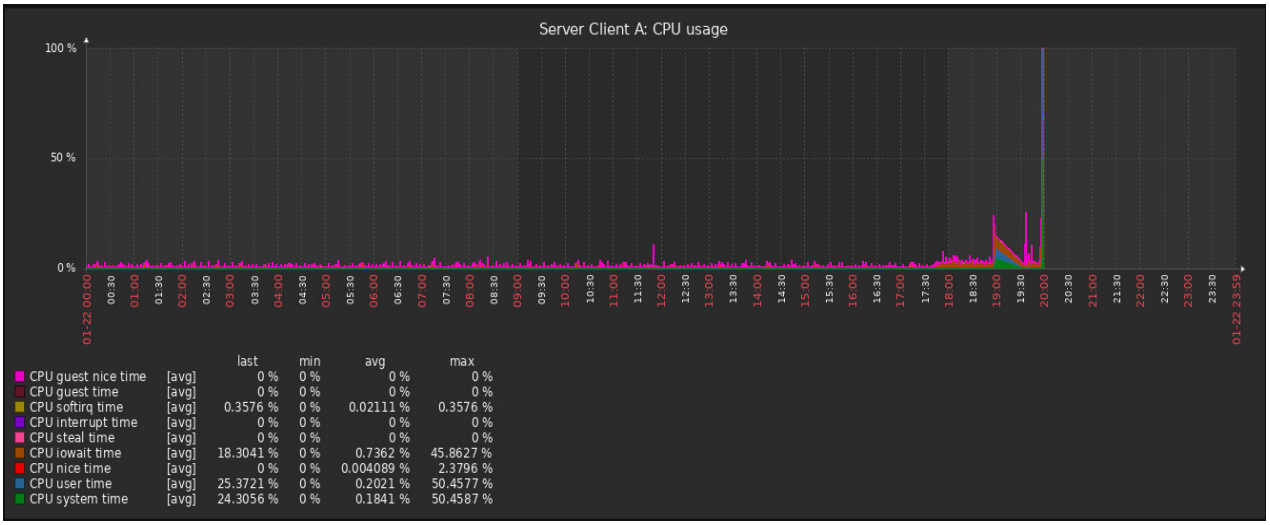


Figura 3.11 Métrica de uso del CPU del Server Client A

En la Figura 3.12 se indica el porcentaje de utilización de memoria del Cliente A en un rango de 24 horas.

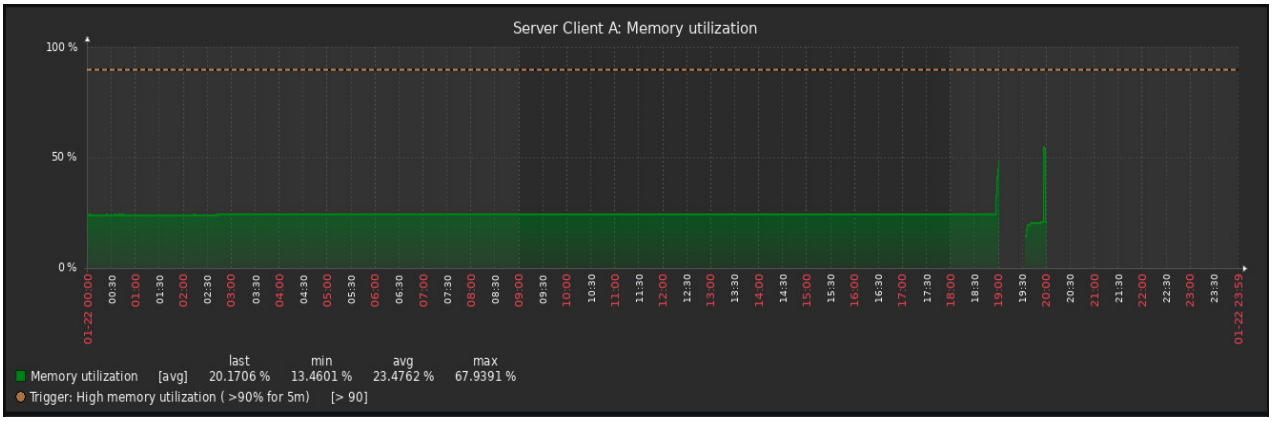


Figura 3.12 Métrica de la utilización de memoria del Server Client A

En la Figura 3.13 se indica se indica la disponibilidad de memoria que tiene el Cliente A en el rango de 24 horas.

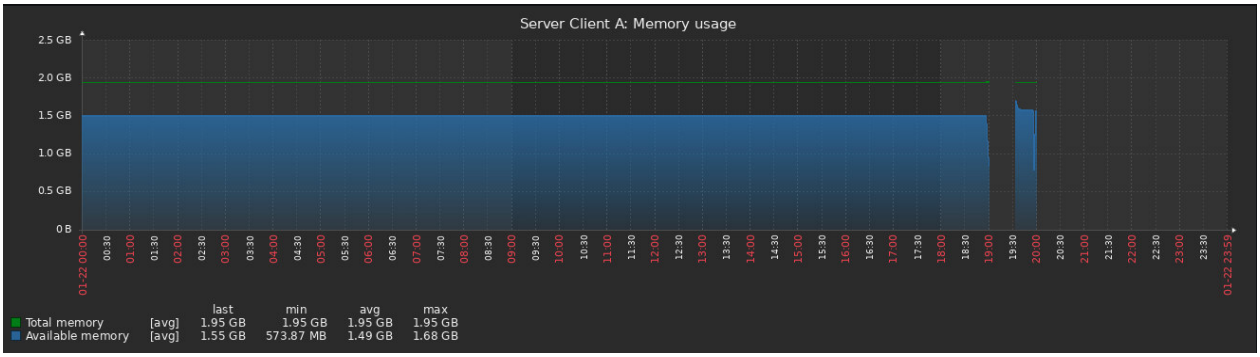


Figura 3.13 Métrica de disponibilidad de memoria del Server Client A

3.1.2. PRUEBAS DE FUNCIONAMIENTO DEL MÓDULO DE INTERCONEXIÓN ODOO – ZABBIX

En la Tabla 3.2 se indican todas las pruebas utilizadas durante el funcionamiento del módulo de interconexión Odoo – Zabbix.

Tabla 3.2. Tipos de Pruebas del funcionamiento del módulo Odoo - Zabbix

Prueba	Definición
Pruebas de Validación	Validan todas las excepciones que se presentan en la conexión entre Odoo y Zabbix, así como las validaciones cuando se crean y se eliminan los clientes en Zabbix.
Prueba de Creación de Clientes Odoo	Verifica la creación de un cliente Odoo desde su interfaz al sistema de monitoreo Zabbix.
Prueba de Eliminación de Clientes Odoo	Verifica la eliminación de un cliente Odoo desde su interfaz al sistema de monitoreo Zabbix.

Para realizar las pruebas de funcionamiento del módulo de interconexión Odoo - Zabbix se crea un módulo de prueba llamado *modulo_prueba_botones*, el cual permite mostrar el funcionamiento del módulo con todas las funcionalidades creadas previamente.

En la Figura 3.14 se muestra el archivo *_manifest_.py* que indica todo lo relacionado a este módulo de prueba.

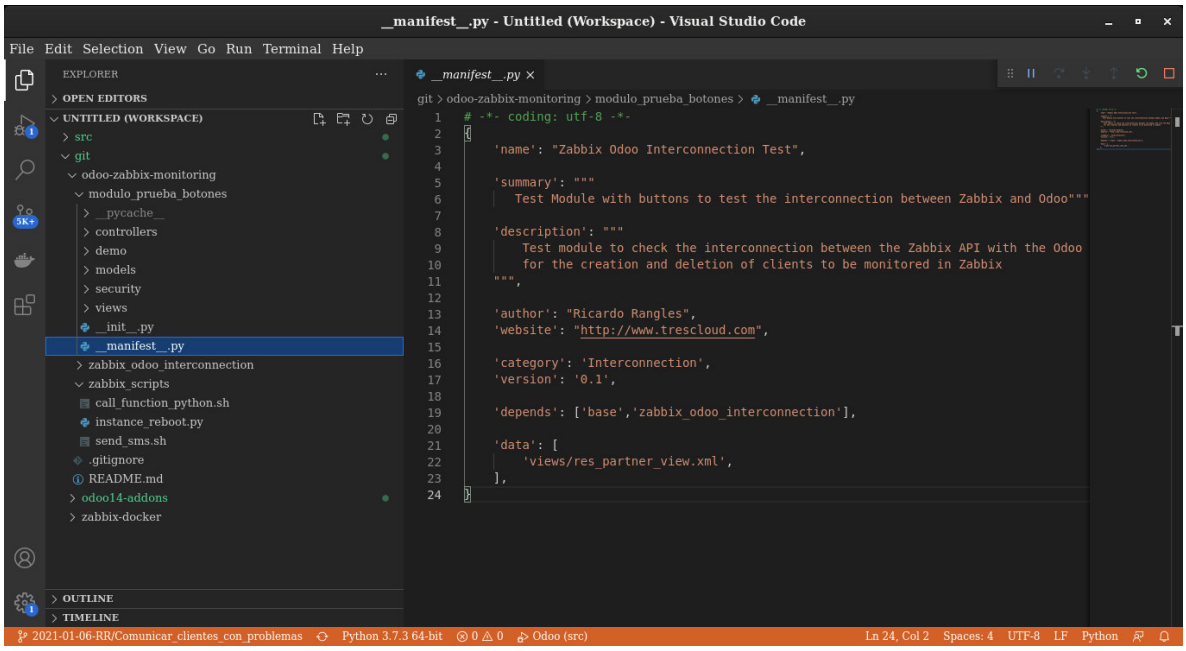


Figura 3.14 Archivo *_manifest_.py* del módulo modulo_prueba_botones

Para construir el módulo primero se crea el archivo *res_partner.py*, que contendrá, el código que servirá para llamar a las funciones del módulo de interconexión Odoo - Zabbix dentro del archivo *zabbix_odoo_library.py* creado previamente (Ver Código 3.1).

```
6   class ResPartner(models.Model):
7
8       #Odoo res.partner module inheritance
9       _inherit = 'res.partner'
10
11      def create_client_button(self):
12          """
13          This function calls the create a client function to monitor in zabbix.
14          """
15          create_client_function = self.env["zabbix.odoo.library"].create_monitor_client(self.id, self.website)
16
17          if create_client_function == 0:
18              raise ValidationError("The URL already exists in Zabbix")
19          if create_client_function == 1:
20              raise ValidationError("The URL has been added correctly in Zabbix")
21          if create_client_function == 2:
22              raise ValidationError("The URL doesn't match the one sent in Zabbix")
```

Código 3.1 Fragmento de la llamada a la función crear_cliente_monitoreo

Al igual que al desarrollar el módulo de interconexión Odoo - Zabbix se ubica el nombre del archivo res_partner.py (ver Figura 3.15) que se encuentra en la carpeta *models*, dentro del archivo *_init_.py* ya que si no se lo hace no funcionará el módulo y no se podrá instalar el mismo en Odoo.

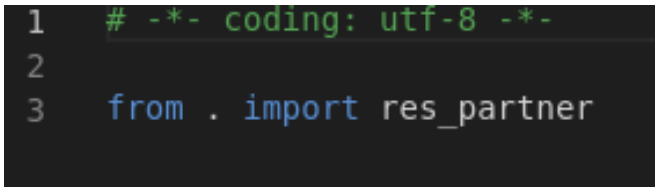


Figura 3.15 Archivo _init_.py del módulo modulo_prueba_botones

Para realizar la prueba de la interconexión Odoo - Zabbix, en la interfaz Odoo se crea un archivo llamado *res_partner_view.xml* dentro de la carpeta *views* (ver Código 3.2), que sirve para diseñar la vista y la creación de botones, para agregar y eliminar clientes al sistema de monitoreo en Zabbix.

```
<field name="arch" type="xml">
  <field name="website" position="after">
    <button string="Añadir cliente para monitoreo" name="create_client_button" type="object" class="oe_highlight"/>
    <button string="Eliminar cliente para monitoreo" name="remove_client_button" type="object" class="oe_highlight"/>
  </field>
</field>
```

Código 3.2 Fragmento de la creación de vista de botones agregar y eliminar

Una vez finalizada la creación del módulo *modulo_prueba_botones* se lo instala en el servidor de prueba de Odoo previamente creado, para ello se utiliza la interfaz de Odoo, con esto ya se pueden realizar las pruebas de agregar y eliminar clientes al servidor de monitoreo Zabbix.

Para su instalación se debe ir a la pestaña de *Aplicaciones* y buscarlo por el nombre de módulo *modulo_prueba_botones* y luego da clic en el botón Instalar, luego de lo cual se obtiene el resultado que se muestra en la Figura 3.16.

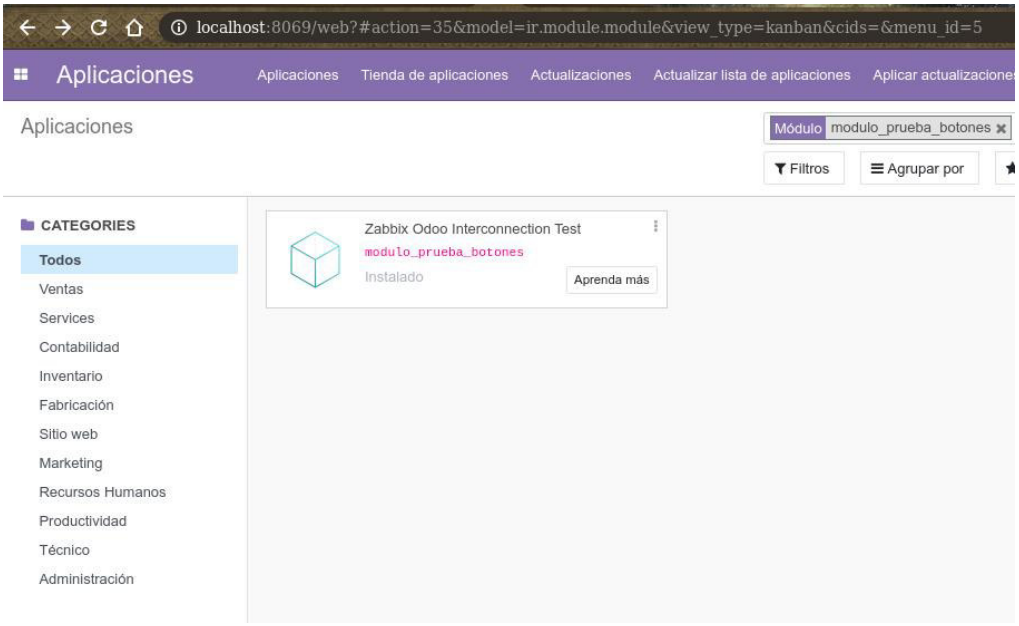


Figura 3.16 Instalación del módulo modulo_prueba_botones en el servidor Odoo

En la Figura 3.17 se indican las especificaciones que tiene el módulo modulo_prueba_botones creado para las pruebas de creación y eliminación de clientes Odoo en Zabbix.

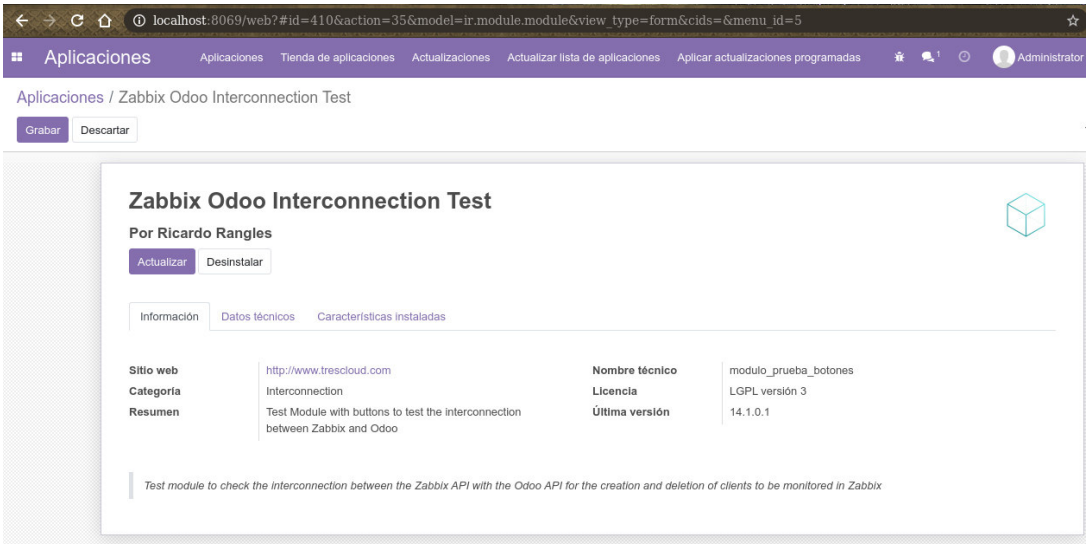


Figura 3.17 Especificaciones del módulo modulo_prueba_botones

En la Figura 3.18 se muestra el módulo *modulo_prueba_botones* instalado con los botones para crear y eliminar clientes Odoo en Zabbix.

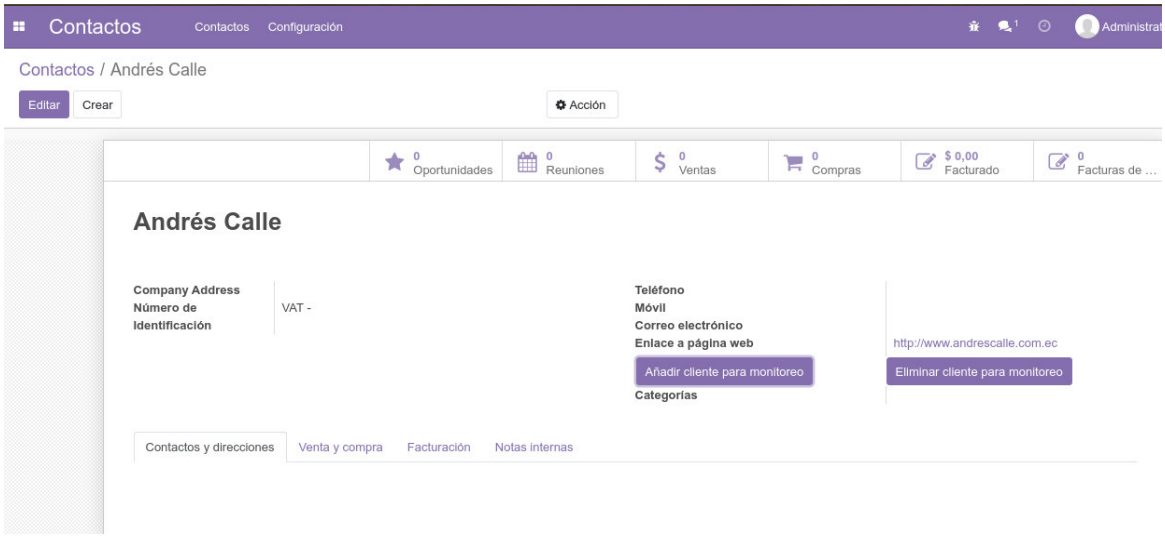


Figura 3.18 Botones creados para crear o eliminar clientes al monitoreo en Odoo

Para hacer la prueba del módulo de interconexión Odoo - Zabbix de manera completa se usa el cliente *Server Client B,* en él se realizan las pruebas necesarias con el fin de demostrar el correcto funcionamiento del módulo de Interconexión.

3.1.2.1. Pruebas de conexión con el servidor Zabbix desde Odoo

En primer lugar se realiza la prueba de conexión al servidor Zabbix al cual se va a conectar el cliente *Server Client B* para ser monitoreado. En el módulo de interconexión existen excepciones cuando no se puede conectar al servidor Zabbix.

En las Figuras 3.19, 3.20, 3.21 y 3.22 se muestran las excepciones que tiene el sistema Odoo cuando no puede conectarse con el servidor Zabbix.

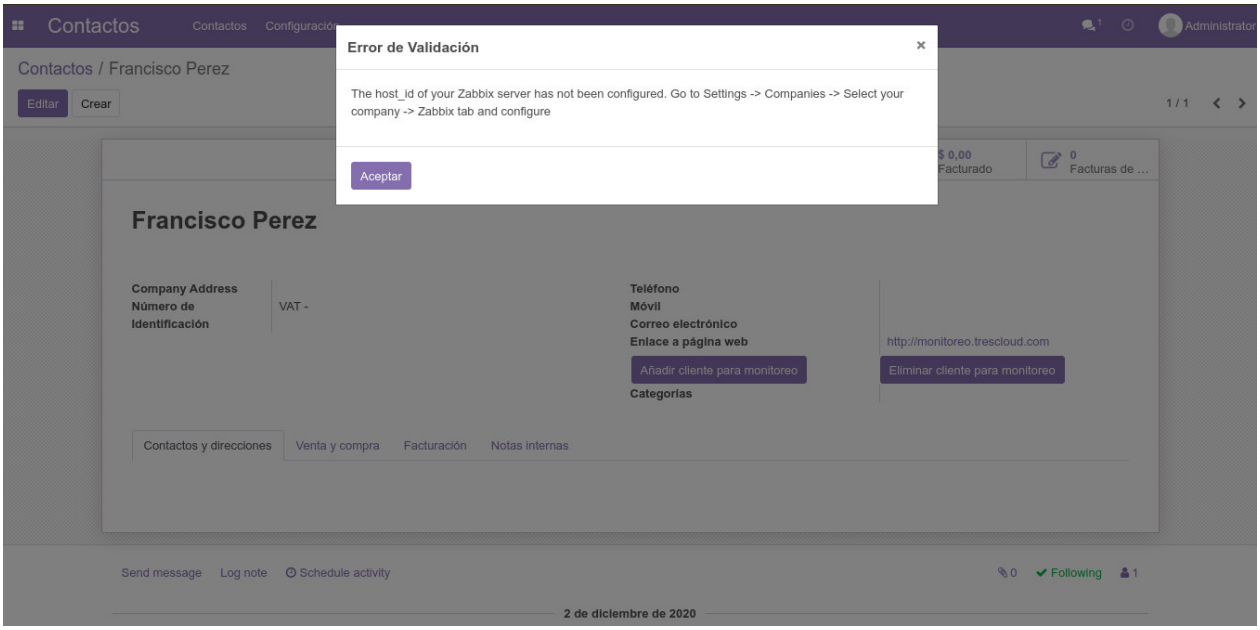


Figura 3.19 Error de validación cuando no se coloca el parámetro host_id

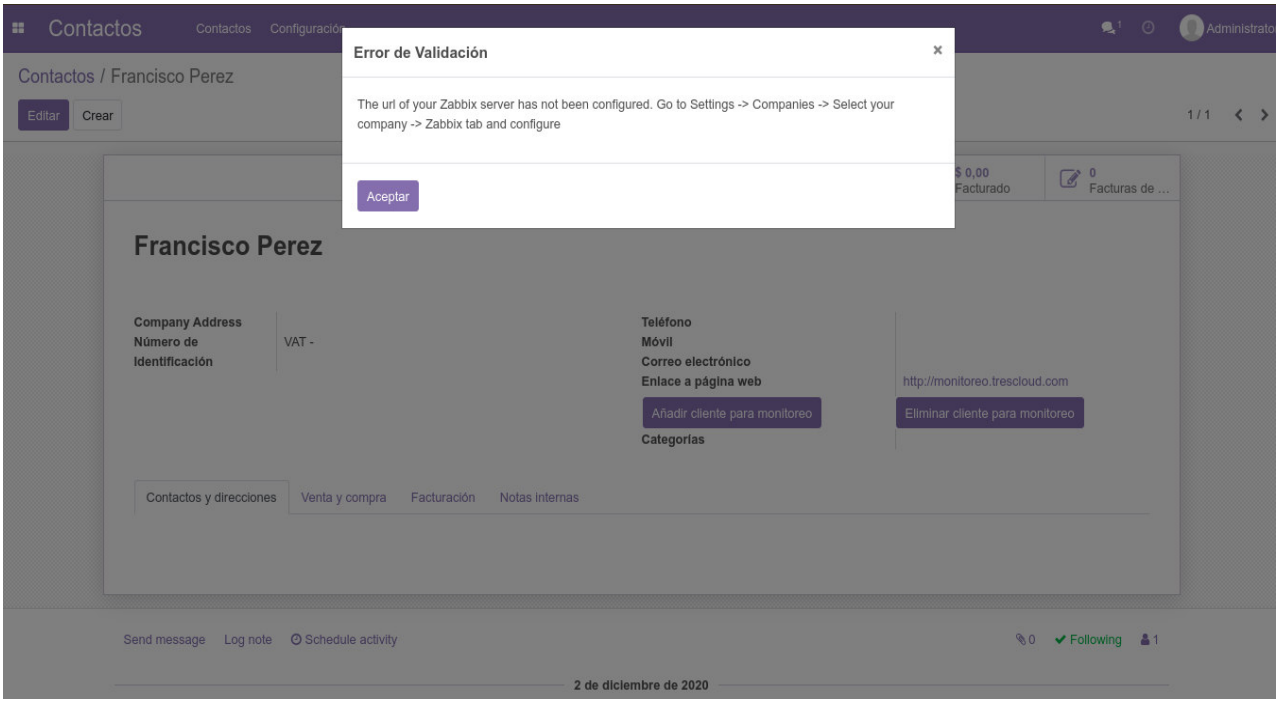


Figura 3.20 Error de validación cuando no se coloca el parámetro url

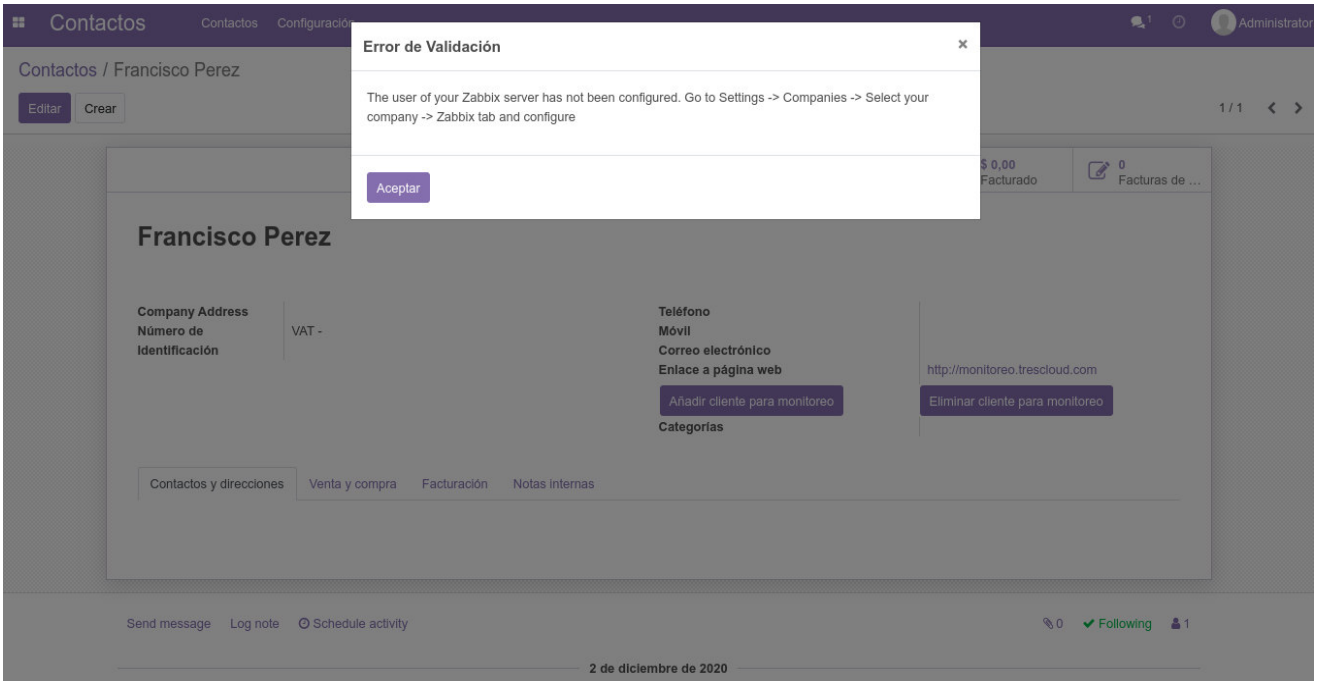


Figura 3.21 Error de validación cuando no se coloca el parámetro user

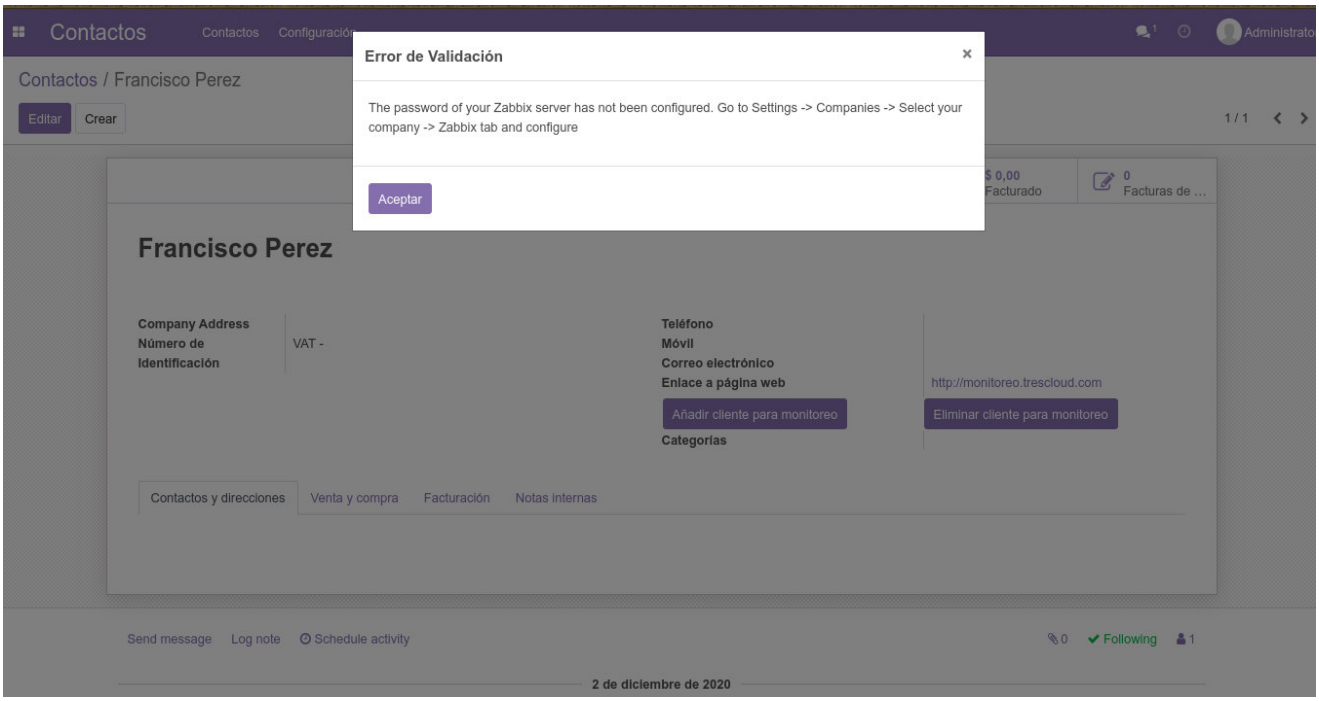


Figura 3.22 Error de validación cuando no se coloca el parámetro password

Cuando el servidor Zabbix se encuentra caído o no responde, el sistema notificará que existe un error al intentar conectarse (ver Figura 3.23).

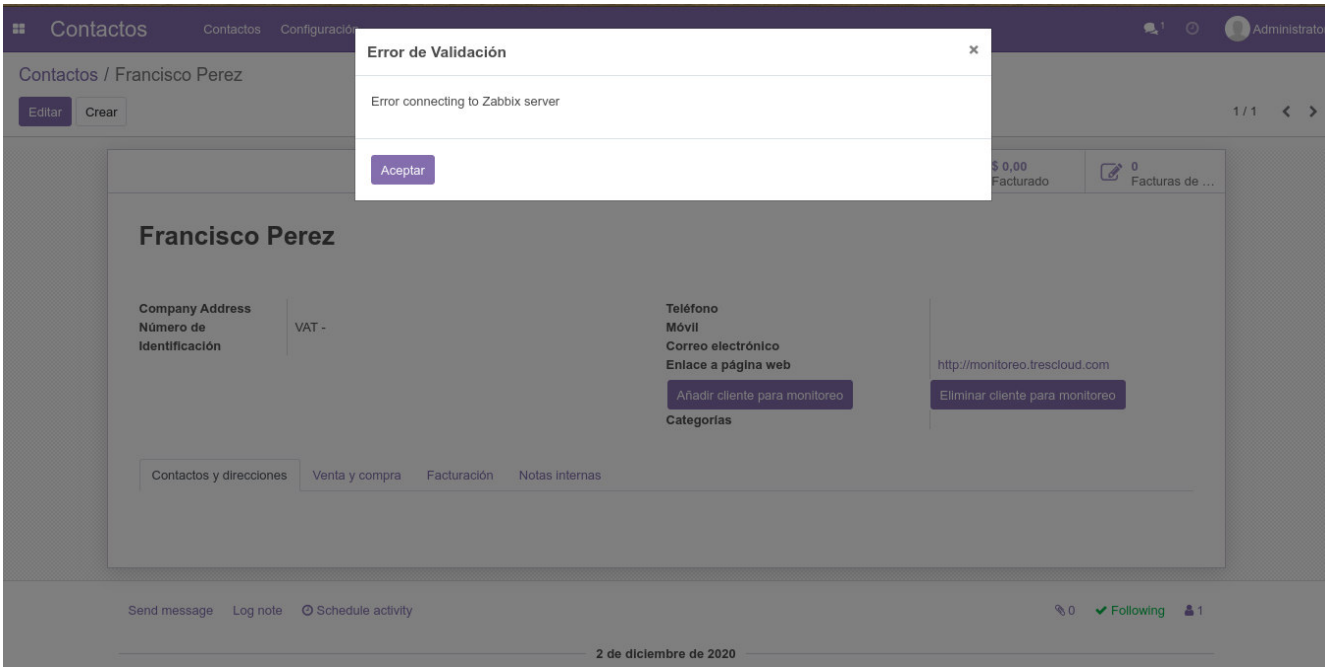


Figura 3.23 Error de validación cuando no se puede conectar al servidor Zabbix

3.1.2.2. Prueba de creación de un cliente Odoo al sistema de monitoreo Zabbix

Cuando existe conexión desde Odoo con el servidor de Zabbix, se puede realizar la prueba de la creación de un cliente Odoo al sistema de monitoreo Zabbix; en este caso se crea el cliente *Server Client B* para ser monitoreado en Zabbix (ver Figura 3.24).

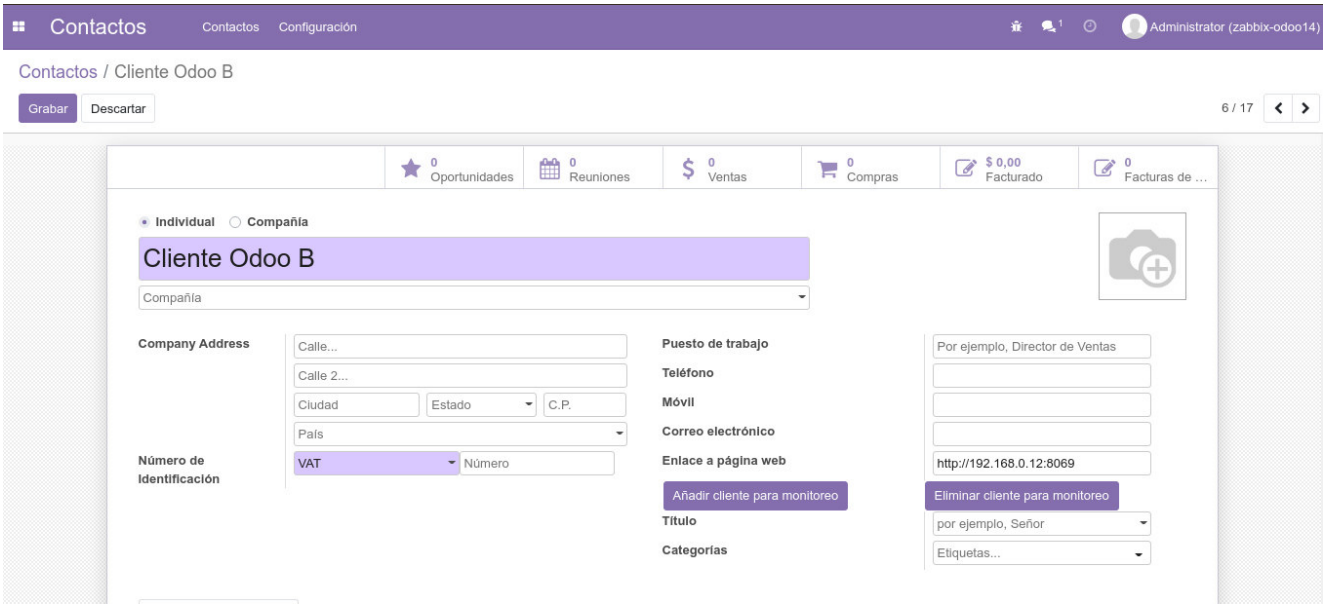


Figura 3.24 Creación del cliente Server Client B en Odoo

Una vez que se ha creado el cliente, se procede a guardarlo, luego se hace clic en el botón *Añadir cliente para monitoreo* para poder crear este cliente en el sistema de monitoreo Zabbix (ver Figura 3.25).

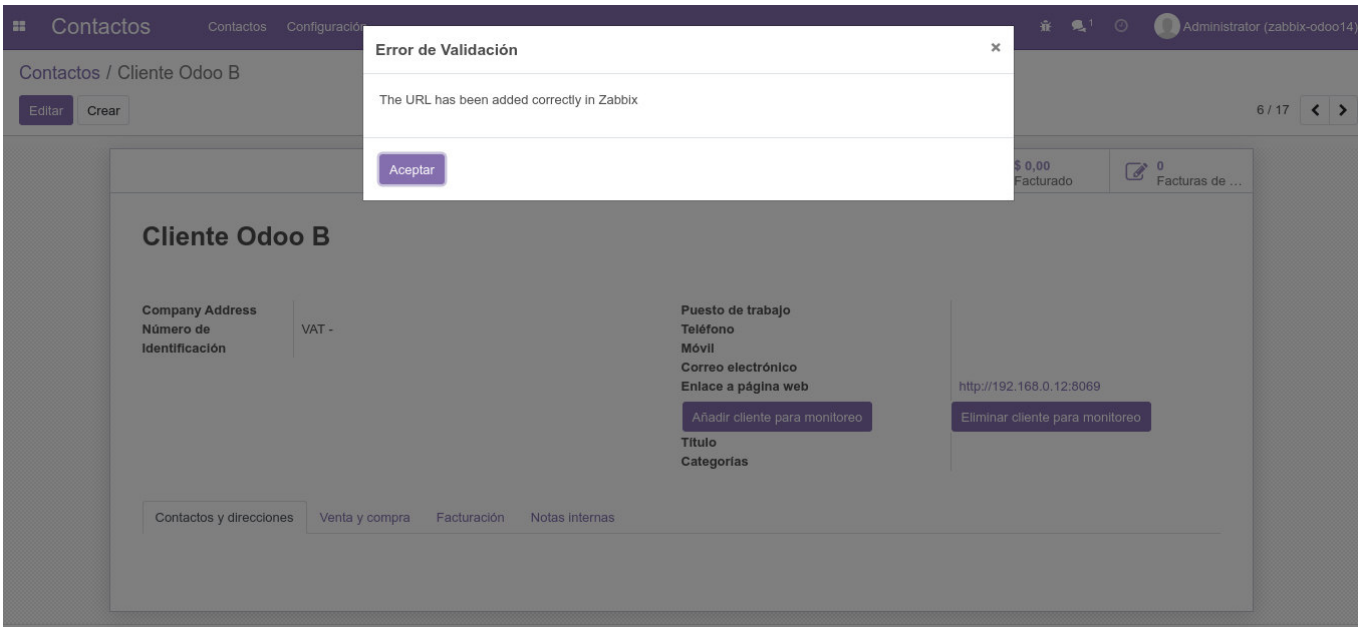


Figura 3.25 Creación del Server Client B en Odoo mediante el botón Añadir

En la Figura 3.26 se muestra cómo se crea el cliente *Server Client B* en Zabbix.

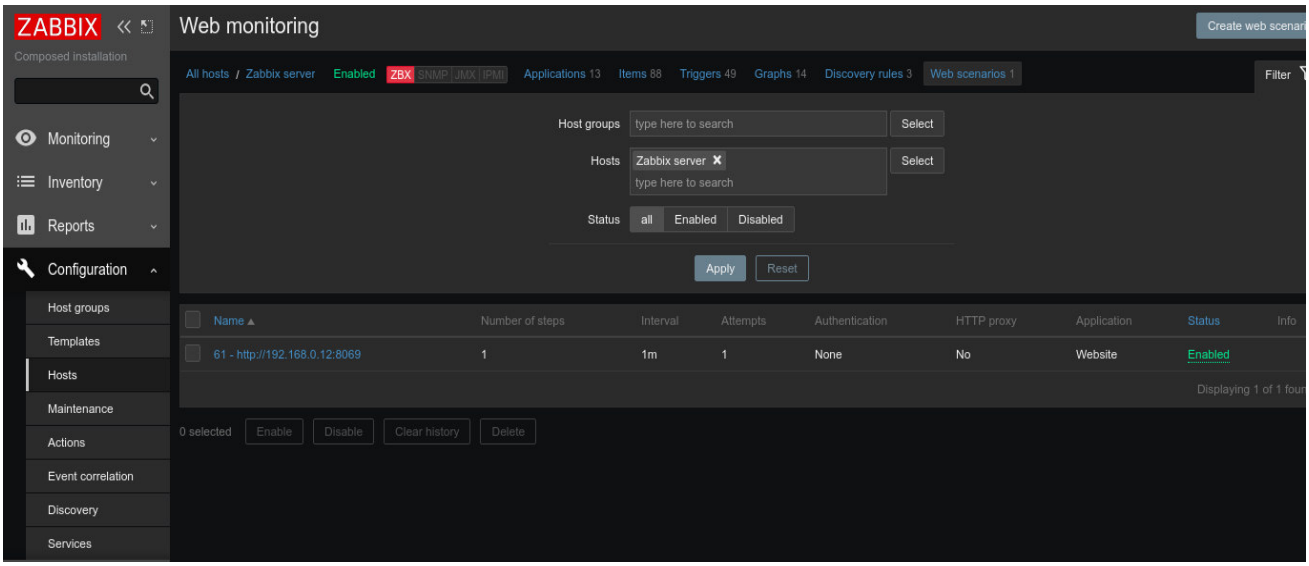


Figura 3.26 Creación del cliente Server Client B en Zabbix

En la Figura 3.27 se muestra que una vez creado el cliente *Server Client B* se crea el *trigger* que está vinculado a este mismo cliente.

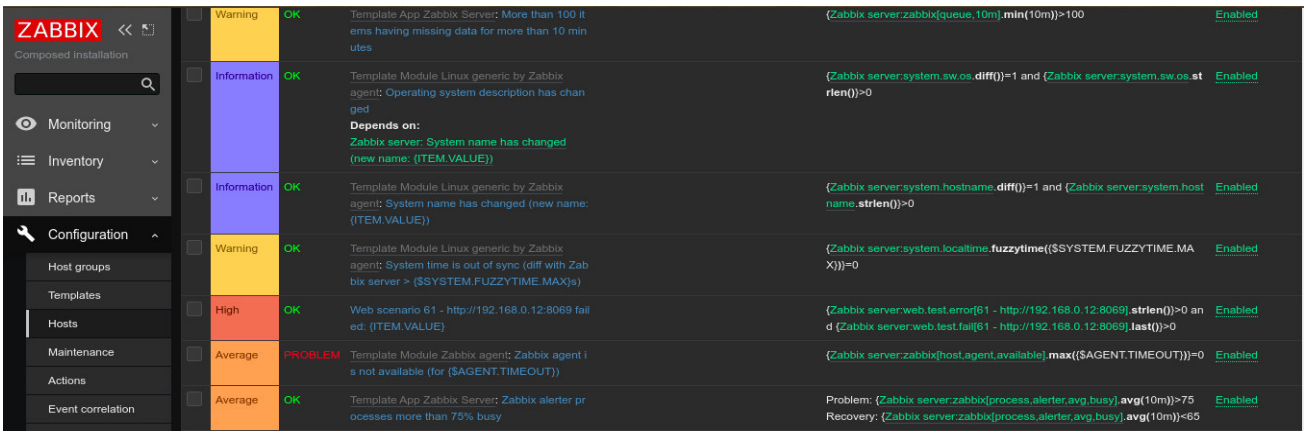


Figura 3.27 Creación del trigger del Server Client B en Zabbix

En la Figura 3.28 se indica la velocidad de descarga de los datos de la URL de Odoo junto con el tiempo de respuesta del mismo servidor en un rango de 5 minutos.

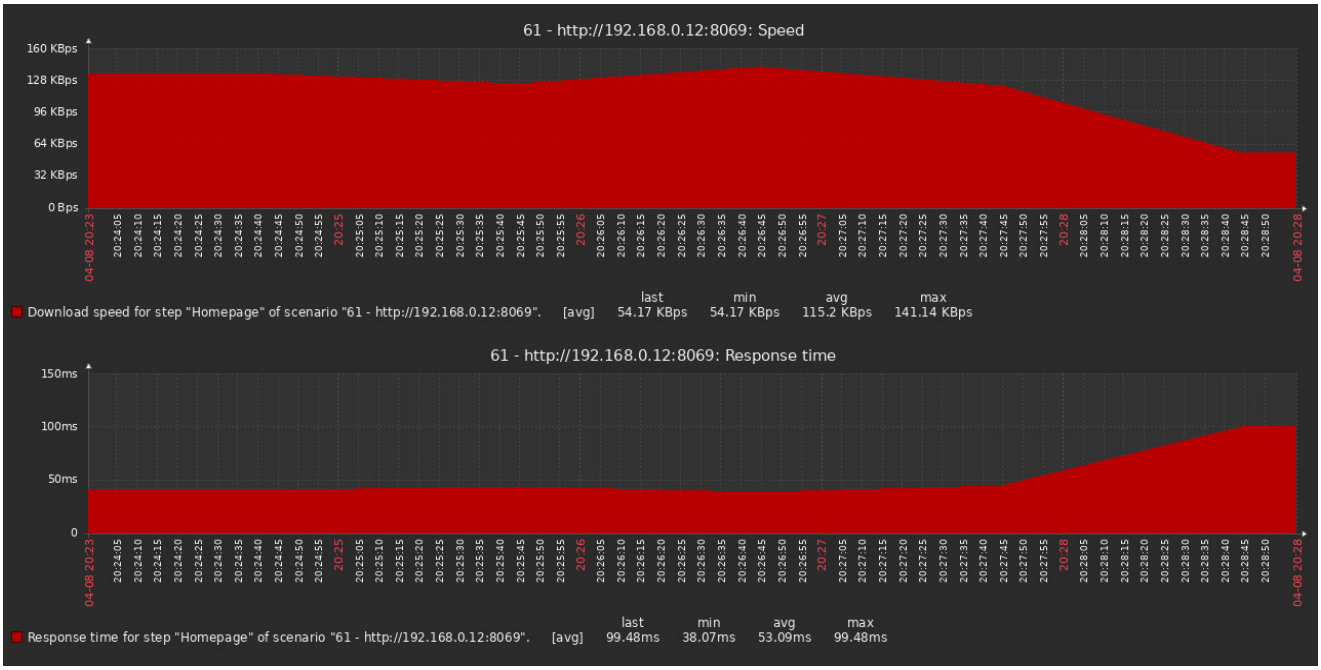


Figura 3.28 Gráficas del cliente Server Client B en Zabbix

Para evitar posibles duplicaciones de un cliente se crean excepciones que avisan al usuario que ya existe ese cliente (ver Figura 3.29). Para futuras mejoras se podría implementar la función de modificar un cliente de monitoreo, con esto el usuario no tendría que borrar el cliente y volverlo a crear.

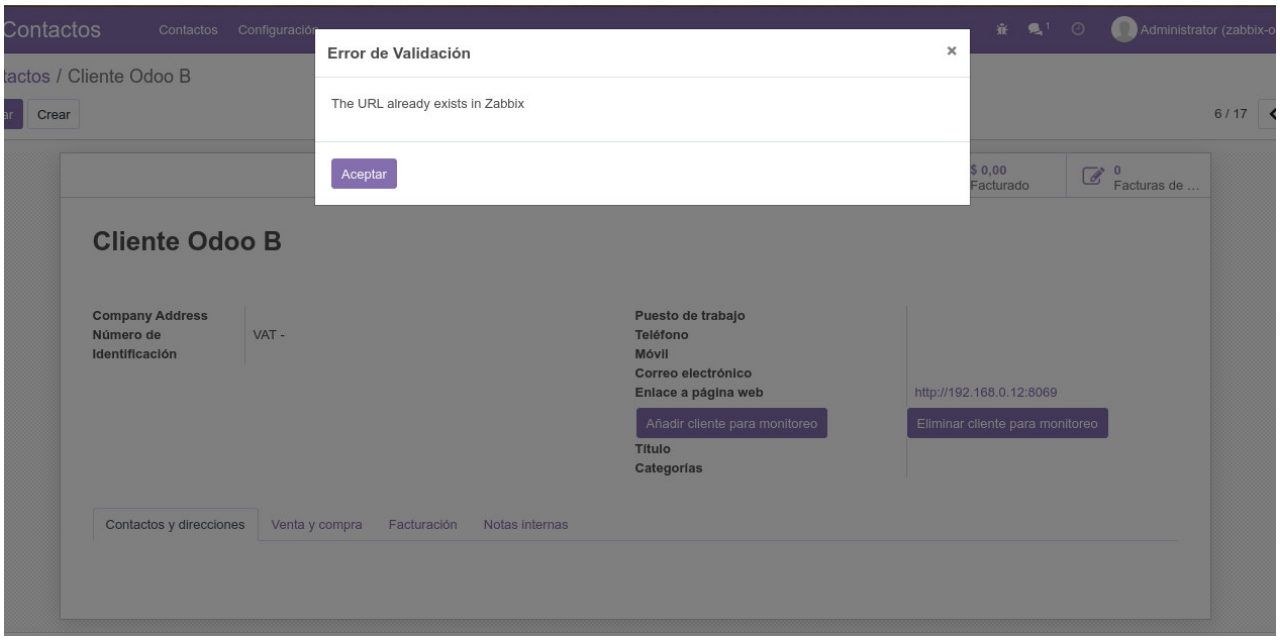


Figura 3.29 Error cuando se quiere crear un cliente que ya existe en Zabbix

En la Figura 3.30 se indica la excepción cuando la URL ingresada no coincide con la enviada en Zabbix.

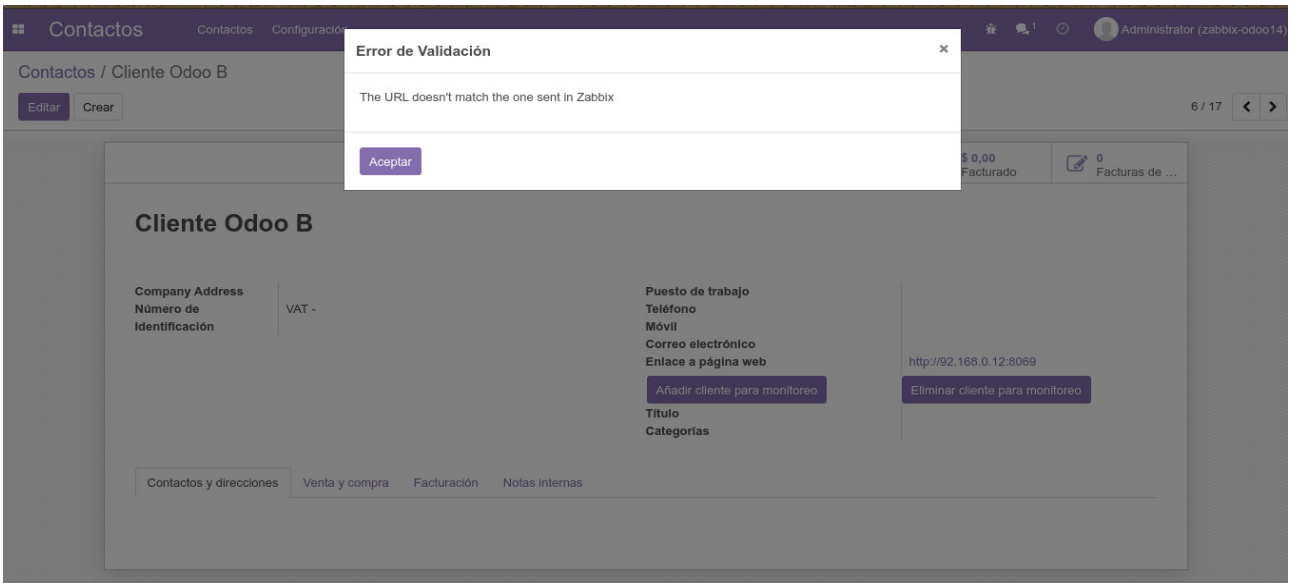


Figura 3.30 Error cuando la URL ingresada no coincide con la enviada en Zabbix

3.1.2.3. Prueba de eliminación de un cliente Odoo del sistema de monitoreo Zabbix

Para hacer esta prueba se emplea el mismo cliente *Server Client B,* con el propósito de eliminarlo del sistema de monitoreo en Zabbix; cabe mencionar que al eliminar este cliente se lo hará también de todo lo relacionado a éste incluyendo su configuración y sus *triggers*.

Para eliminar un cliente del sistema de monitoreo desde Odoo hay que ubicarse en el cliente que se desea eliminar y se da clic en *Eliminar cliente para monitoreo* (ver Figura 3.31).

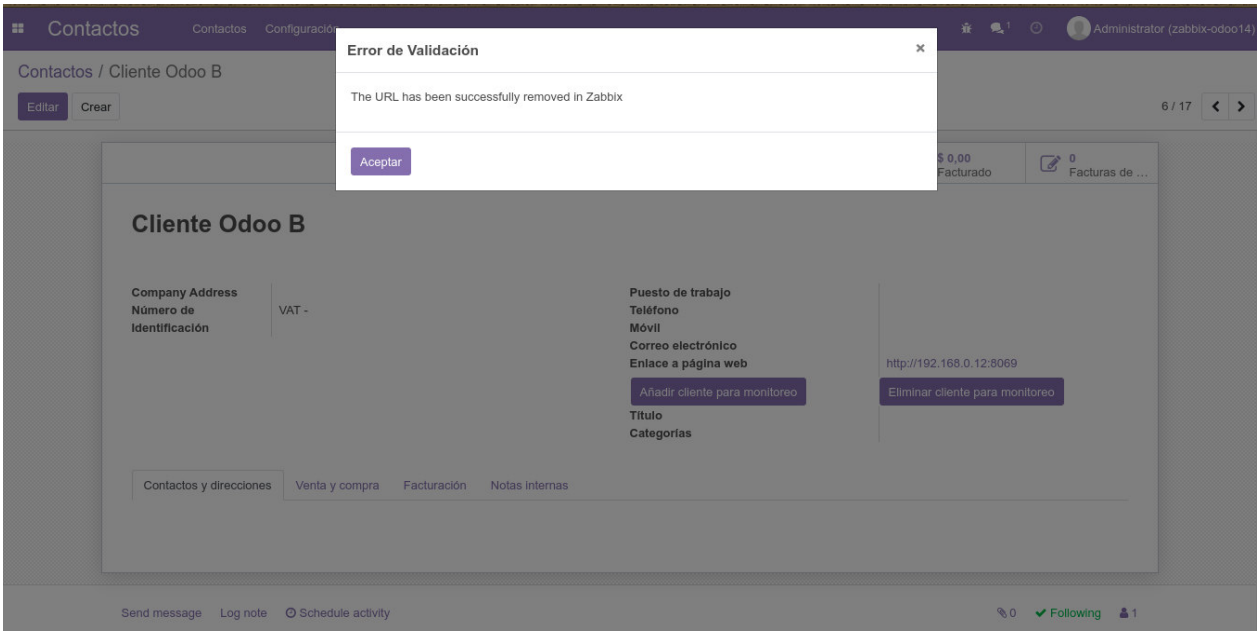


Figura 3.31 Eliminación del Server Client B en Odoo mediante el botón Eliminar

En la Figura 3.32 se muestra efectivamente que ha sido eliminado el cliente B del sistema de monitoreo Zabbix.

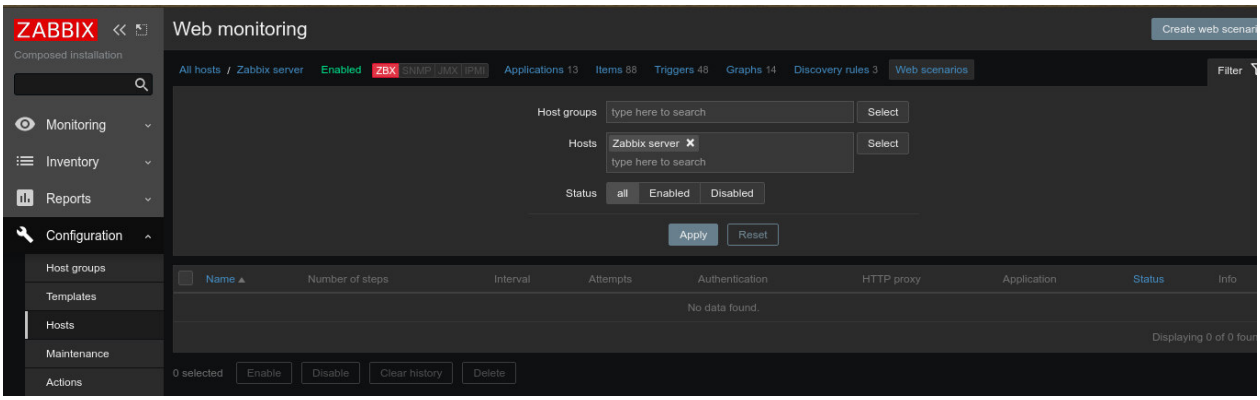


Figura 3.32 Eliminación del Server Client B en Zabbix

Al igual que cuando se crea un cliente para el monitoreo de Zabbix se tienen excepciones, también existen cuando se elimina un cliente del sistema Zabbix, esto con el fin evitar posibles problemas de confusión con el usuario y la plataforma. Cuando se quiere volver a eliminar el cliente previamente eliminado, el sistema indica que ese cliente con ese ID no está siendo monitoreado (ver Figura 3.33).

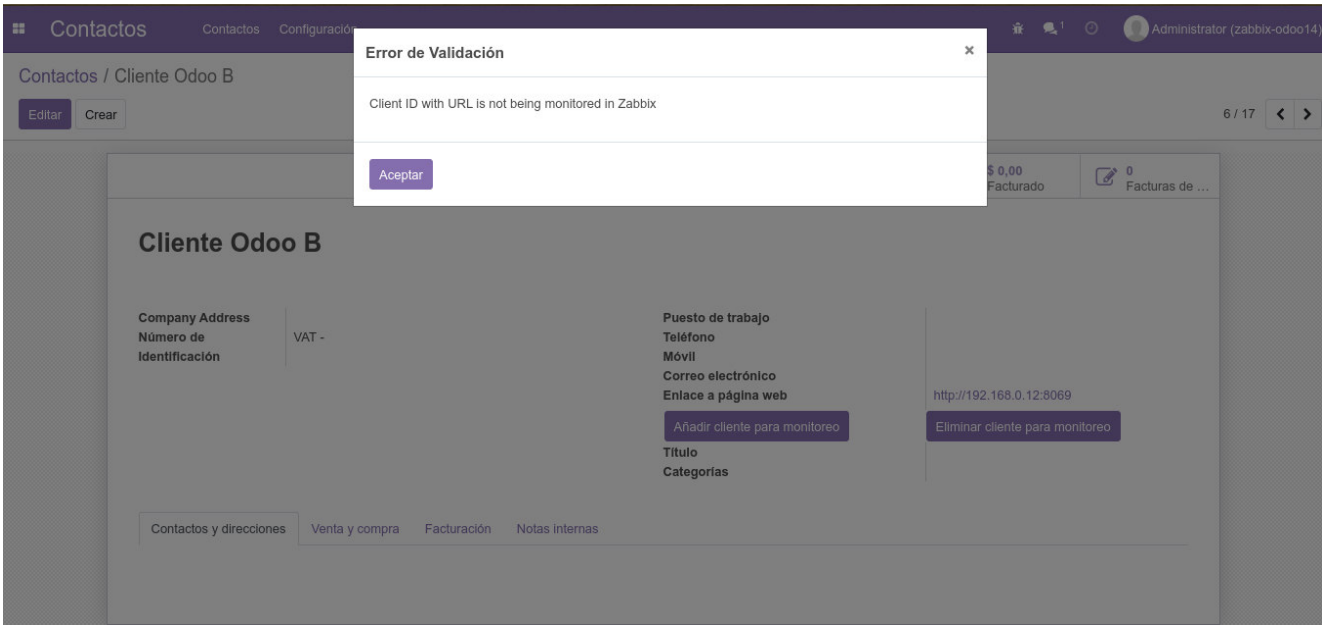


Figura 3.33 Error cuando un ID no está en Zabbix ya que fue eliminado antes

3.1.3. PRUEBAS DE FUNCIONAMIENTO DE LAS NOTIFICACIONES DE LA PLATAFORMA ZABBIX

Para las pruebas de funcionamiento de las notificaciones que llegan al usuario cuando se cae un cliente o un servidor Odoo, se emplea el servidor "*Server Client B"* y el cliente "*Cliente Odoo B"*.

Para esta prueba lo que se hizo fue "bajar" (apagar) el servidor, y para verificar la no disponibilidad del cliente lo que se hizo fue "bajar" el servicio de Odoo, con esto se comprueba que Zabbix realiza el envío de notificaciones tanto por vía *email* como por vía SMS de los problemas que se suscitan al hacer estas acciones en ambas máquinas.

En la Tabla 3.3 se indican todas las pruebas realizadas para el funcionamiento de las notificaciones de la plataforma Zabbix.

Tabla 3.3. Tipos de Pruebas del funcionamiento de las notificaciones en Zabbix

Prueba	Definición
Pruebas de Notificaciones	Indican todas las notificaciones que fueron llegando del cliente *Server Client B.*
Pruebas de Notificación *Email*	Indican todas las notificaciones que llegan por vía *email*.
Pruebas de Notificación SMS	Indican todas las notificaciones que llegan por vía SMS.

3.1.3.1. Prueba de funcionamiento de la notificación de fallo de un servidor o cliente Odoo por vía correo electrónico

Se debe tomar en cuenta que para que funcione este tipo de notificación se debe configurar todo lo relacionado al *media types* Email TRESCLOUD; para más información de su configuración referirse al Anexo D.

3.1.3.1.1. Prueba de funcionamiento de la notificación de fallo de un servidor Odoo por vía correo electrónico

En la Figura 3.34 se muestra la notificación de la caída del servidor *Server Client B.*

Figura 3.34 Notificación de Zabbix sobre la caída del servidor Odoo

En la Figura 3.35 se muestra la notificación de la caída del servidor *Server Client B* por *email.*

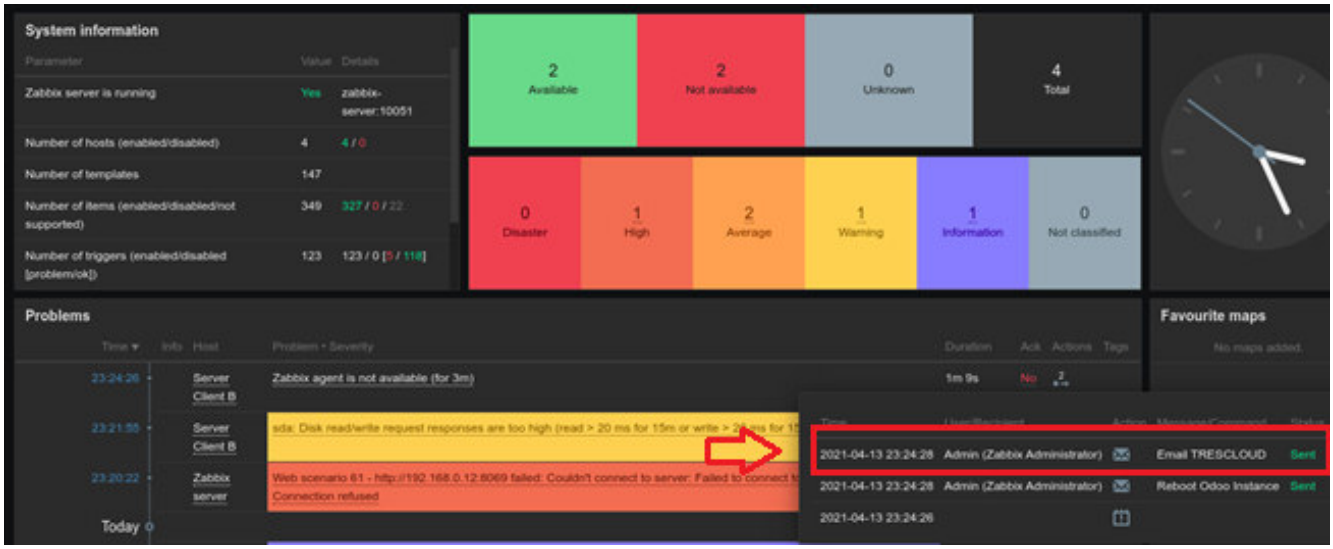


Figura 3.35 Notificación de Zabbix por email sobre la caída del servidor Odoo

En la Figura 3.36 se muestra la llegada de la notificación de la caída del servidor *Server Client B* por *email.*

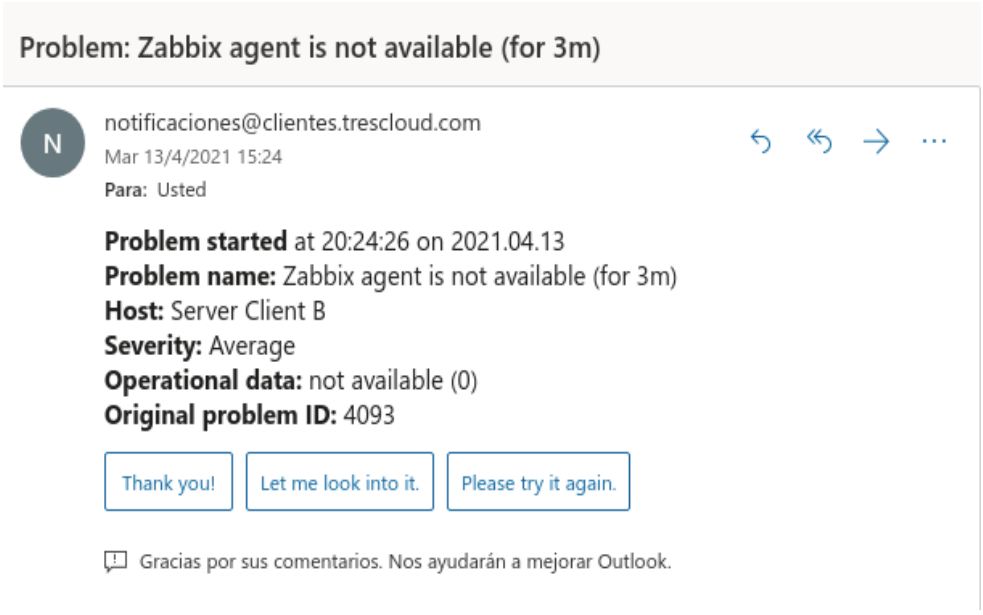


Figura 3.36 Notificación de fallo del servidor Odoo al email

En la Figura 3.37 se muestra la llegada de la notificación por *email* cuando se solucionó la caída del servidor *Server Client B.*

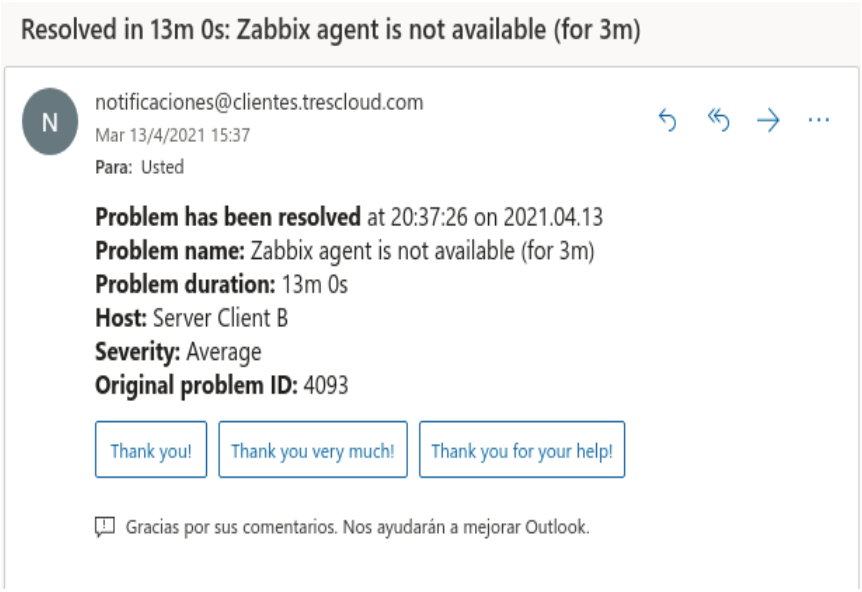


Figura 3.37 Notificación al email de solución de la caída del servidor Odoo

3.1.3.1.2. Prueba de funcionamiento de la notificación de fallo de un cliente Odoo por vía correo electrónico.

En la Figura 3.38 se muestra la notificación de la caída del cliente *Server Client B.*

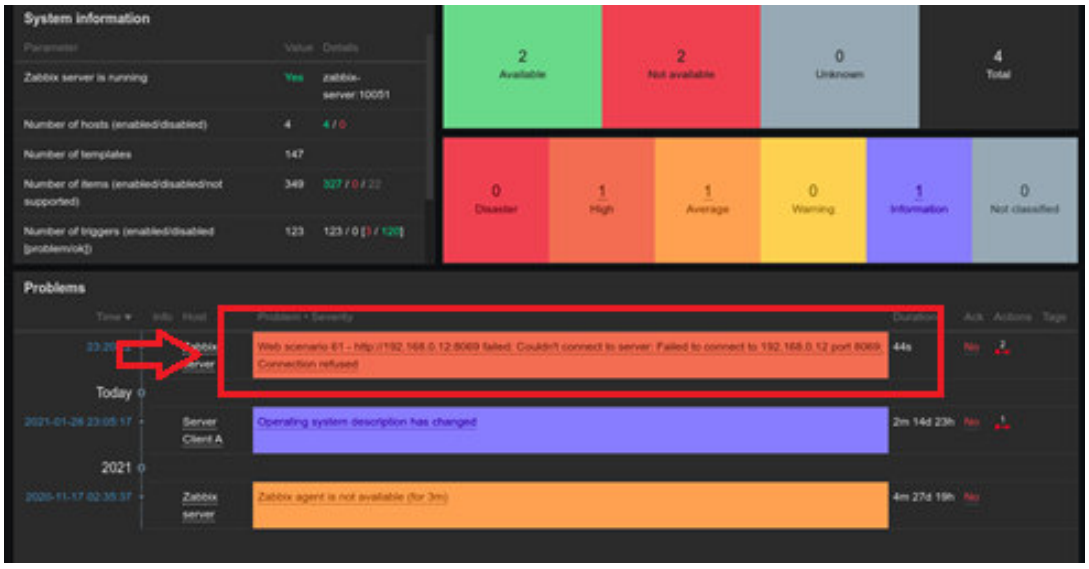


Figura 3.38 Notificación de Zabbix sobre la caída del cliente Odoo

En la Figura 3.39 se muestra la notificación de la caída del cliente *Server Client B* por vía *email* en Zabbix.

Figura 3.39 Notificación de Zabbix por vía email sobre la caída del cliente Odoo

En la Figura 3.40 se muestra la recepción de la notificación de la caída del cliente *Server Client B* por vía *email*.

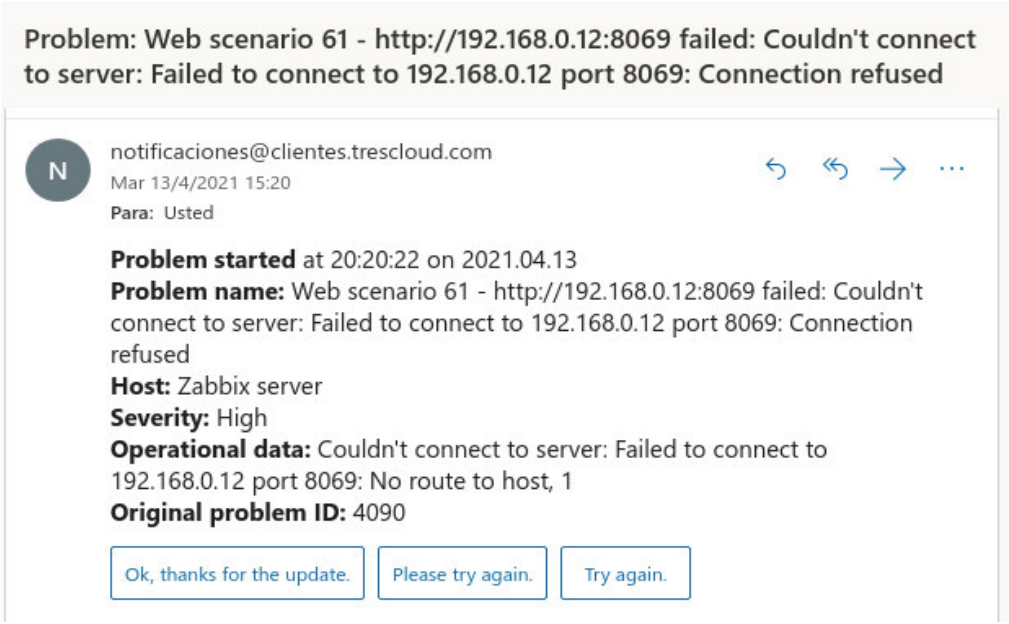


Figura 3.40 Notificación al email de fallo del cliente Odoo

En la Figura 3.41 se muestra la notificación por vía *email* cuando se solucionó la caída del cliente *Server Client B.*

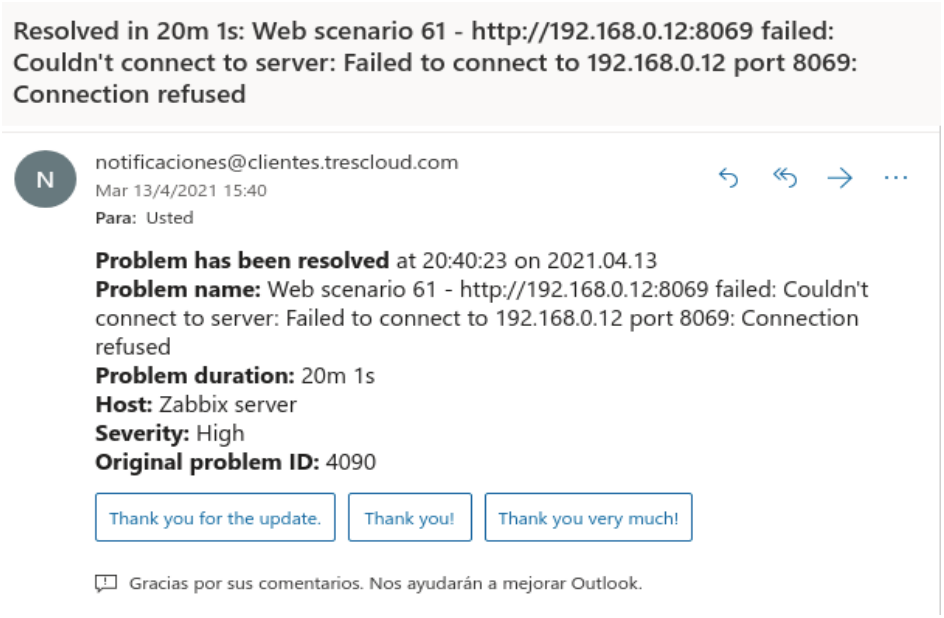


Figura 3.41 Notificación al email de solución de la caída del cliente Odoo

3.1.3.2. Prueba de funcionamiento de la notificación de fallo de un servidor o cliente Odoo por vía mensaje SMS

Se debe tomar en cuenta que para que funcione este tipo de notificación se debe configurar todo lo relacionado al *media types* SMSGlobal SMS. Para más información de su configuración referirse al anexo D.

3.1.3.2.1. Prueba de funcionamiento de la notificación de fallo de un servidor Odoo por vía mensaje SMS

En la Figura 3.42 se muestra la notificación de la caída del servidor Server Client B en Zabbix por vía SMS.

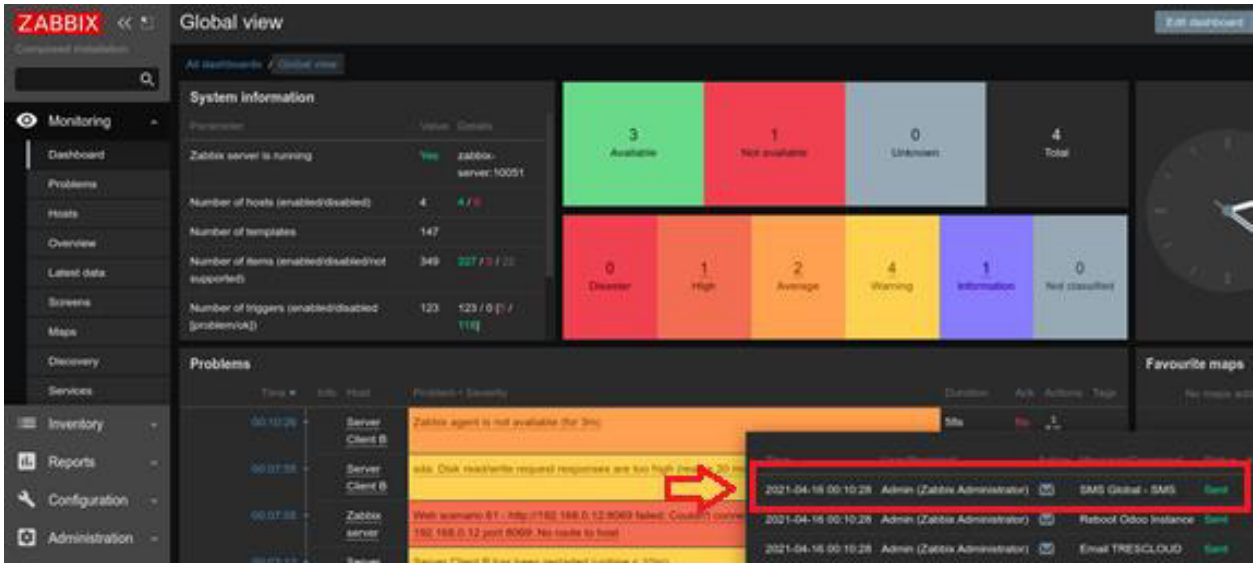


Figura 3.42 Notificación de Zabbix por vía SMS sobre la caída del servidor Odoo

En la Figura 3.43 se muestra la notificación de la caída del servidor *Server Client B* por vía SMS al teléfono móvil.

Figura 3.43 Notificación de la caída del servidor Odoo al teléfono móvil

3.1.3.2.2. Prueba de funcionamiento de la notificación de fallo de un cliente Odoo por vía mensaje SMS

En la Figura 3.44 se muestra la notificación por vía SMS de la caída del cliente Server Client B en Zabbix.

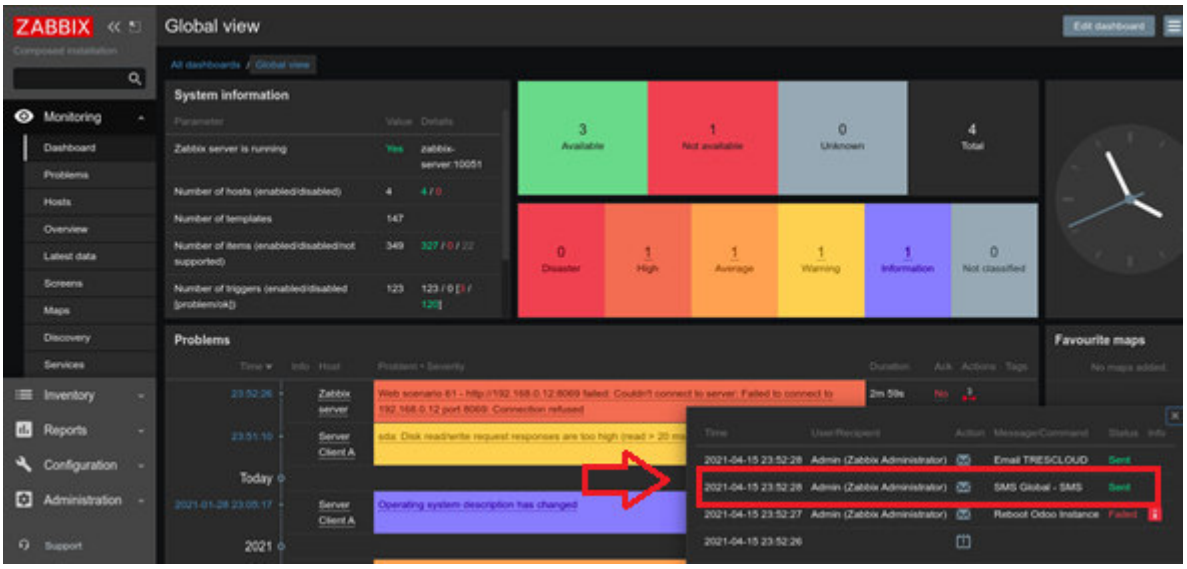


Figura 3.44 Notificación de Zabbix por vía SMS sobre la caída del cliente Odoo

En la Figura 3.45 se muestra la notificación de la caída del cliente *Server Client B* por vía SMS al teléfono móvil.

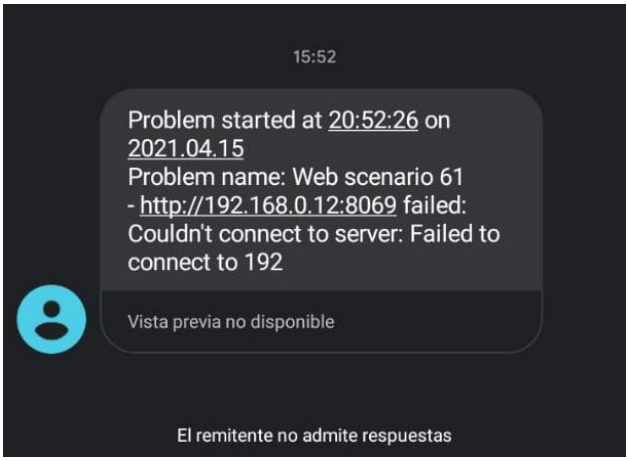


Figura 3.45 Notificación de la caída del cliente Odoo al teléfono móvil

3.1.4. PRUEBA DE FUNCIONAMIENTO DEL ENVÍO DE NOTIFICACIÓN A ODOO CUANDO UN CLIENTE ODOO SE CAYÓ O TUVO PROBLEMAS

Para realizar la prueba de que un cliente Odoo cayó o tuvo problemas y se notifique esta situación al sistema Odoo, se debe tomar en cuenta la configuración del *media types* Reboot Odoo Instance. Más información sobre la configuración de este *media types* se detalla en el anexo D.

Para esta prueba se toma como ejemplo al cliente "*Cliente Odoo C"* creado previamente en el sistema Odoo; para realizar la prueba se detiene el servicio de Odoo y a continuación se comprueba que Zabbix le notifique a Odoo que ese cliente no está disponible.

En la Tabla 3.4 se indican todas las pruebas utilizadas para el funcionamiento de las notificaciones cuando un cliente Odoo cayó o tuvo problemas.

Tabla 3.4. Tipos de Pruebas del funcionamiento de las notificaciones en Zabbix - Odoo

Prueba	Definición
Prueba de Notificación	Indica todas las notificaciones que llegaron del cliente *Server Client C* en Zabbix.
Prueba de Notificación Odoo	Indica la notificación en Odoo cuando el cliente *Server Client C* cayó o tuvo problemas.

En la Figura 3.46 se muestra la notificación de que el cliente *Server Client C*. tuvo algún problema.

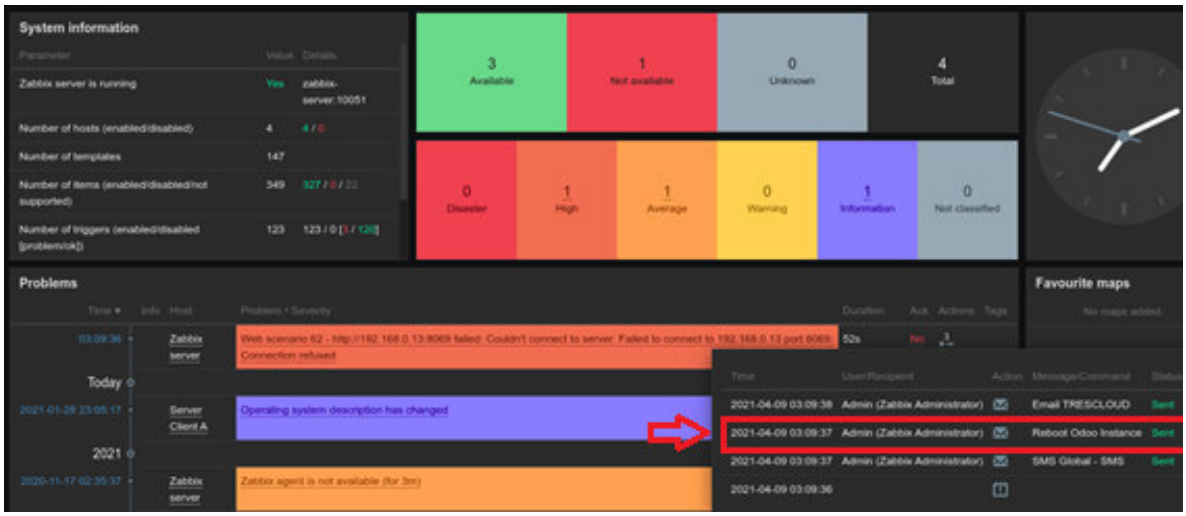


Figura 3.46 Notificación de que el cliente Odoo C tuvo problemas desde Zabbix

En la Figura 3.47 se muestra la notificación en Odoo de que el cliente *Server Client C* tuvo algún problema.

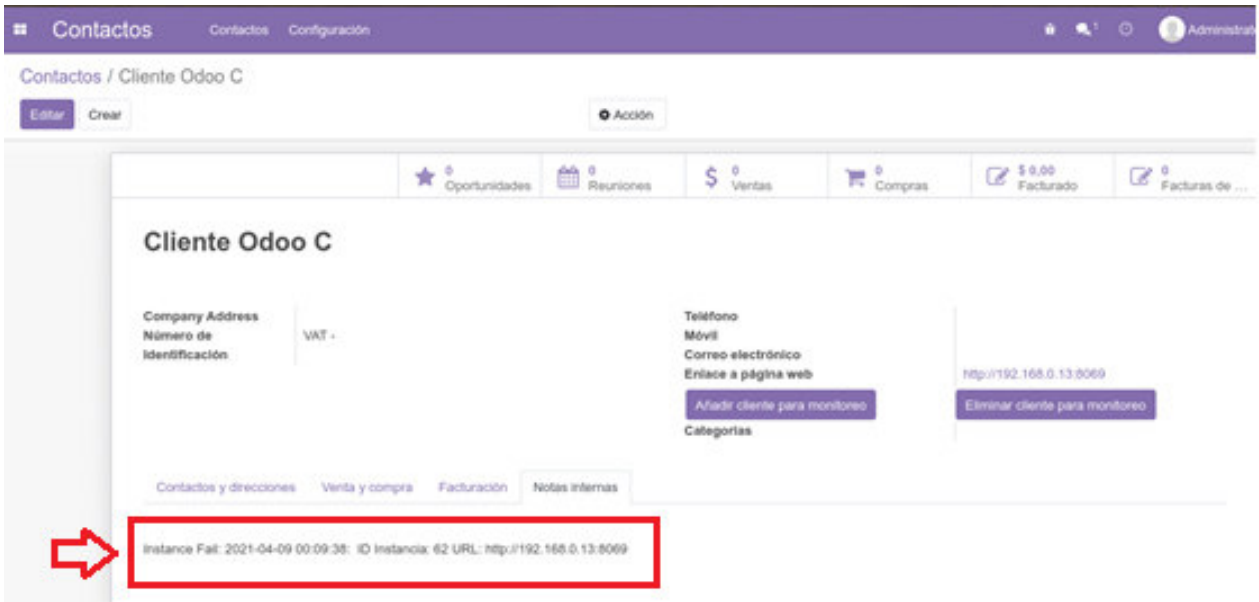


Figura 3.47 Notificación al sistema Odoo de que el Cliente Odoo C tuvo un fallo

Para concluir este capítulo se debe entender que todas las pruebas realizadas previamente fueron hechas bajo un ambiente controlado, ya que no se dispone de las facilidades de un ambiente real, lo cual coincide con el alcance del plan propuesto.

CAPÍTULO IV

CAPÍTULO IV

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Con el uso de la herramienta de monitoreo Zabbix y el sistema de gestión empresarial Odoo, se puede dar paso a la tendencia de utilizar ambos sistemas en conjunto, con el fin de que una empresa trabaje con un sistema de gestión empresarial completo como lo es Odoo y que con la ayuda de Zabbix monitoree toda su infraestructura tanto de servidores como de clientes, gracias a la creación del módulo de interconexión Odoo - Zabbix.

- El sistema de monitoreo implementado junto con su módulo de interconexión creados en este proyecto de titulación, no se los pudo poner a prueba en producción debido a la privacidad que tiene la empresa TRESCLOUD CIA LTDA. con sus módulos de Odoo, por lo que este proyecto lo podrán emplear las empresas como una guía de utilización del módulo de interconexión Odoo - Zabbix, pudiendo modificarlo y utilizarlo según sus requerimientos.

- Se realizó un análisis comparativo con la herramienta de monitoreo Nagios que es la más conocida a nivel empresarial, donde se estudió y se evaluaron las principales características técnicas y el costo de las distintas opciones que ofrece el mercado, con el fin de tener la mejor relación entre costo y beneficio, siendo Zabbix el software que mejor llegó a cumplir esa relación.

- Trabajar con la tecnología Docker para la instalación del software Zabbix, fue más eficiente que seguir comando por comando como lo recomienda la página de Zabbix en el caso de repetir la instalación, ya que al hacerlo con Docker se trabajó con menos comandos que el manual de Zabbix. Con Docker Compose se pudo ejecutar en un solo archivo la instalación de Zabbix tanto en el servidor de prueba como en su implementación final.

- El lenguaje Python es muy versátil y más simplificado que otros lenguajes de programación, además, permite crear incontables aplicaciones que junto con su vasta cantidad de librerías, permitió crear el módulo de interconexión Odoo – Zabbix; un ejemplo del uso de Python en este proyecto fue el del *if, try* y *except* para las validaciones de conexión al servidor Zabbix desde Odoo.

- En este proyecto de titulación se trabajó con Web Services tanto en Odoo como en Zabbix exponiendo sus respectivas API's, esto permite la interoperabilidad entre muchas aplicaciones y una comunicación de sus datos de manera más sencilla.

- Para el envío de notificaciones por vía SMS es necesario contar con un proveedor de mensajería para enviar las notificaciones de Zabbix por vía SMS, caso contrario el envío de notificaciones solo se las haría por vía *email*.

- Para poder trabajar con la aplicación SMS Global se tuvo que contactar con el soporte de la aplicación debido a que no funcionaba correctamente el envío de los mensajes SMS al teléfono móvil, esto se solventó pidiendo a la operadora móvil que cambie el enrutamiento de los mensajes que llegan al teléfono.

- Gracias al uso de SMS Global se puede comunicar sobre eventuales fallos, de una manera rápida y ágil, permitiendo que el usuario pueda tomar acciones adecuadas cuando un servidor o cliente Odoo sufre algún problema grave.

- Mediante el uso del software de monitoreo Zabbix y el módulo programado en Odoo se pudo implementar todo lo necesario para que la empresa TRESCLOUD tenga a su disposición la herramienta de monitoreo completa, sencilla y eficaz requerida para una empresa con un creciente número de clientes.

4.2. RECOMENDACIONES

- Al realizar la instalación recomendada de Zabbix, se debe tomar en cuenta que para las pruebas se debe colocar la dirección IP local del servidor al momento de editar el archivo de configuración del sitio web de Nginx, esto porque no se cuenta con un dominio como en el ejemplo de instalación de la herramienta Zabbix.

- La administración de redes es un componente importante en el desempeño de una empresa, por tanto es recomendable realizar una actualización de las herramientas que presten beneficios tecnológicos para mantener a la empresa actualizada en su infraestructura.

- Para la implementación de esta solución tecnológica se recomienda utilizar la última versión de Zabbix 5.2 para realizar el monitoreo de los clientes o servidores Odoo así como la versión 14 de Odoo, debido a que si quiere trabajar con otra versión anterior o posterior podrían darse problemas de librerías y el módulo de interconexión o cualquiera de los *scripts* no funcionarán correctamente.

- Se recomienda al administrador de la red de la empresa buscar e instalar las actualizaciones de seguridad y corrección de errores tanto de Zabbix como de Odoo y de las herramientas utilizadas en este proyecto, siempre cuidando que sea dentro de las versiones anteriormente mencionadas.

- Al momento de crear un *script* para el servidor Zabbix se recomienda codificarlo en lenguaje Python ya que es más amigable que codificar en lenguaje Bash, pero Zabbix solo permite ejecutar *scripts* en Bash así que se debe crear un *script* del tipo Bash que llame al *script* hecho en Python.

- Se recomienda añadir una función en la librería de interconexión Odoo - Zabbix que modifique los clientes creados en el sistema de monitoreo Zabbix, debido a que brindaría mayor eficiencia al momento de hacer la gestión de clientes al monitoreo en Zabbix en caso de posibles modificaciones en su URL a ser monitoreada.

- En el módulo de interconexión Odoo - Zabbix se podría implementar la funcionalidad de poder monitorear servicios como PostgreSQL, que es el motor de base de datos de los servidores Odoo.

- Se recomienda tomar en cuenta a las empresas que deseen implementar esta solución tecnológica sobre la cantidad de servidores que se van a manejar, debido a que esto genera más o menos tiempo para su implementación.

- Se recomienda que si no se logran envían mensajes SMS mediante la aplicación SMS Global al teléfono móvil, primero se debe verificar que el número de teléfono esté con recepción y esté activo; si el problema persiste como segundo paso debe contactarse con el soporte de SMS Global indicando el problema para que se pueda obtener ayuda.

- El desarrollo de la plataforma de monitoreo y su módulo de interconexión se diseñó para funcionar en la empresa TRESCLOUD CIA LTDA., sin embargo, se recomienda extenderlo a toda empresa que trabaje con el sistema de gestión empresarial Odoo y requiere un sistema de monitoreo competitivo y escalable interconectado con Odoo.

- El desarrollo de la plataforma de monitoreo y su módulo de interconexión se diseñó exclusivamente para monitorear clientes y servidores Odoo. Se recomienda ampliar el alcance de la plataforma como crear un botón que permita modificar la URL del cliente Odoo sin necesidad de eliminarlo y volverlo a crear en Zabbix., esto con el fin de darle un mayor beneficio al monitoreo con respecto a la infraestructura de una empresa.

BIBLIOGRAFÍA

5. BIBLIOGRAFÍA

- [1] D.R. Naranjo Villacrés & P.E.Ortega Tobar, Desarrollo de una aplicación gráfica basada en el sistema operativo Linux para el monitoreo y administración del tráfico de datos de redes lan, Quito: Escuela Politécnica Nacional, 2006. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/2353. [Consultado: 10-feb-2021].

- [2] C.A. Velasco Briones & G.S. Cagua Ordoñez, Implementación de un sistema de monitoreo de redes utilizando herramientas Open Source y proveer servicios de directorio a través de Active Directory en la Facultad de Filosofía, Letras y Ciencias de la Educación de la Universidad De Guayaquil, Guayaquil: Universidad de Guayaquil, 2017. [En línea]. Disponible en: http://dspace.ups.edu.ec/handle/123456789/13474. [Consultado: 10-feb-2021].

- [3] M.A. Ulloa Solis, Implementación de plataforma de monitoreo Zabbix para sistemas de Telecomunicaciones TELSUR, Valdivia: Universidad Austral de Chile, 2014. [En línea]. Disponible en: http://cybertesis.uach.cl/tesis/uach/2014/bmfciu.41i/doc/bmfciu.41i.pdf. [Consultado: 10-feb-2021].

- [4] J.I. Bayas Villagómez, Servidor de control de dispositivos y servicios mediante el protocolo SNMP para la red de datos en CELEC .E.P. Unidad De Negocio HIDROAGOYÁN, Ambato: Universidad Técnica De Ambato, 2015. [En línea]. Disponible en: https://repositorio.uta.edu.ec/jspui/handle/123456789/13063. [Consultado: 11-feb-2021].

- [5] C.A. Vicente Altamirano, Monitoreo de recursos de red, México: Universidad Nacional Autónoma de México, 2005. [En línea]. Disponible en: https://juliorestrepo.files.wordpress.com/2011/04/monitoreo.pdf. [Consultado: 11-feb-2021].

- [6] M.A. Santana, Análisis de Nagios Core como herramienta para el monitoreo de redes de datos, Toluca: Universidad Autónoma del Estado de México, 2017. [En línea]. Disponible en: http://hdl.handle.net/20.500.11799/68999 . [Consultado: 11-feb-2021].

- [7] T. G. Díaz, Software libre, software de código abierto, licencias, Créteil: Université Paris-Est Créteil, 2015. [En línea]. Disponible en: http://igm.univ-mlv.fr/~teresa/logicielsLIGM/documents/Internacional/2015septTGD_FS_OSS_Lic_Proc_esp.pdf . [Consultado: 11-feb-2021].

- [8] GNU.org, ¿Qué es el software libre?, 2021. [En línea]. Disponible en: https://www.gnu.org/philosophy/free-sw.es.html. [Consultado: 11-feb-2021].

- [9] M. C. Juárez, W.G.G. Herrera & S.T.Sanchez, Software libre vs software propietario Ventajas y desventajas, Ciudad de México: Universidad La Salle, 2006 [En línea]. Disponible en: http://test.esupcom.unr.edu.ar/bv_tics/biblioteca/info_complementaria/informatica_y_sociedad/derechos_de_autor/archivos/apuntes/1-libre_vs_propietario.pdf. [Consultado:11-feb-2021].

- [10] D.G. Aragón, Desarrollo de una plataforma de virtualización, Barcelona: Universitat Politécnica de Catalunya, 2008. [En línea]. Disponible en: https://upcommons.upc.edu/bitstream/handle/2099.1/4798/memoria.pdf.pdf?sequence=1&isAllowed=y. [Consultado: 23-feb-2021].

- [11] L.F. Ulloa, La virtualización y su impacto en las ciencias computacionales, Cali: Universidad del Valle, 2009. [En línea]. Disponible en: https://dialnet.unirioja.es/descarga/articulo/3402334.pdf . [Consultado: 23-feb-2021].

- [12] N. A. Balambá, Propuesta para el mejoramiento de la estrategia en aprovisionamiento de servidores y máquinas virtuales utilizando la herramienta de virtualización BMC, Bogotá D.C.: Universidad Católica de Colombia, 2013. [En línea]. Disponible en: http://hdl.handle.net/10983/982. [Consultado: 23-feb-2021].

- [13] D.M., M.M., J.U., E.B. & J.A. Moreiro, Virtualización, una solución para la eficiencia, seguridad y administración de Intranets, Madrid: Universidad Carlos III de Madrid, 2011. [En línea]. Disponible en: https://e-archivo.uc3m.es/bitstream/handle/10016/21643/virtualizacion_EPI_2011.pdf?sequence=2&isAllowed=y. [Consultado:23-feb-2021].

- [14] Oracle VirtualBox, <<VirtualBox,>>, 2021. [En línea]. Disponible en: https://www.virtualbox.org/ . [Consultado: 23-feb-2021].

- [15] Visual Studio Code, <<Visual Studio Code,>> 2021. [En línea]. Disponible en: https://code.visualstudio.com/docs . [Consultado: 25-feb-2021].

- [16] Microsoft Documentation, <<Microsoft Documentation,>> 25-05-2018. [En línea]. Disponible en: https://docs.microsoft.com/en-us/visualstudio/ide/using-intellisense?view=vs-2019&viewFallbackFrom=vs-2019. [Consultado: 25-feb-2021]

- [17] Visual Studio Code <<Visual Studio Code,>>, 2021. [En línea]. Disponible en: https://code.visualstudio.com/docs/getstarted/userinterface. [Consultado: 25-feb-2021]

- [18] GNU Operating System, <<GNU Bash,>>, 2020. [En línea]. Disponible en: https://www.gnu.org/software/bash/manual/bash.pdf. [Consultado: 01-mar-2021]

- [19] R.M. Gómez, Programación Avanzada en Shell , Sevilla: Universidad de Sevilla, 2005. [En línea]. Disponible en: http://www.informatica.us.es/~ramon/articulos/Programacion-BASH.pdf. [Consultado: 01-mar-2021].

- [20] R.G. Duque, "Python para todos", 2014. [En línea]. Disponible en: http://bvc.cea-atitlan.org.gt/397/. [Consultado: 03-mar-2021].

- [21] W. Chun, "Core python programming", Vol 1, 2001. [En línea]. Disponible en: https://books.google.es/books?hl=es&lr=&id=mh0bU6NXrBgC&oi=fnd&pg=PR1&dq=python+programming&ots=XBgCGw7-d9&sig=4GAkgDWZEWtq_UgaKaEEeq4B-Uk#v=onepage&q=python%20programming&f=false. [Consultado: 03-mar-2021].

- [22] K.R.Srinath, "Python-The Fastest Growing Programming Language", Telangana: International Research Journal of Engineering and Technology (IRJET), 2017. [En línea]. Disponible en: https://www.academia.edu/35592490/Python_The_Fastest_Growing_Programming_Language. [Consultado: 04-mar-2021].

- [23] A. Marzal & I. Gracia, “Introducción a la programación con Python”, Castellón de la Plana: Universidad Jaime I, 2003. [En línea]. Disponible en: https://ns2.elhacker.net/timofonica/manuales/Introduccion_%20Programacion_Python.pdf. [Consultado: 04-mar-2021].

- [24] Recursos Python, “30 métodos de las cadenas”, 2018. [En línea]. Disponible en: https://recursospython.com/guias-y-manuales/30-metodos-de-las-cadenas/. [Consultado: 05-mar-2021].

- [25] Uniwebsidad, “Algoritmos de Programación con Python”, 2014. [En línea]. Disponible en: https://uniwebsidad.com/libros/algoritmos-python/capitulo-7/tuplas. [Consultado: 05-mar-2021].

- [26] C.E. Plasencia, <<DevCode,>>, “Diccionarios en Python”, 2019. [En línea]. Disponible en: https://devcode.la/tutoriales/diccionarios-en-python/. [Consultado: 05-mar-2021].

- [27] Covantec, “Programación en Python- Nivel Básico”, Funciones, 2019. [En línea]. Disponible en: https://entrenamiento-python-basico.readthedocs.io/es/latest/leccion5/funciones.html. [Consultado: 08-mar-2021].

- [28] Python.org, <<Python Documentation,>>, Errores, 2021. [En línea]. Disponible en: https://docs.python.org/es/3/tutorial/errors.html. [Consultado: 08-mar-2021].

- [29] A.S. Velásquez Cruz, Diseño e implementación de un módulo software para la monitorización de elementos de una red informática utilizando el protocolo SNMP y el lenguaje XML, Quito: Escuela Politécnica Nacional, 2009. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/1135.[Consultado: 09-mar-2021].

- [30] T. Boulanger, “XML práctico: Bases Esenciales, conceptos y casos prácticos”, 2ª Edición, 2015. [En línea]. Disponible en: https://books.google.es/books?hl=es&lr=&id=x6QToh9wHMYC&oi=fnd&pg=PA21&dq=lenguaje+xml&ots=3PA1aVrJfT&sig=fZODXCpi65RcFvCA9kwEgPtVnAw. [Consultado: 09-mar-2021].

- [31] O. Pascual Núñez, Desarrollo de un lenguaje XML para la participación electrónica, Madrid: Universidad Rey Juan Carlos, 2010. [En línea]. Disponible en: https://burjcdigital.urjc.es/handle/10115/4385. [Consultado: 09-mar-2021].

- [32] J. Conti, "Recomendaciones al programar en Python", 2008. [En línea]. Disponible en: http://www.juanjoconti.com/posts/2008/10/28/recomendaciones-al-programar-en-python/. [Consultado: 10-mar-2021].

- [33] SMSGlobal, <<SMSGlobal,>>, 2021. [En línea]. Disponible en: https://www.smsglobal.com/. [Consultado: 10-mar-2021].

- [34] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/introduction/about. [Consultado: 10-mar-2021].

- [35] M.A. Ulloa Solis, Implementación de plataforma de monitoreo Zabbix para sistemas de Telecomunicaciones TELSUR, Valdivia: Universidad Austral de Chile, 2014. [En línea]. Disponible en: http://cybertesis.uach.cl/tesis/uach/2014/bmfciu.41i/doc/bmfciu.41i.pdf. [Consultado: 10-mar-2021].

- [36] L.A. Gavilanes Rivera, Implementación de un sistema de monitoreo en el Data Center de la empresa Seguros Oriente S.A., Quito: Universidad de las Américas, 2016. [En línea]. Disponible en: http://dspace.udla.edu.ec/handle/33000/4914. [Consultado: 10-mar-2021].

- [37] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/introduction/features. [Consultado: 10-mar-2021].

- [38] mancomun, "Arquitectura de Zabbix", 2017. [En línea]. Disponible en: https://www.mancomun.gal/es/artigo-tic/zabbix-24-arquitectura/#:~:text=El%20dise%C3%B1o%20de%20Zabbix%20se,de%20una%20arquitectura%20en%20estrella.&text=Son%20sistemas%20propensos%20a%20la,acceso%20al%20mismo%20servidor%20central. [Consultado: 10-mar-2021].

- [39] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/concepts. [Consultado: 10-mar-2021].

- [40] Custos Monitoring, “Zabbix checks: Active Checks - Passive Checks”, 2020. [En línea]. Disponible en: https://custos.uy/zabbix-checks/. [Consultado: 11-mar-2021].

- [41] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/config/hosts. [Consultado: 11-mar-2021].

- [42] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/config/notifications/media. [Consultado: 11-mar-2021].

- [43] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/config/users_and_usergroups/permissions. [Consultado: 11-mar-2021].

- [44] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/web_monitoring/items. [Consultado: 11-mar-2021].

- [45] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/encryption. [Consultado: 12-mar-2021].

- [46] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/api. [Consultado: 12-mar-2021].

- [47] Zabbix, <<Python,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/la/integrations/python. [Consultado: 12-mar-2021].

- [48] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/api/reference/application/create. [Consultado: 12-mar-2021].

- [49] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/api/reference/trigger. [Consultado: 12-mar-2021].

- [50] Zabbix, <<Zabbix Documentation 5.2,>>, 2021. [En línea]. Disponible en: https://www.zabbix.com/documentation/current/manual/api/reference/httptest. [Consultado: 12-mar-2021].

- [51] J.A. Mogrovejo Bucheli, Implementación del ERP OPEN SOURCE ODOO en una PYME, Guayaquil: Escuela Superior Politécnica del Litoral, 2017. [En línea]. Disponible en: https://www.dspace.espol.edu.ec/handle/123456789/38698. [Consultado: 15-mar-2021].

- [52] P. Sastre Pons, Desarrollo de una aplicación para ODOO ERP, Valencia: Universitat Politécnica de Valencia, 2019. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/151984. [Consultado: 15-mar-2021].

- [53] DekaLabs, “Los beneficios de implementar Odoo en tu empresa”, 2020. [En línea]. Disponible en: https://dekalabs.com/los-beneficios-de-implentar-odoo-en-tu-empresa/. [Consultado:16-mar-2021].

- [54] Odoo, <<Odoo Documentation,>>, 2021. [En línea]. Disponible en: https://www.odoo.com/documentation/14.0/webservices/odoo.html. [Consultado: 16-mar-2021].

- [55] arsys, “Comó funcionan Docker y sus componentes”, 2019. [En línea]. Disponible en: https://www.arsys.es/blog/soluciones/infraestructura/como-funcionan-contenedores-docker. [Consultado: 18-mar-2021].

- [56] Slideshare, “An evening with Docker”, 2017. [En línea]. Disponible en: https://www.slideshare.net/arkhotech/an-evening-with-docker. [Consultado: 18-mar-2021].

- [57] Red Hat, “¿Qué es Docker?”, 2021. [En línea]. Disponible en: https://www.redhat.com/es/topics/containers/what-is-docker#:~:text=%22Docker%22%2C%20el%20software%20de,los%20usuarios%20de%20forma%20gratuita. [Consultado: 18-mar-2021].

- [58] Agilia, “Docker Compose; concepto, funcionalidades y retos que afronta”, 2021. [En línea]. Disponible en: https://www.agiliacenter.com/funciones-docker-compose/. [Consultado: 18-mar-2021].

ANEXOS

6. ANEXOS

ANEXO A - PLANTILLA DE ENTREVISTA

A.1 PLANTILLA DE LA ENTREVISTA PARA LOS GERENTES DE LA EMPRESA TRESCLOUD CIA LTDA

1. ¿Existe en la empresa algún tipo de monitoreo para sus clientes?

2. ¿Cuántas métricas monitorea el sistema actual?

3. ¿Cómo afecta el sistema de monitoreo en su área de gerencia?

4. ¿Estaría dispuesto a invertir en licenciamiento de software para un sistema de monitoreo más completo?

5. ¿Qué mejoras le gustaría tener en el sistema de monitoreo que tienen la empresa actualmente?

ANEXO B - USUARIOS Y CONTRASEÑAS PARA INGRESAR A ZABBIX Y ODOO

B.1 USUARIO Y CONTRASEÑAS EN ZABBIX

- Credenciales para ingresar al servidor de Zabbix.

USUARIO ZABBIX	DIRECCIÓN IP	CONTRASEÑA ZABBIX
Admin	192.168.0.10:8081	zabbix

- Credenciales para ingresar al servidor Zabbix con distintos usuarios y diferentes tipos de permiso.

USUARIO ZABBIX	CONTRASEÑA ZABBIX	TIPO DE PERMISO
Ricardo Rangles	ricardo008	Super administrador
Pablo Hidalgo	pablohidalgo	Administrador
Patricio Rangles	pato123	Usuario
Cristhian Ampudia	cristhianampudia	Usuario

B.2 USUARIO Y CONTRASEÑAS EN ODOO

- Credenciales para ingresar al servidor Odoo especial.

USUARIO ODOO	DIRECCIÓN IP	EMAIL	CONTRASEÑA ODOO
Zabbix-Odoo14	192.168.0.125:8069	ricardo_rangles@hotmail.com	zabbixodoo

- Credenciales para ingresar a los clientes Odoo.

USUARIO ODOO	DIRECCIÓN IP	EMAIL	CONTRASEÑA ODOO
Cliente Odoo A	192.168.0.11:8069	ricardo_rangles@hotmail.com	servera
Cliente Odoo B	192.168.0.12:8069	ricardo_rangles@hotmail.com	serverb
Cliente Odoo C	192.168.0.13:8069	ricardo_rangles@hotmail.com	serverc

ANEXO C - CÓDIGOS DEL PROYECTO

C.1 CÓDIGO DEL MÓDULO DE INTERCONEXIÓN ODOO – ZABBIX

C.1.1 ARCHIVO RES_COMPANY.PY

```
1.  # -*- coding: utf-8 -*-
2.  #Library that imports the Odoo methods necessary for this module
3.  from odoo import models, fields, api
4.
5.  #Creation of the ResPartner class for handling buttons in Odoo
6.  class ResCompany(models.Model):
7.
8.  #Odoo res.company module inheritance
9.  _inherit = 'res.company'
10.
11. #Columns
12. host_id = fields.Char(string="Host ID")
13. url = fields.Char(string="URL")
14. user = fields.Char(string="User")
15. password = fields.Char(string="Password")
```

C.1.2 ARCHIVO ZABBIX_ODOO_LIBRARY.PY

```
1.  # -*- coding: utf-8 -*-
2.  #Library that imports the Odoo methods necessary for this module
3.  from odoo import models, fields, api
4.
5.  #Library that imports the necessary Zabbix methods for module interconnection
6.  from pyzabbix.api import ZabbixAPI
7.
8.  #Library that imports notifications and errors to the user
9.  from odoo.exceptions import ValidationError
10.
11. #library that allows working with dates and times
12. from datetime import datetime
13.
14. #Variable AGENT indicates with which agent is going to connect
15. AGENT = "Zabbix"
16.
17. #Creation of the LibreriaZabbix class to manage the interconnection between Zabbix and
    Odoo
18. class LibreriaZabbix(models.AbstractModel):
19.
20. _name = 'zabbix.odoo.library'
21.
22. def create_monitor_client(self, instance_id, website):
```

```
23. """
24. This function is to create a client to monitor in Zabbix using the button
25. "Add client for monitoring" created in the contact form of Odoo v14
26. """
27. #Unique code of the client instance
28. instance_id = str(instance_id)
29. #Value of the first item to search in each field of res.company
30. company_id = 1
31.
32. #Validation in case the host_id variable is null
33. if not self.env["res.company"].browse(company_id).host_id:
        raise ValidationError("The host_id of your Zabbix server has not been configured.
        Go to Settings -> Companies -> Select your company -> Zabbix tab and
        configure")
34. #Variable host_id indicates the id of the server where the instances are located
35. host_id = self.env["res.company"].browse(company_id).host_id
36.
37. #Validation in case the url variable is null
38. if not self.env["res.company"].browse(company_id).url:
        raise ValidationError("The url of your Zabbix server has not been configured. Go
        to Settings -> Companies -> Select your company -> Zabbix tab and configure")
39. #Variable url indicates the URL of the Zabbix server where it is going to connect
40. url = self.env["res.company"].browse(company_id).url
41.
42. #Validation in case the user variable is null
```

```
43. if not self.env["res.company"].browse(company_id).user:
        raise ValidationError("The user of your Zabbix server has not been configured.
        Go to Settings -> Companies -> Select your company -> Zabbix tab and
        configure")
44. #Variable user indicates the user to login to the Zabbix server
45. user = self.env["res.company"].browse(company_id).user
46.
47. #Validation in case the password variable is null
48. if not self.env["res.company"].browse(company_id).password:
        raise ValidationError("The password of your Zabbix server has not been
        configured. Go to Settings -> Companies -> Select your company -> Zabbix tab
        and configure")
49. #Variable password indicates the password to log in to the Zabbix server
50. password = self.env["res.company"].browse(company_id).password
51. #Validation for the connection with the Zabbix server
52. try:
        zapi = ZabbixAPI(url=url, user=user, password=password)
53. except:
        raise ValidationError("Error connecting to Zabbix server")


54. #Each URL is verified to avoid duplication and errors
55. web_list = zapi.httptest.get(hostids=host_id)
56. #Search for website name in list
57. for web in web_list:
        #If find the same instance, it is already created
```

```
            if instance_id in web['name']:
            #It's verified that the unique id and the URL is the same as the website name
            if instance_id + " - " + website == web['name']:
                #The function returns the value 0 when the URL already exists in Zabbix
                return 0
            else:
                #The function returns the value of 2 when the URL doesn't match the one
                sent in Zabbix
                return 2
58. #Function that checks if the Website application exists in the Zabbix monitoring
59. application_id = self.verify_zabbix_application(zapi)
60. zapi.httptest.create(
            name= instance_id + " - " + website,
            hostid=host_id,
            applicationid = application_id,
            agent=AGENT,
            steps=[
            {
                "name": "Homepage",
                "url": website,
                "status_codes": "200",
                "no": 1
            },
            ]
```

```
61. )
62.
63. zapi.trigger.create(
        description="Web scenario " + instance_id + " - " + website + " failed:
        {ITEM.VALUE}",
        priority=4,
        expression="{Zabbix server:web.test.error[" + instance_id + " - " + website +
        "].strlen()}>0 and {Zabbix server:web.test.fail[" + instance_id + " - " + website +
        "].last()}>0"
64. )
65. #The function returns 1 when the URL has been correctly added in Zabbix
66. return 1
67.
68. def remove_monitor_client(self, instance_id, website):
69. """
70. This function is to delete a client to monitor in Zabbix using the button
71. "Delete client for monitoring" created in the contact form of Odoo v14
72. """
73. instance_id = str(instance_id)
74. company_id = 1
75.
76. if not self.env["res.company"].browse(company_id).host_id:
        raise ValidationError("The host_id of your Zabbix server has not been configured.
        Go to Settings -> Companies -> Select your company -> Zabbix tab and
        configure")
77. host_id = self.env["res.company"].browse(company_id).host_id
```

```
78.
79. if not self.env["res.company"].browse(company_id).url:
        raise ValidationError("The url of your Zabbix server has not been configured. Go to Settings -> Companies -> Select your company -> Zabbix tab and configure")
80. url = self.env["res.company"].browse(company_id).url
81.
82. if not self.env["res.company"].browse(company_id).user:
        raise ValidationError("The user of your Zabbix server has not been configured. Go to Settings -> Companies -> Select your company -> Zabbix tab and configure")
83. user = self.env["res.company"].browse(company_id).user
84.
85. if not self.env["res.company"].browse(company_id).password:
        raise ValidationError("The password of your Zabbix server has not been configured. Go to Settings -> Companies -> Select your company -> Zabbix tab and configure")
86. password = self.env["res.company"].browse(company_id).password
87.
88. try:
        zapi = ZabbixAPI(url=url, user=user, password=password)
89. except:
        raise ValidationError("Error al conectarse con el servidor Zabbix")
90. web_list = zapi.httptest.get(hostids=host_id)
91. web_found = False
92. for web in web_list:
        if instance_id in web['name']:
```

```
                web_found = True
                #It's checked if the unique id and the URL are equal to the website name
                if instance_id + " - " + website == web['name']:
                        zapi.httptest.delete(
                        httptestid=web['httptestid']
                        )
                        #The function returns 1 when the URL has been correctly removed in
                        Zabbix
                        return 1
    93. if not web_found:
                #The function returns 0 when the client ID with the URL is not being monitored in
                Zabbix
                return 0
    94. def verify_zabbix_application(self, zapi):
    95. """
    96. This function is to check if the Website application exists in the Zabbix monitoring system
    97. """
    98. application_list = zapi.application.get(search={"name":"Website"}))
    99. if application_list:
                applicationid = application_list[0]["applicationid"]
else:
                applicationid = zapi.application.create(
                name="Website"
                )
    100.        return applicationids
```

```
101.
102.        def restart_instance_odoo(self, params, id, url):
103.        """
104.        This function allows identifying the id and url of the instance that is going to be
            restarted in odoo,
105.        in addition to that this function must be inherited and rewritten by whoever needs
            to use it later.
106.        """
107.        #Browse method uses integers not strings
108.        ID = int(id)
109.
110.        #The associated res.partener is removed to modify your comments
111.        contact = self.env["res.partner"].browse(ID)
112.
113.        #Message that appears in the internal notes of the contact
114.        message = "Instance Fail: " + datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            + ": " ID Instancia: " + id + " URL: " + url + "\n"

115.        #If the comment has nothing, it puts the message, otherwise it adds it
116.        if not contact.comment:
                contact.comment = message
117.        else:
                contact.comment += message
118.        return True
```

C.1.3 ARCHIVO _INIT_.PY

```
1. # -*- coding: utf-8 -*-
2.
3. from . import zabbix_odoo_library
4. from . import res_company
```

C.1.4 ARCHIVO RES_COMPANY_VIEW.XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <odoo>
3. <record id="view_company_form_zabbix_odoo_library" model="ir.ui.view">
4. <field name="name">view.company.form.zabbix.odoo.library</field>
5. <field name="model">res.company</field>
6. <field name="inherit_id" ref="base.view_company_form"/>
7. <field name="arch" type="xml">
                        <data>
                            <xpath expr="//notebook" position="inside">
                                <page string="Zabbix" name='zabbix'>
                                    <group>
                                        <group name="group_zabbix"
                                        string="Zabbix">

                <field name="host_id"/>
                <field name="url"/>
                <field name="user"/>
```

```
                    <field name="password"/>
                                                    </group>
                                                </group>
                                            </page>
                                        </xpath>
                                    </data>
                                </field>
                            </record>
</odoo>
```

C.2 CÓDIGO DEL MÓDULO DE PRUEBA LLAMADO MODULO_PRUEBA_BOTONES

C.2.1 ARCHIVO RES_PARTNER.PY

```
1.  from odoo import models, fields, api
2.
3.  #Library that imports notifications and errors to the user
4.  from odoo.exceptions import ValidationError
5.
6.  class ResPartner(models.Model):
7.
8.  #Odoo res.partner module inheritance
9.  _inherit = 'res.partner'
10.
11. def create_client_button(self):
```

```
12. """
13. This function calls the create a client function to monitor in zabbix.
14. """
15. create_client_function = self.env["zabbix.odoo.library"].create_monitor_client(self.id,
    self.website)
16.
17. if create_client_function == 0:
        raise ValidationError("The URL already exists in Zabbix")
18. if create_client_function == 1:
        raise ValidationError("The URL has been added correctly in Zabbix")
19. if create_client_function == 2:
        raise ValidationError("The URL doesn't match the one sent in Zabbix")


20. def remove_client_button(self):
21. """
22. This function calls the remove a client function to monitor it in zabbix.
23. """
24. delete_client_function = self.env["zabbix.odoo.library"].remove_monitor_client(self.id,
    self.website)
25.
26. if delete_client_function == 0:
        raise ValidationError("Client ID with URL is not being monitored in Zabbix")
27. if delete_client_function == 1:
        raise ValidationError("The URL has been successfully removed in Zabbix")
```

C.2.2 ARCHIVO _INIT_.PY

```
1. # -*- coding: utf-8 -*-
2.
3. from . import res_partner
```

C.2.3 ARCHIVO RES_PARTNER_VIEW.XML

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <odoo>
3. <data noupdate='0'>
4. <record id="view_partner_form_zabbix" model="ir.ui.view">
5. <field name="name">res.partner.form</field>
6. <field name="model">res.partner</field>
7. <field name="inherit_id" ref="base.view_partner_form"/>
8. <field name="arch" type="xml">
        <field name="website" position="after">
        <button string="Añadir cliente para monitoreo" name="create_client_button"
        type="object" class="oe_highlight"/>
        <button string="Eliminar cliente para monitoreo" name="remove_client_button"
        type="object" class="oe_highlight"/>
        </field>
9. </field>
10. </record>
11. </data>
12. </odoo>
```

C.3 CÓDIGO DEL SCRIPT DE ENVÍO DE NOTIFICACIONES DESDE ZABBIX MEDIANTE MENSAJES SMS

C.3.1 ARCHIVO SEND_SMS.SH

```
1.  #!/bin/sh
2.  #
3.  #Script to send SMS alerts via Zabbix
4.  #
5.  #More information about the SMS Global HTTP API: https://www.smsglobal.com/http-api/
6.  #
7.  #Sending the output to a file. I am getting that file's data into Zabbix.
8.  exec 3>&1 4>&2
9.  trap 'exec 2>&4 1>&3' 0 1 2 3
10.
11. exec 1>/tmp/send_sms_log.out 2>&1
12.
13. #SMSGLOBAL credentials
14. USERNAME=1nako4ak
15. PASSWORD=T73yKf3B
16. PHONE=${1}
17. MESSAGE=${2}
18.
19. #echo "date +%H:%M:%: Sending SMS Text to $1"
20. #
21. #Posting the SMS via HTTP API
```

```
22. #
23. #NOTE: Because the Zabbix alerts have spaces/new lines, we are using the "--data-urlencode" option to encode the message.
24. curl -X POST "https://api.smsglobal.com/http-api.php?action=sendsms&user=$USERNAME&password=$PASSWORD&from=Test&to=$PHONE" --data-urlencode text="$MESSAGE"
25. STATUS=$?
26. echo "SMS Text sent to $1"
27. echo "$MESSAGE"
28. exit $STATUS
```

C.4 CÓDIGO DEL SCRIPT DE ENVÍO DE NOTIFICACIÓN A ODOO CUANDO UN CLIENTE CAYÓ O TUVO PROBLEMAS

C.4.1 ARCHIVO INSTANCE_REBOOT.PY

```
1.  # -*- coding: utf-8 -*-
2.  import erppeek
3.  import sys
4.  import os
5.
6.  #Create a text file as log
7.  file = open("/tmp/log_odoo.txt",'w')
8.
9.  #Parameters required for connection to the Odoo server
10. SERVER = sys.argv[1]
```

```
11. DATABASE = sys.argv[2]
12. USERNAME = sys.argv[3]
13. PASSWORD = sys.argv[4]
14. DATA = sys.argv[5]
15. message = DATA
16.
17. #Check if the event is due to a fallen website
18. if not "Problem name: Web scenario" in message:
19. file.close()
20. exit(0)
21.
22. #Detach the captured message to find the id and url of the crashed Odoo instance
23. section = message.split("Problem name: Web scenario")
24.
25. ID_URL = section[1].split(' - ')
26.
27. ID = ID_URL[0].strip()
28.
29. URL = ID_URL[1].split(' failed:')[0]
30.
31. #Odoo API connection
32. client = erppeek.Client(SERVER, DATABASE, USERNAME, PASSWORD,
    verbose=True)
33.
```

```
#Write in the log to verify what parameters were sent
file.write("ID: " + ID + os.linesep)
file.write("URL:" + URL + os.linesep)

#Method to execute the reset method of the zabbbix_odoo_library.py library
client.execute('zabbix.odoo.library','restart_instance_odoo',[],{},ID,URL)

#Closing the log_odoo.txt file
file.close()
```

C.5 CÓDIGO DEL SCRIPT QUE LLAMA AL SCRIPT INSTANCE_REBOOT.PY

C.5.1 ARCHIVO CALL_FUNCTION_PYTHON.SH

```
#!/bin/sh
#
# Script to call the script made in python to restart an instance in odoo
#

#Write in log to check the values coming from Zabbix
echo "parametros que llegaron: $*" >>  /tmp/send_odoo_log.out

#Python script execution
python3 /usr/lib/zabbix/alertscripts/instance_reboot.py $1 $2 $3 $4 "$5"
```

ANEXO D - MANUAL DE INSTRUCCIONES Y CONFIGURACIONES

D.1. INSTALACIÓN DEL SERVIDOR DE PRUEBA PARA ZABBIX

1. Buscar en la página https://releases.ubuntu.com/18.04/ la imagen de Ubuntu Server 18.04.5 LTS.

2. Descargar la imagen de Ubuntu Server 18.04.5 LTS para instalarlo en la máquina virtual en VirtualBox.

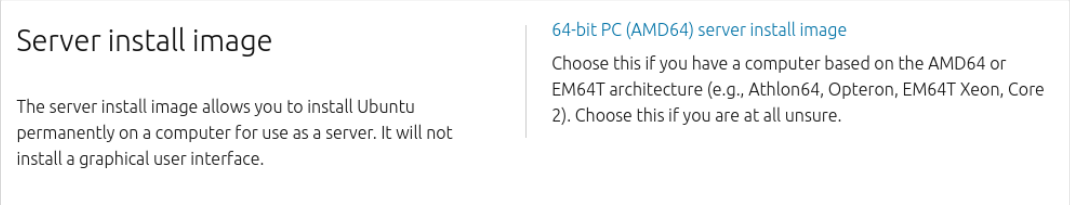


Figura D.1 Imagen para instalar Ubuntu server 18.04.5 LTS

3. Crear la máquina virtual donde se instalará el Ubuntu Server 18.04.5 LTS.

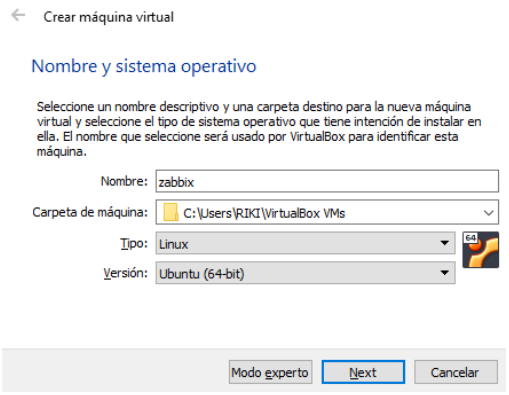


Figura D.2 Creación de la máquina virtual

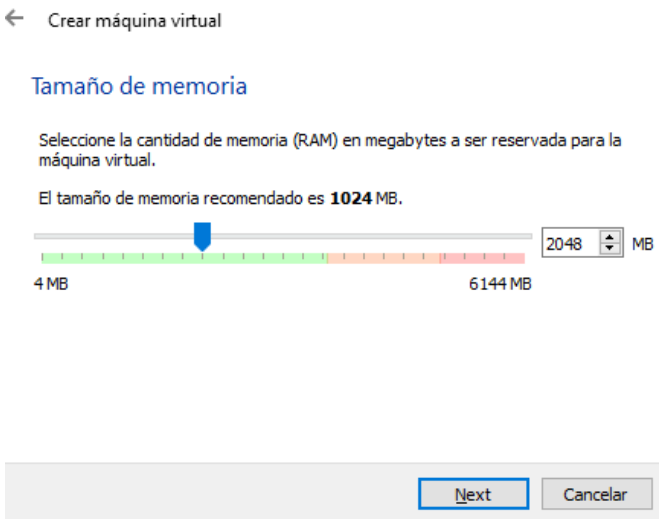


Figura D.3 Tamaño de memoria de la máquina virtual

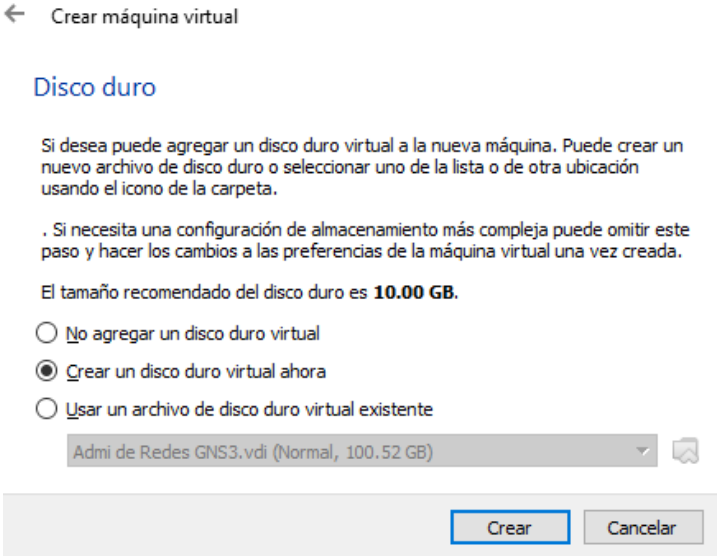


Figura D.4 Disco duro de la máquina virtual

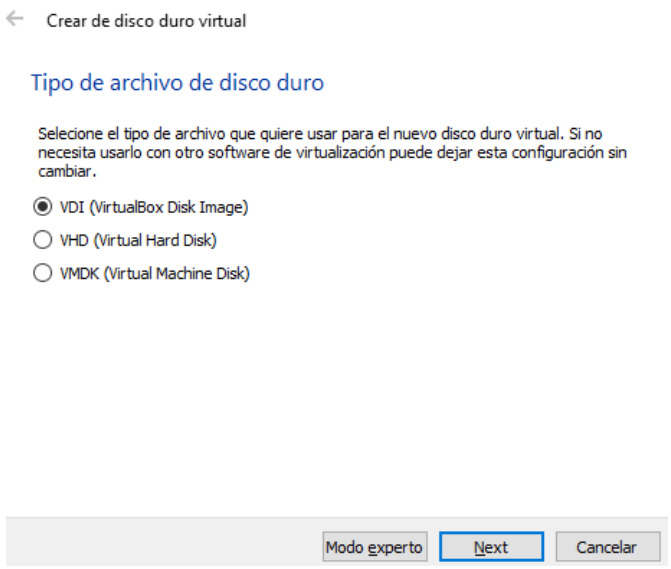


Figura D.5 Tipo de archivo de disco duro de la máquina virtual

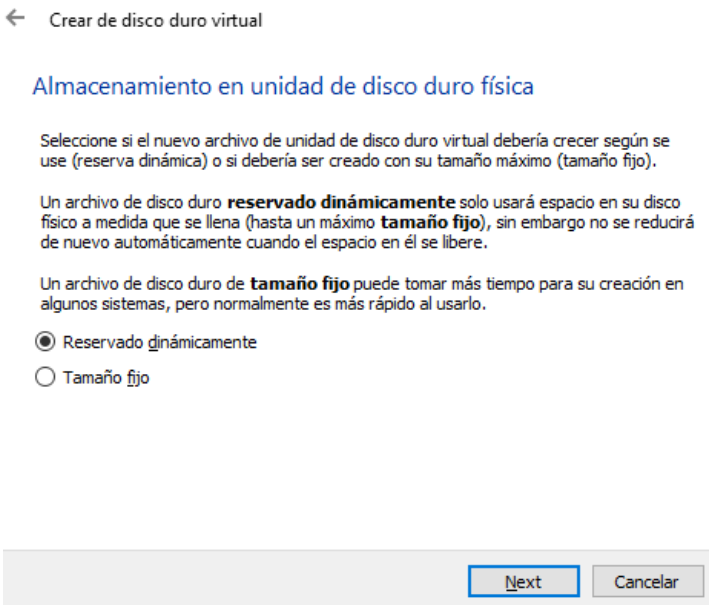


Figura D.6 Almacenamiento en unidad de disco duro físico de la máquina virtual

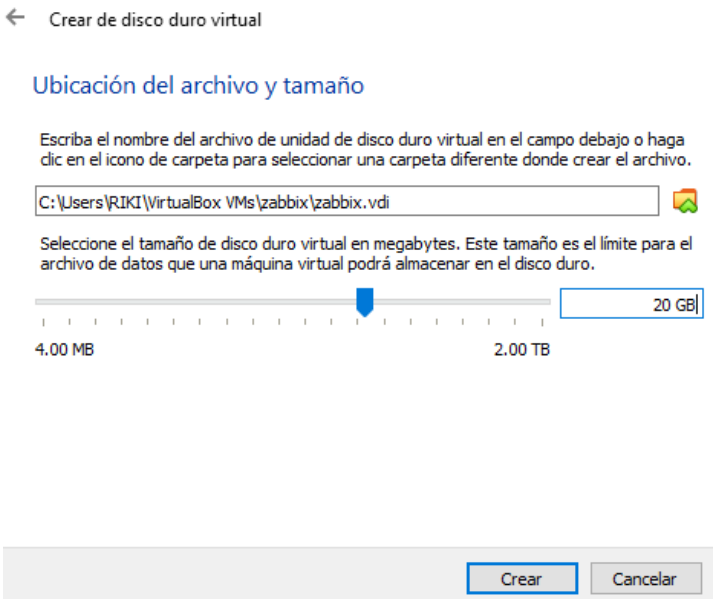


Figura D.7 Ubicación del archivo y tamaño de la máquina virtual

Figura D.8 Creación final de la máquina virtual

4. Una vez creada la máquina virtual se debe instalar el sistema operativo Ubuntu Server 18.04.5 LTS.

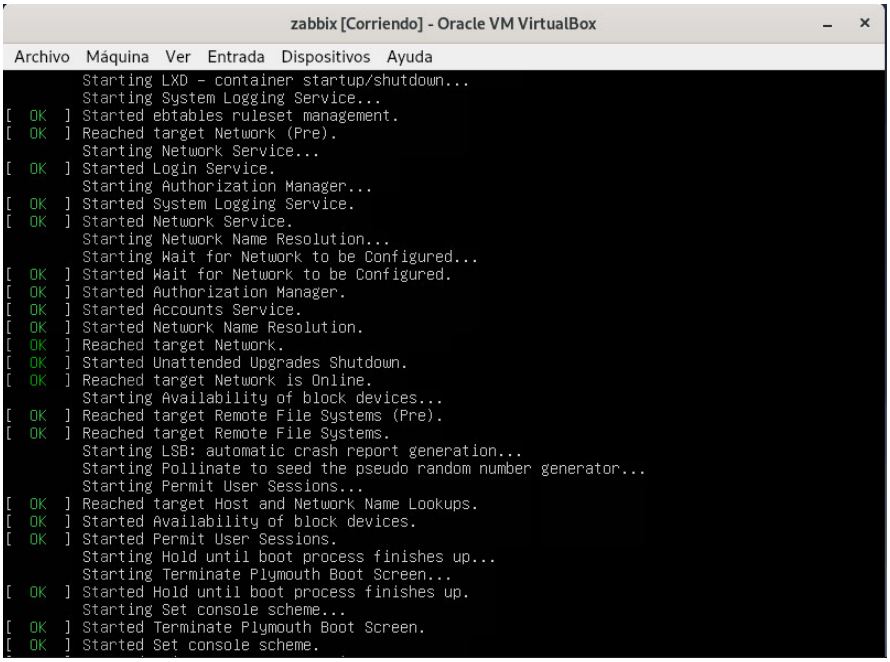


Figura D.9 Arranque de la máquina virtual

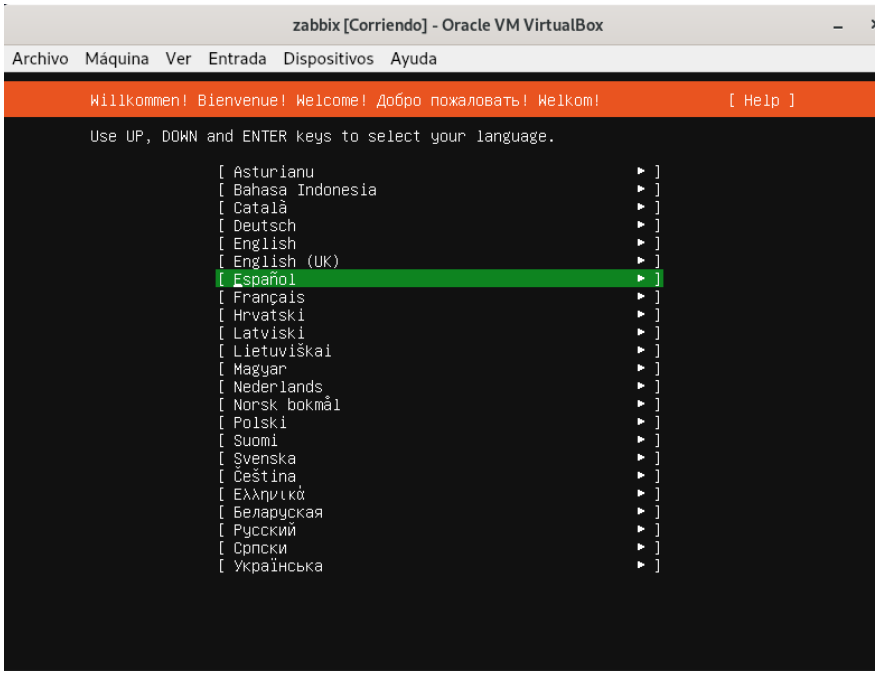


Figura D.10 Inicio de instalación de Ubuntu Server 18.04.5 LTS

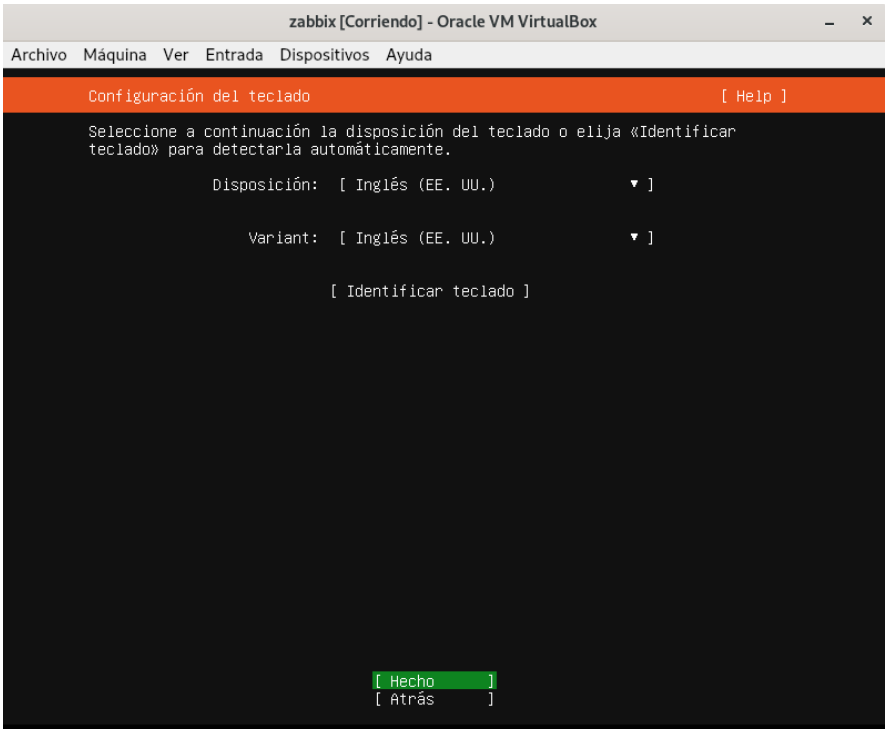


Figura D.11 Configuración del teclado

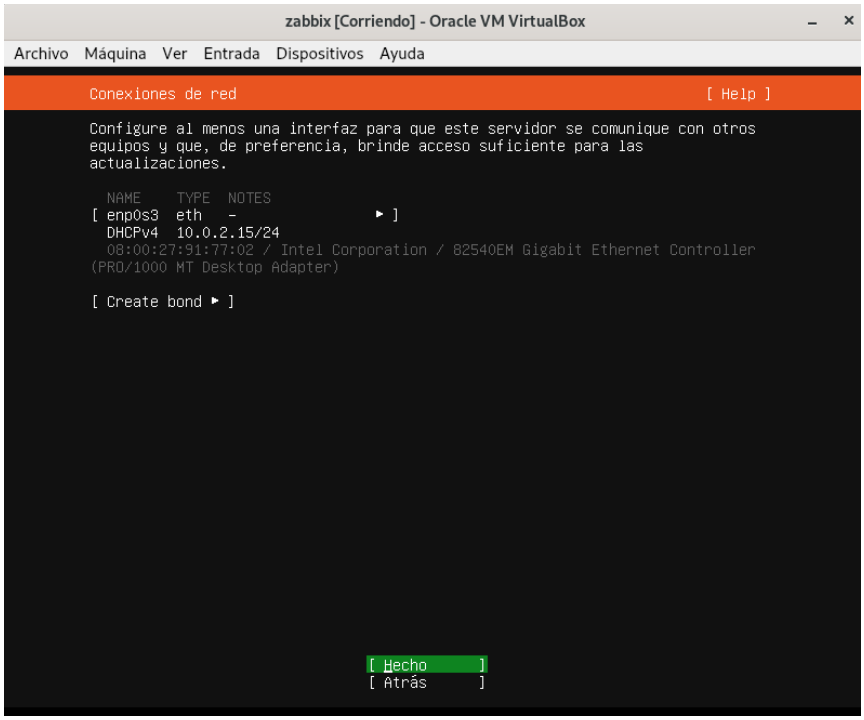


Figura D.12 Configuración de la conexión de red

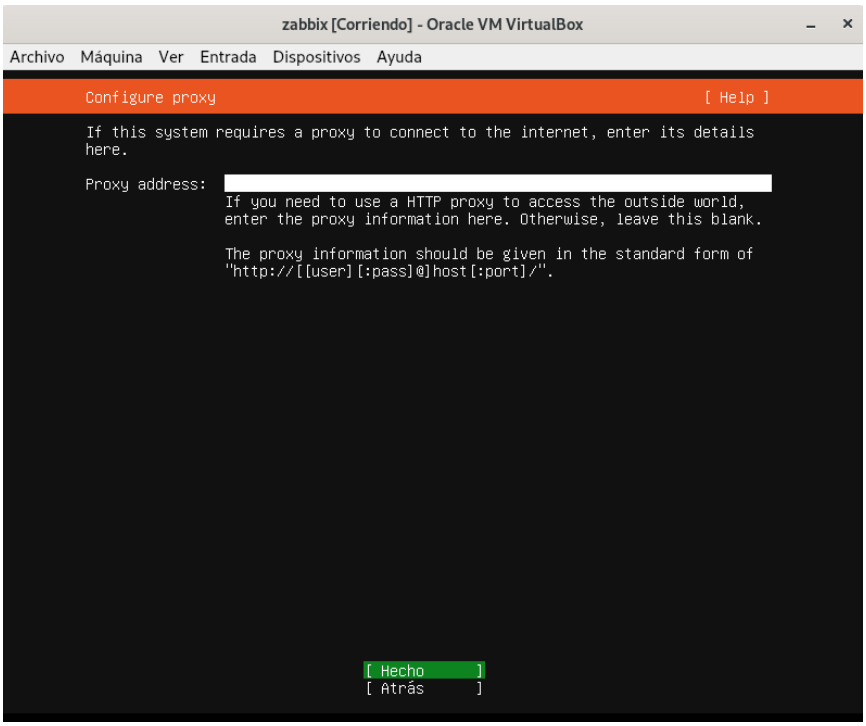


Figura D.13 Configuración del proxy

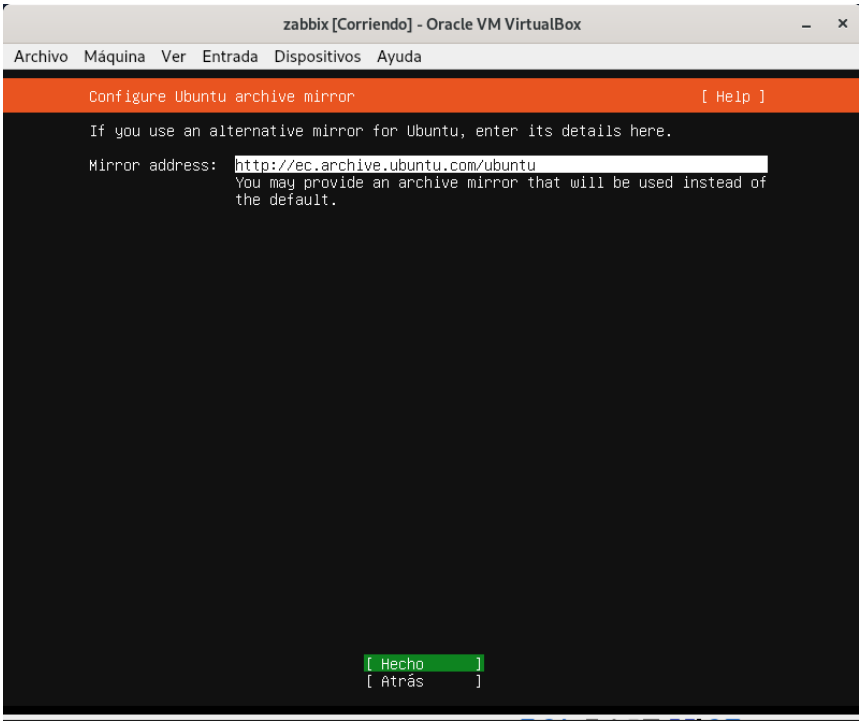


Figura D.14 Configuración de los archivos espejo en Ubuntu

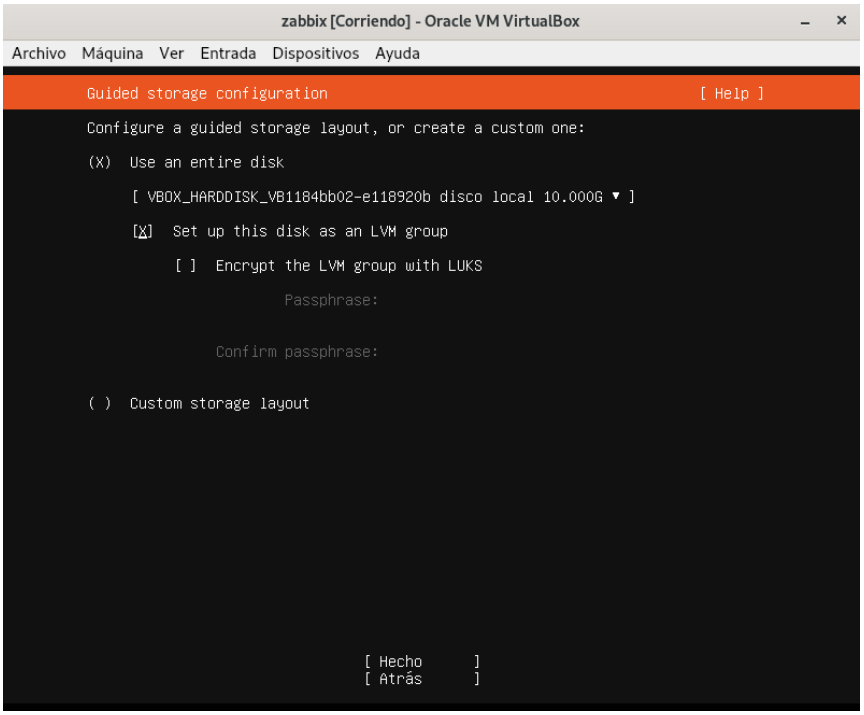


Figura D.15 Configuración del almacenamiento

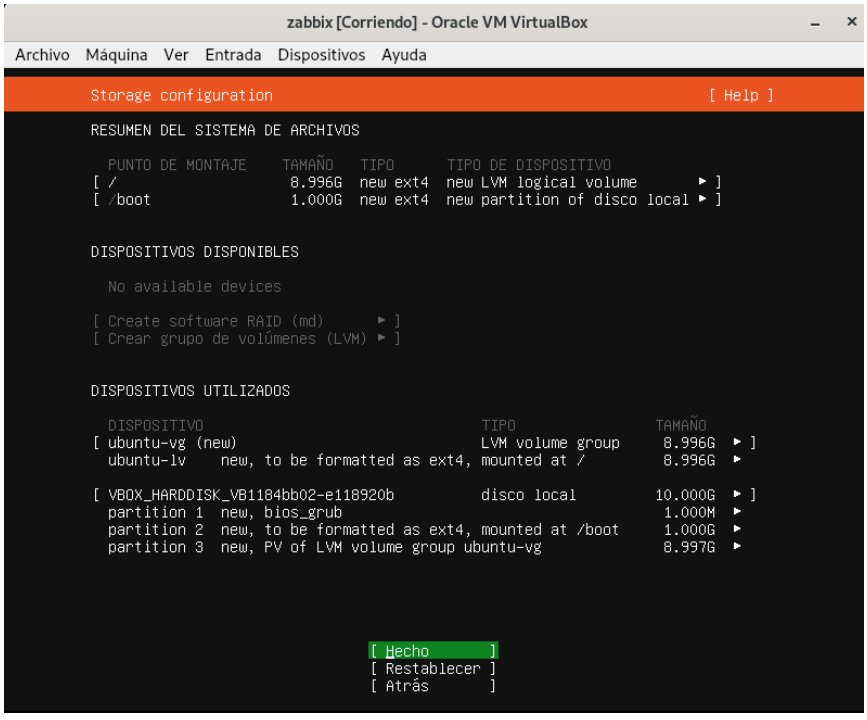


Figura D.16 Resumen de las configuraciones hechas

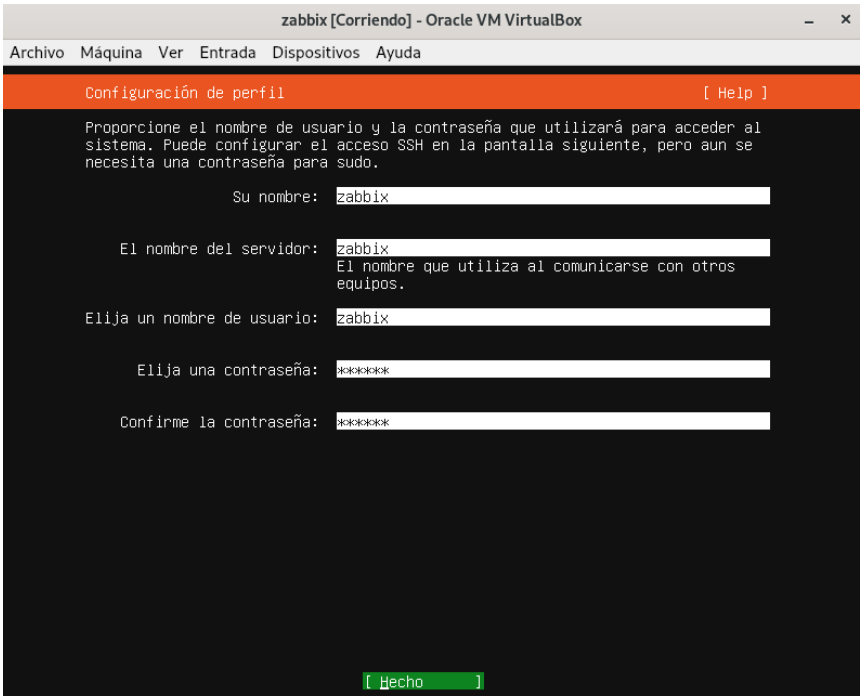


Figura D.17 Configuración de perfil

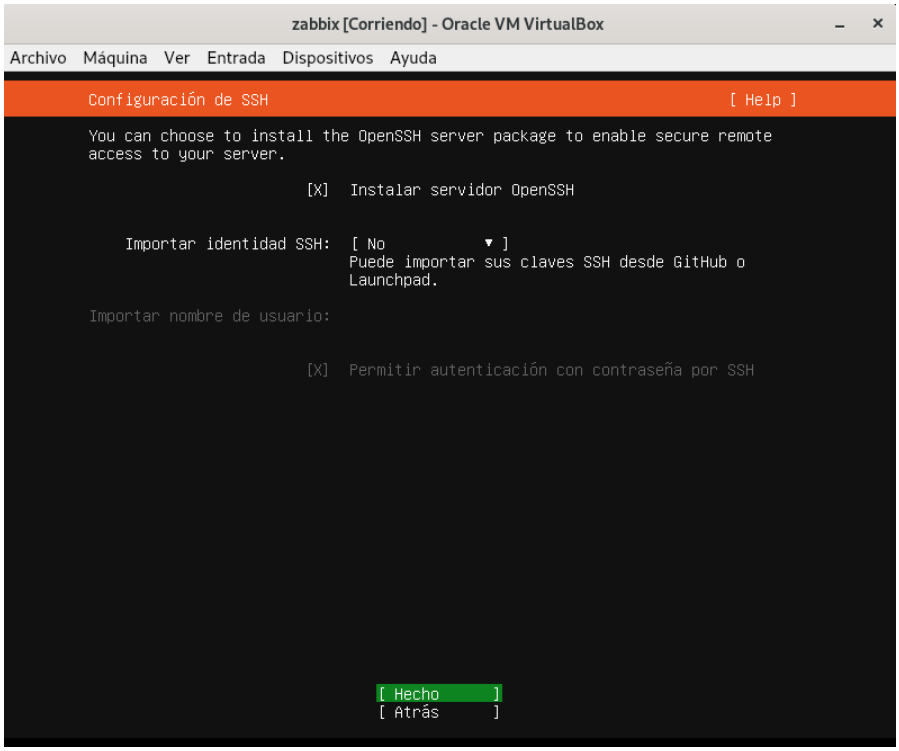


Figura D.18 Configuración de SSH

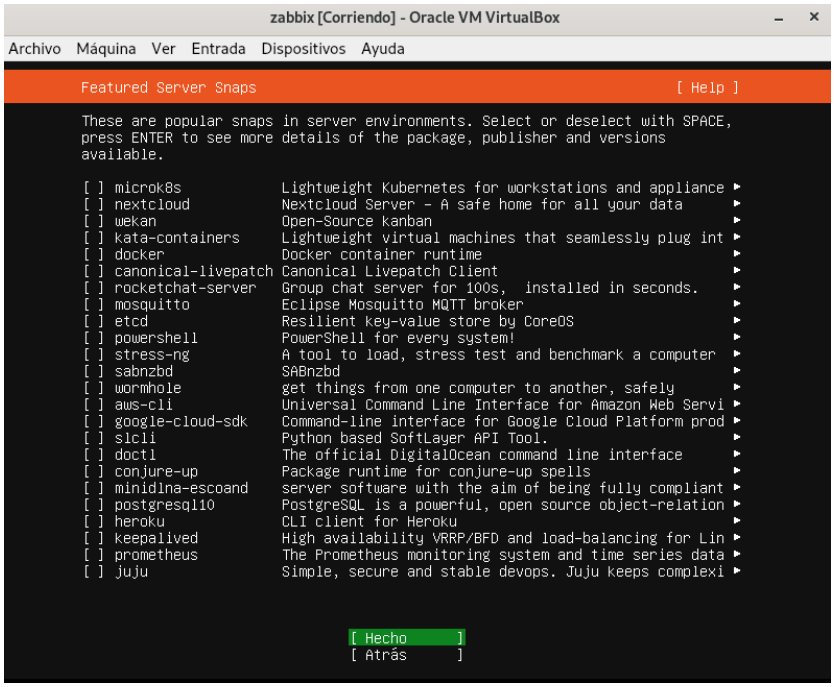


Figura D.19 Configuración de entornos opcionales

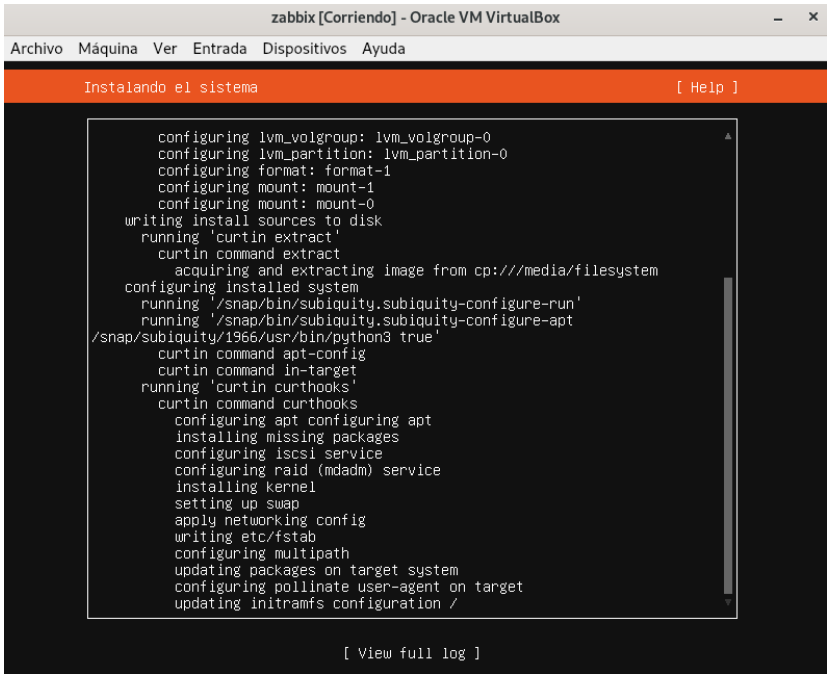


Figura D.20 Instalación de Ubuntu Server 18.04.5 LTS

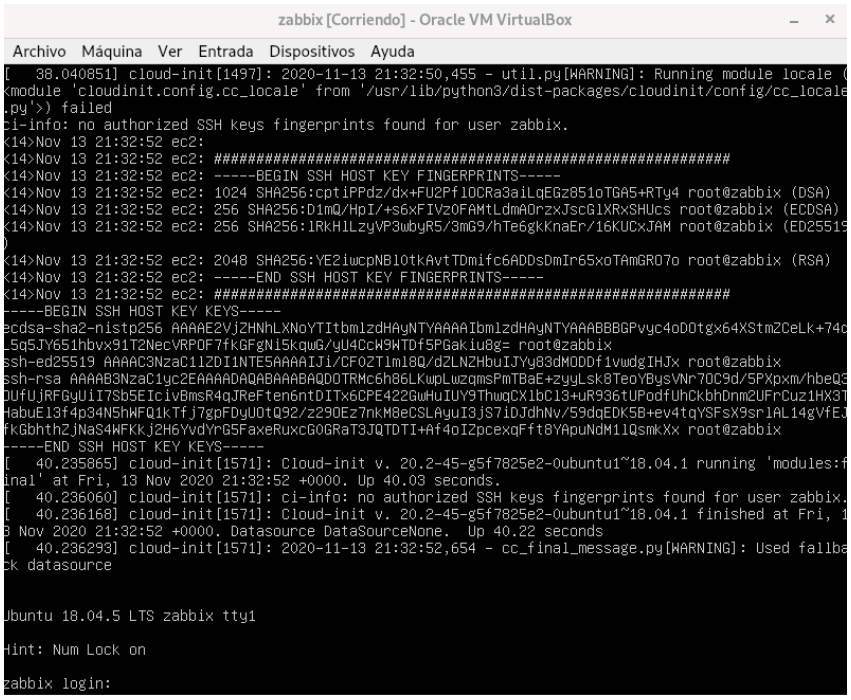


Figura D.21 Finalización de la instalación de Ubuntu Server 18.04.5 LTS

5. Teniendo ya instalado el sistema operativo Ubuntu Server 18.04.5 LTS se procede a instalar la tecnología Docker para poder virtualizar el servicio de software de monitoreo Zabbix.

a) Actualizar el índice de paquetes de apt e instalar los paquetes que permiten que apt use un repositorio sobre HTTPS:

```
zadocker@zabbixdocker:~$ sudo apt-get update
[sudo] password for zadocker:
Hit:1 http://ec.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://ec.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://ec.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://ec.archive.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 2s (123 kB/s)
Reading package lists... Done
```

Figura D.22 Actualización de paquetes en Ubuntu

```
zadocker@zabbixdocker:~$ sudo apt-get install \
>     apt-transport-https \
>     ca-certificates \
>     curl \
>     gnupg-agent \
>     software-properties-common
Reading package lists... Done
Building dependency tree
Reading state information... Done
ca-certificates is already the newest version (20201027ubuntu0.18.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.58.0-2ubuntu3.10).
curl set to manually installed.
software-properties-common is already the newest version (0.96.24.32.14).
software-properties-common set to manually installed.
The following NEW packages will be installed:
  apt-transport-https gnupg-agent
0 upgraded, 2 newly installed, 0 to remove and 25 not upgraded.
Need to get 6568 B of archives.
After this operation, 196 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ec.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 apt-transport-https all 1.6.12ubuntu0.1 [1692 B]
Get:2 http://ec.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 gnupg-agent all 2.2.4-1ubuntu1.3 [4876 B]
Fetched 6568 B in 1s (12.8 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 67106 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_1.6.12ubuntu0.1_all.deb ...
Unpacking apt-transport-https (1.6.12ubuntu0.1) ...
Selecting previously unselected package gnupg-agent.
Preparing to unpack .../gnupg-agent_2.2.4-1ubuntu1.3_all.deb ...
Unpacking gnupg-agent (2.2.4-1ubuntu1.3) ...
Setting up apt-transport-https (1.6.12ubuntu0.1) ...
Setting up gnupg-agent (2.2.4-1ubuntu1.3) ...
```

Figura D.23 Instalación de paquetes en Ubuntu

b) Agregar la clave GPG oficial de Docker

```
zadocker@zabbixdocker:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
```

Figura D.24 Agregación de la clave GPG oficial de Docker

c) Verificar la llave de la huella digital

```
zadocker@zabbixdocker:~$ sudo apt-key fingerprint 0EBFCD88
pub   rsa4096 2017-02-22 [SCEA]
      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88
uid           [ unknown] Docker Release (CE deb) <docker@docker.com>
sub   rsa4096 2017-02-22 [S]
```

Figura D.25 Verificación de la llave digital

d) Configurar el repositorio estable

```
zadocker@zabbixdocker:~$ sudo add-apt-repository \
>    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
>    $(lsb_release -cs) \
>    stable"
Hit:1 http://ec.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://ec.archive.ubuntu.com/ubuntu bionic-updates InRelease
Get:3 https://download.docker.com/linux/ubuntu bionic InRelease [64.4 kB]
Hit:4 http://ec.archive.ubuntu.com/ubuntu bionic-backports InRelease
Get:5 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages [13.0 kB]
Hit:6 http://ec.archive.ubuntu.com/ubuntu bionic-security InRelease
Fetched 77.4 kB in 1s (83.1 kB/s)
Reading package lists... Done
```

Figura D.26 Instalación del repositorio estable de Docker

```
zadocker@zabbixdocker:~$ sudo apt-get update
Hit:1 https://download.docker.com/linux/ubuntu bionic InRelease
Hit:2 http://ec.archive.ubuntu.com/ubuntu bionic InRelease
Hit:3 http://ec.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:4 http://ec.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:5 http://ec.archive.ubuntu.com/ubuntu bionic-security InRelease
Reading package lists... Done
```

e) Instalar el motor de Docker

Figura D.27 Actualización de los repositorios del sistema

```
zadocker@zabbixdocker:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  aufs-tools cgroupfs-mount libltdl7 pigz
The following NEW packages will be installed:
  aufs-tools cgroupfs-mount containerd.io docker-ce docker-ce-cli libltdl7 pigz
0 upgraded, 7 newly installed, 0 to remove and 25 not upgraded.
Need to get 91.3 MB of archives.
After this operation, 410 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://download.docker.com/linux/ubuntu bionic/stable amd64 containerd.io amd64 1.3.7-1 [24.4 MB]
Get:2 http://ec.archive.ubuntu.com/ubuntu bionic/universe amd64 pigz amd64 2.4-1 [57.4 kB]
Get:3 http://ec.archive.ubuntu.com/ubuntu bionic/universe amd64 aufs-tools amd64 1:4.9+20170918-1ubuntu1 [104 kB]
Get:4 http://ec.archive.ubuntu.com/ubuntu bionic/universe amd64 cgroupfs-mount all 1.4 [6320 B]
Get:5 http://ec.archive.ubuntu.com/ubuntu bionic/main amd64 libltdl7 amd64 2.4.6-2 [38.8 kB]
Get:6 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce-cli amd64 5:19.03.13~3-0~ubuntu-bionic [44.2 MB]
Get:7 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce amd64 5:19.03.13~3-0~ubuntu-bionic [22.5 MB]
Fetched 91.3 MB in 44s (2060 kB/s)
Selecting previously unselected package pigz.
(Reading database ... 67114 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
Unpacking pigz (2.4-1) ...
Selecting previously unselected package aufs-tools.
Preparing to unpack .../1-aufs-tools_1%3a4.9+20170918-1ubuntu1_amd64.deb ...
Unpacking aufs-tools (1:4.9+20170918-1ubuntu1) ...
Selecting previously unselected package cgroupfs-mount.
Preparing to unpack .../2-cgroupfs-mount_1.4_all.deb ...
Unpacking cgroupfs-mount (1.4) ...
Selecting previously unselected package containerd.io.
Preparing to unpack .../3-containerd.io_1.3.7-1_amd64.deb ...
Unpacking containerd.io (1.3.7-1) ...
```

Figura D.28 Instalación del motor de Docker

f) Verificar la versión de Docker

```
zadocker@zabbixdocker:~$ docker --version
Docker version 19.03.13, build 4484c46d9d
```

Figura D.29 Verificación de la versión de Docker

g) Instalar Docker Compose

```
zadocker@zabbixdocker:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/loca
l/bin/docker-compose
[sudo] password for zadocker:
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--  10
0   651  100   651    0     0   1251      0 --:--:-- --:--:-- --:--:--  1249
 55 11.6M   55 6612k    0     0  1307k      0  0:00:09  0:00:05  0:00:04 1 64 11.6M   64 7734k    0     0  1270k      0  0:00:09  0:00:06  0:00:03 1 7
3 11.6M   73 8804k    0     0  1247k      0  0:00:09  0:00:07  0:00:02 1 80 11.6M   80 9689k    0     0  1202k      0  0:00:09  0:00:08  0:00:01 1 89
100 11.6M  100 11.6M    0     0  1178k      0  0:00:10  0:00:10 --:--:-- 1040k
```

Figura D.30 Instalación Docker Compose

h) Dar permisos de ejecución

```
zadocker@zabbixdocker:~$ sudo chmod +x /usr/local/bin/docker-compose
```

Figura D.31 Permisos de ejecución

i) Verificar que funcione correctamente

```
zadocker@zabbixdocker:~$ docker-compose -v
docker-compose version 1.26.0, build d4451659
```

Figura D.32 Verificación de Docker Compose

6. Instalar Zabbix con la tecnología Docker

a) Clonar el repositorio zabbix-docker ubicado en Github

Ubicación del repositorio: https://github.com/RicardoRangles/zabbix-docker.git

Comando: *git clone https://github.com/RicardoRangles/zabbix-docker.git*

```
zadocker@zabbixdocker:~$ git clone https://github.com/zabbix/zabbix-docker.git
Cloning into 'zabbix-docker'...
remote: Enumerating objects: 1354, done.
remote: Counting objects: 100% (1354/1354), done.
remote: Compressing objects: 100% (578/578), done.
remote: Total 40797 (delta 912), reused 1057 (delta 715), pack-reused 39443
Receiving objects: 100% (40797/40797), 24.79 MiB | 520.00 KiB/s, done.
Resolving deltas: 100% (30033/30033), done.
```

Figura D.33 Ejemplo de clonación del repositorio zabbix-docker de Github

b) Para esta instalación se utilizó la versión 5.0 de Zabbix con el fin de hacer las pruebas, se cambia de rama a la versión 5.0 del repositorio zabbix-docker

```
zadocker@zabbixdocker:~/zabbix-docker$ git checkout -b 5.0 origin/5.0
Branch '5.0' set up to track remote branch '5.0' from 'origin'.
Switched to a new branch '5.0'
zadocker@zabbixdocker:~/zabbix-docker$ git status
On branch 5.0
Your branch is up to date with 'origin/5.0'.

nothing to commit, working tree clean
```

Figura D.34 Cambio de rama del repositorio zabbix-docker

c) Se modifica el archivo *docker-compose_v3_ubuntu_pgsql_local.yaml* se tiene al momento de clonar el repositorio. Se debe modificar el archivo solo con los servicios que se necesitan como Nginx, PostgreSQL. Para este caso se cambió el nombre del archivo a *docker-compose_v3_ubuntu_pgsql_latest_trescloud.yaml*

d) Se levanta el servicio de Docker Compose para que se construya los contenedores con los servicios necesarios para el funcionamiento de Zabbix con el siguiente comando:

docker-compose -f docker-compose_v3_ubuntu_pgsql_latest_trescloud.yaml up -d

e) Si existe algún reinicio inesperado y no funciona el servidor se debe bajar el servicio de Docker Compose y volverlo a subir:

docker-compose -f docker-compose_v3_ubuntu_pgsql_local.yaml down

docker-compose -f docker-compose_v3_ubuntu_pgsql_latest_trescloud.yaml up -d

f) Una vez levantado el servicio se ingresa a la dirección del servidor Zabbix, en este caso es la dirección 192.168.0.10:8080, las credenciales para ingresar son para el usuario: Admin y para la contraseña: zabbix

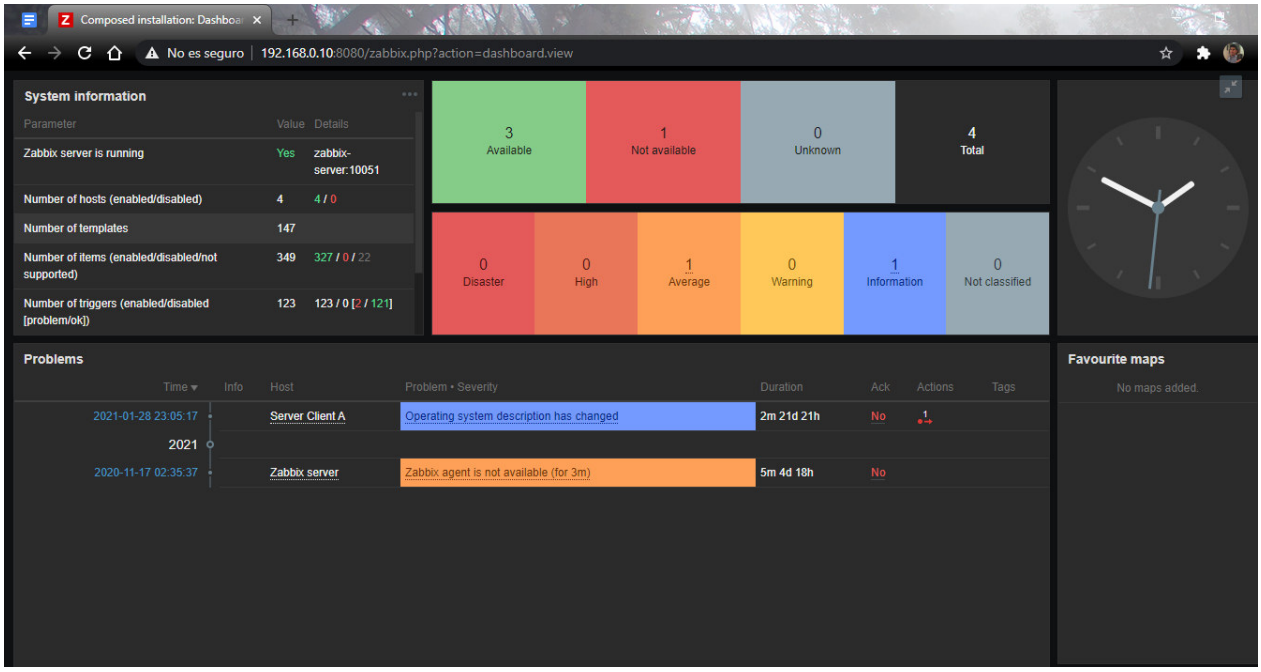


Figura D.35 Interfaz del servidor de prueba Zabbix

D.2. INSTALACIÓN DEL SERVIDOR DE PRUEBA PARA ODOO

1. Para instalar Odoo v14 se utilizó la máquina local, es decir no se creó una máquina virtual para hacer su debida instalación. La instalación de Odoo v14 se la hizo en el sistema operativo Debian 10.

2. Ingresamos al terminal de la máquina y actualizamos los repositorios del sistema.

```
root@odoo:/home/odoo# apt-get update
Obj:1 http://security.debian.org/debian-security buster/updates InRelease
Obj:2 http://deb.debian.org/debian buster InRelease
Obj:3 http://deb.debian.org/debian buster-updates InRelease
Leyendo lista de paquetes... Hecho
```

Figura D.36 Actualización de paquetes del sistema

3. Luego instalamos el servicio PostgreSQL.

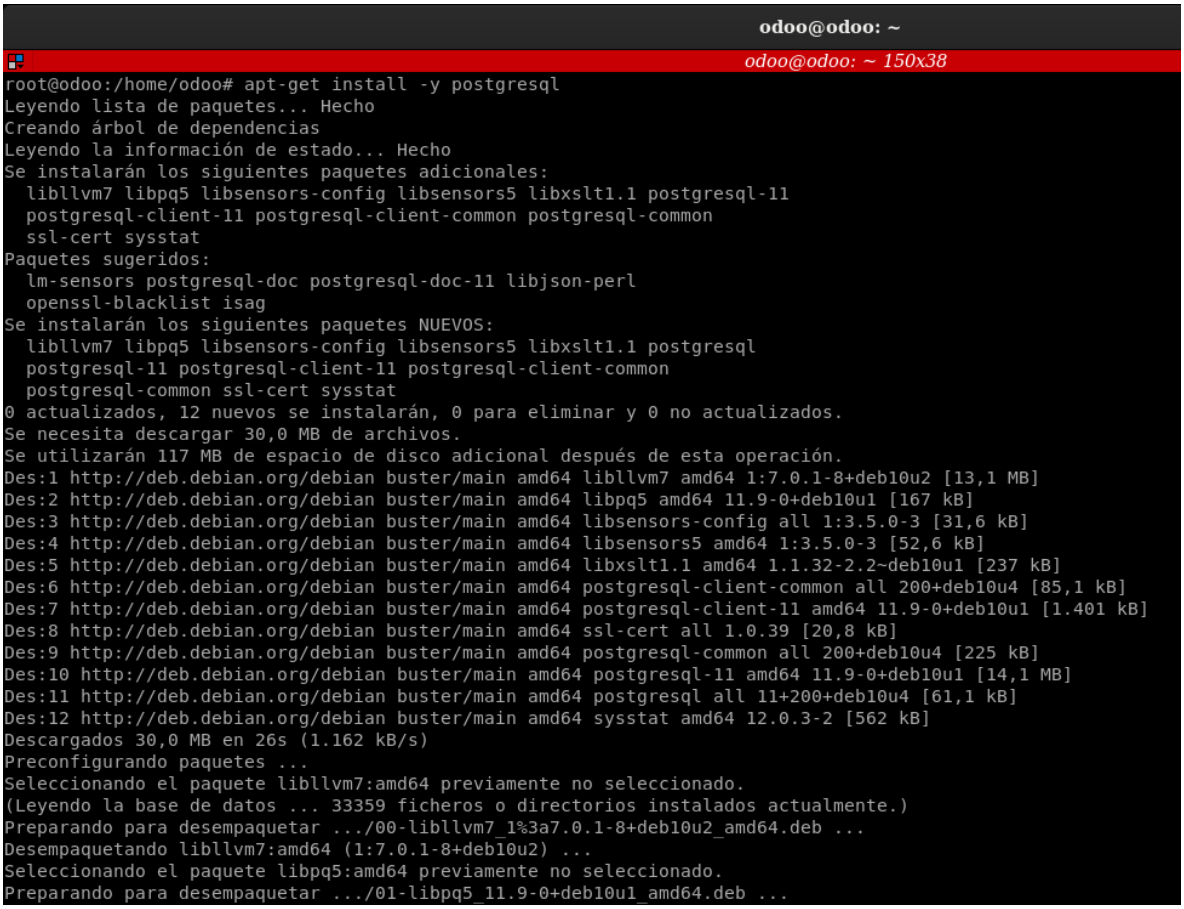


Figura D.37 Instalación de PostgreSQL

4. Crear un usuario de Odoo para PostgreSQL con el comando:

sudo su - postgres -c "createuser -s odoo" 2> / dev / null || cierto

5. Instalar dependencias de Python con el comando:

sudo apt-get install git python3 python3-pip build-essential wget python3-dev python3-venv python3-wheel libxslt-dev libzip-dev libldap2-dev libsasl2-dev python3-setuptools sin nodo libjpeg-dev gdebi -y

6. Instalar dependencias de Python PIP con el comando:

sudo -H pip3 install -r https://raw.githubusercontent.com/odoo/odoo/master/requirements.txt

7. Instalar otros paquetes requeridos con los siguientes comandos:

sudo apt-get install nodejs npm -y

sudo npm install -g rtlcss

8. Instalar Wkhtmltopdf con los siguientes comandos:

sudo apt-get install xfonts-75dpi

sudo wget https://github.com/wkhtmltopdf/packaging/releases/download/0.12.6-1/wkhtmltox_0.12.6-1.bionic_amd64.deb

sudo dpkg -i wkhtmltox_0.12.6-1.bionic_amd64.deb

sudo cp / usr / local / bin / wkhtmltoimage / usr / bin / wkhtmltoimage

sudo cp / usr / local / bin / wkhtmltopdf / usr / bin / wkhtmltopdf

9. Crear directorio de registros.

sudo mkdir / var / log / odoo

sudo chown odoo: odoo / var / log / odoo

10. Instalar Odoo v14.

sudo git clone --depth 1 --branch 14.0 https://www.github.com/odoo/odoo / odoo / odoo-server

11. Establecer permisos en la carpeta de inicio con el siguiente comando:

*sudo chown -R odoo: odoo / odoo / **

12. Crear archivo de configuración del servidor con los siguientes parámetros.

sudo nano /etc/odoo-server.conf

Luego se cambia el permiso y el usuario que tenga acceso al archivo:

sudo chown odoo: odoo /etc/odoo-server.conf

sudo chmod 640 /etc/odoo-server.conf

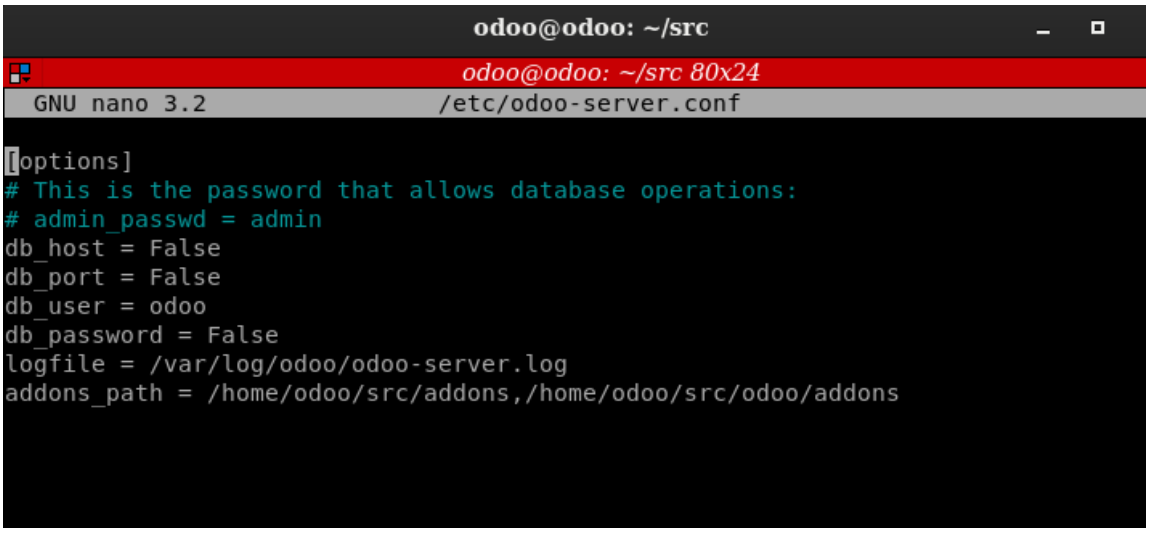


Figura D.38 Configuración del archivo odoo-server.conf

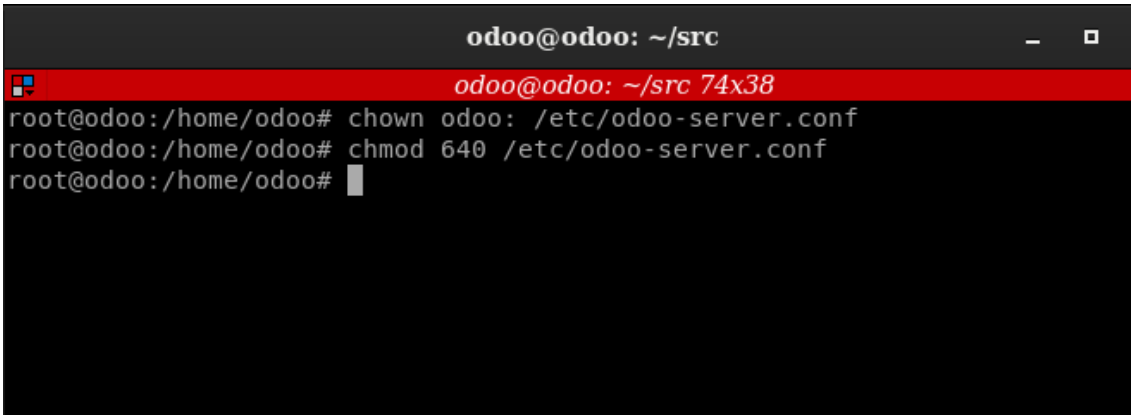


Figura D.39 Cambio de permiso y de usuario al archivo odoo-server.conf

13. Una vez cambiado los permisos se deben iniciar el servicio Odoo, esto se realiza con los siguientes comandos:

sudo su - odoo -s / bin / bash

servidor de cd / odoo / odoo

./odoo-bin -c /etc/odoo-server.conf

14. Ahora la instancia de Odoo está en funcionamiento. Para verificar hay que ir al navegador web y acceder a Odoo en la dirección localhost:8069, si no funciona con la dirección mencionada antes, se debe ingresar la dirección IP que tenga la máquina.

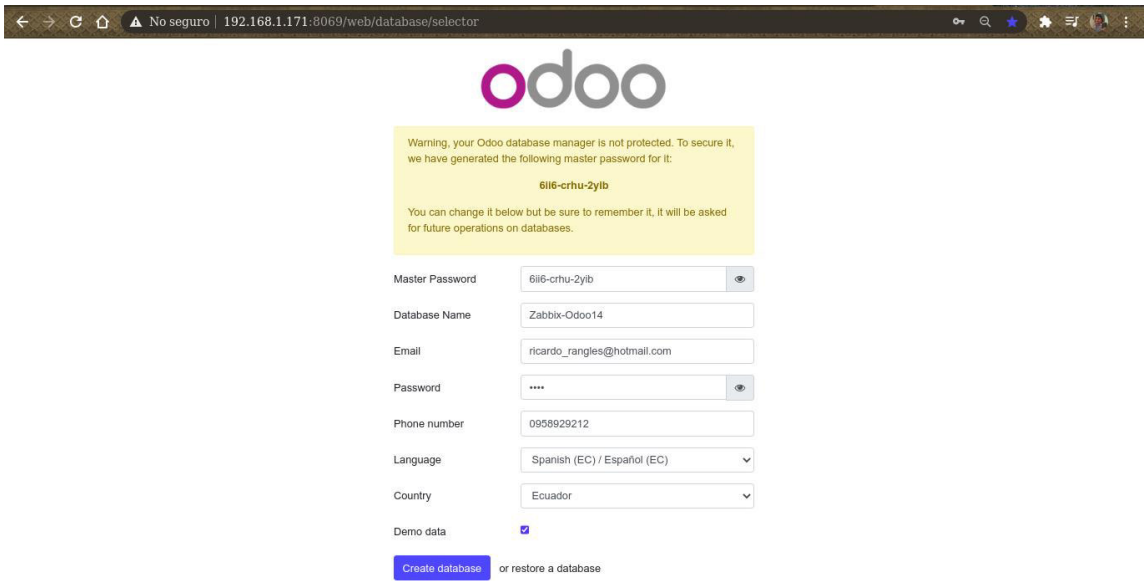


Figura D.40 Instalación final de Odoo v14

D.3. USO DEL MÓDULO DE INTERCONEXIÓN ODOO – ZABBIX

1. Para utilizar el módulo de interconexión Odoo – Zabbix, lo que se debe hacer es copiar la carpeta llamada *zabbix_odoo_interconnection* a su máquina, esta carpeta se encuentra ubicada en: https://github.com/TRESCLOUD/odoo-zabbix-monitoring.git.

2. Una vez copiada la carpeta se debe añadir la ruta de la carpeta copiada al archivo odoo-server.conf, en la sección addons_path.

Figura D.41 Ejemplo de añadir la ruta de una carpeta en addons_path

3. Una vez colocado el path se reinicia el servicio de Odoo que tenga instalado en su máquina.

4. Al reiniciar el servicio Odoo, se debe ir a la interfaz web de Odoo e ir a la sección aplicaciones y buscar el nombre del módulo zabbix_odoo_interconnection para poder instalar el módulo.

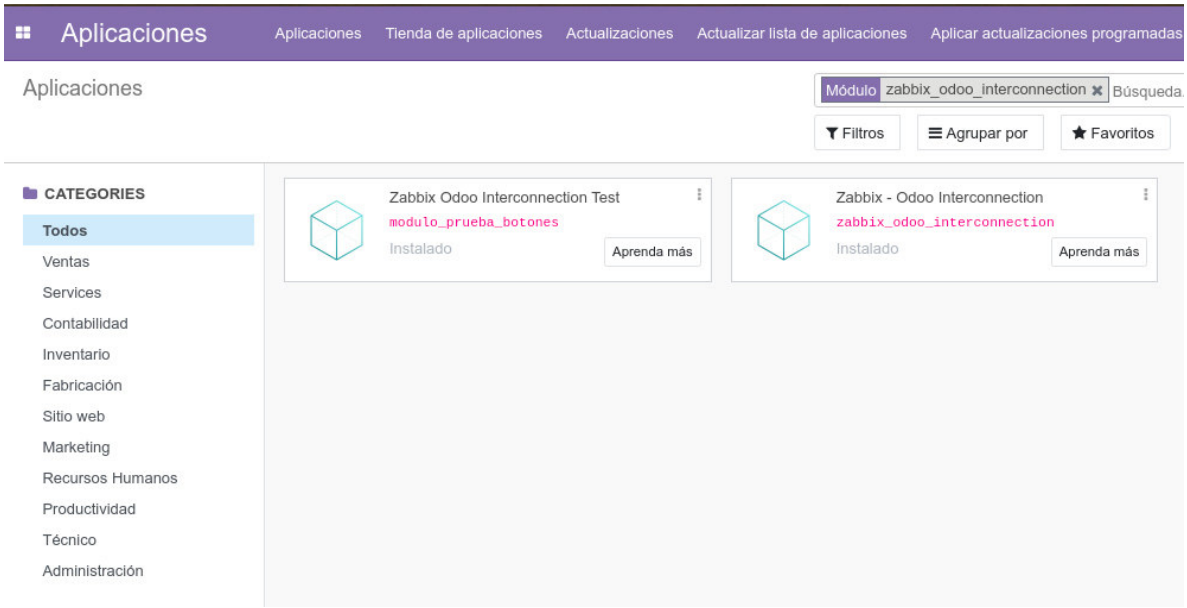


Figura D.42 Instalación del módulo zabbix_odoo_interconnection

5. Finalmente cuando se instale el módulo se lo puede utilizar sin problemas , cabe recalcar que todos los métodos de este módulo deben ser heredables para que se puedan utilizar, además todo el módulo puede ser modificado de acorde a su necesidad.

D.4. CONFIGURACIONES GENERALES EN ZABBIX

A continuación se explica cómo se configuraron todos los parámetros necesarios para realizar el sistema de monitoreo en Zabbix y para el funcionamiento correcto del módulo de interconexión Odoo – Zabbix.

D.4.1 CONFIGURACIÓN DE UN HOST GROUPS

1. Para configurar un host groups en Zabbix primero vamos a la pestaña de *Configuration* y damos clic en *Host groups.*

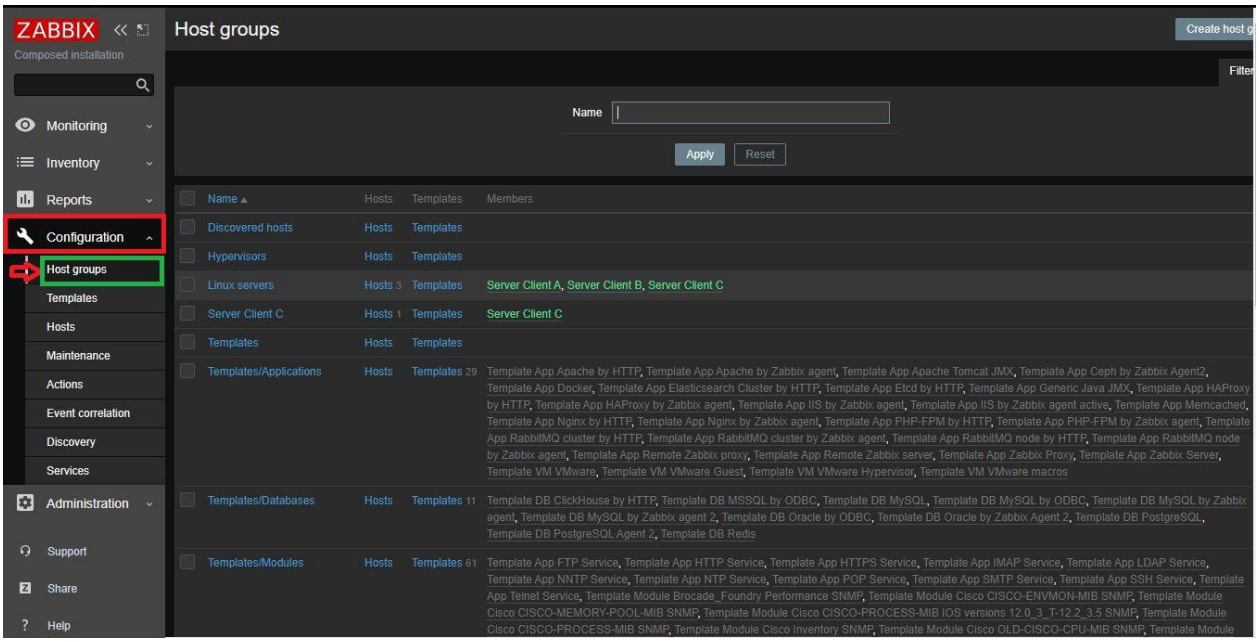


Figura D.43 Pestaña de host groups en Zabbix

2. Una vez en esta pestaña damos clic en *Create host group*.

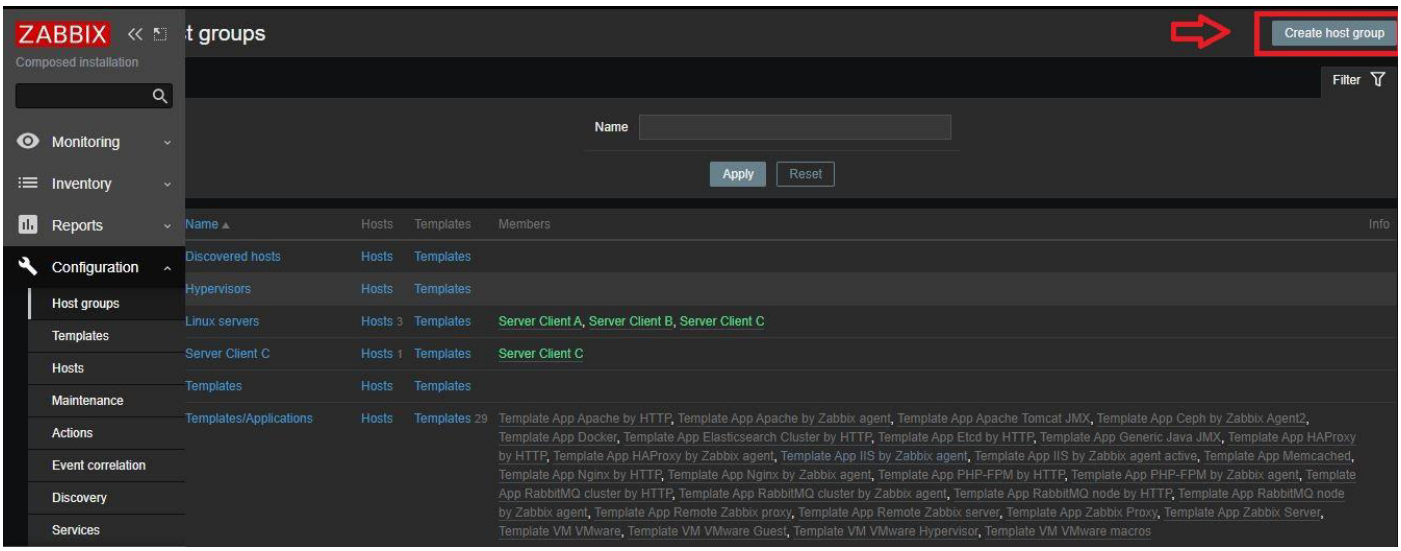


Figura D.44 Botón Create host group

3. Al hacer clic nos pide que ingresemos el nombre del grupo, en este ejemplo pondremos *Server Client C.*

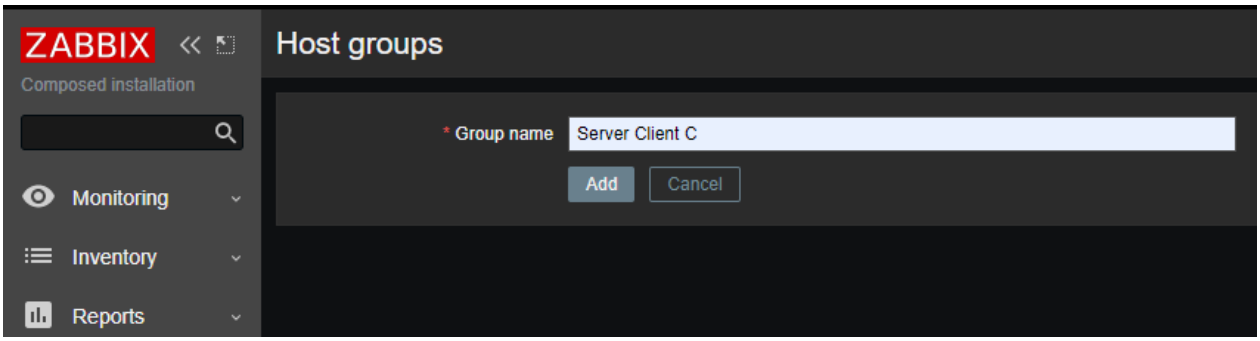


Figura D.45 Creación del nombre del host group Server Client C

4. Por último hacemos clic en *Add* y se añadirá este host group al sistema de monitoreo Zabbix.

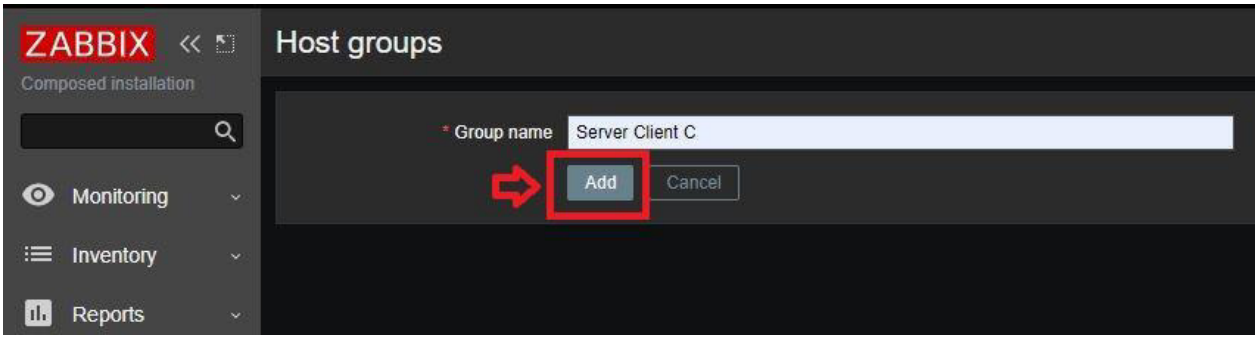


Figura D.46 Creación del host group Server Client C

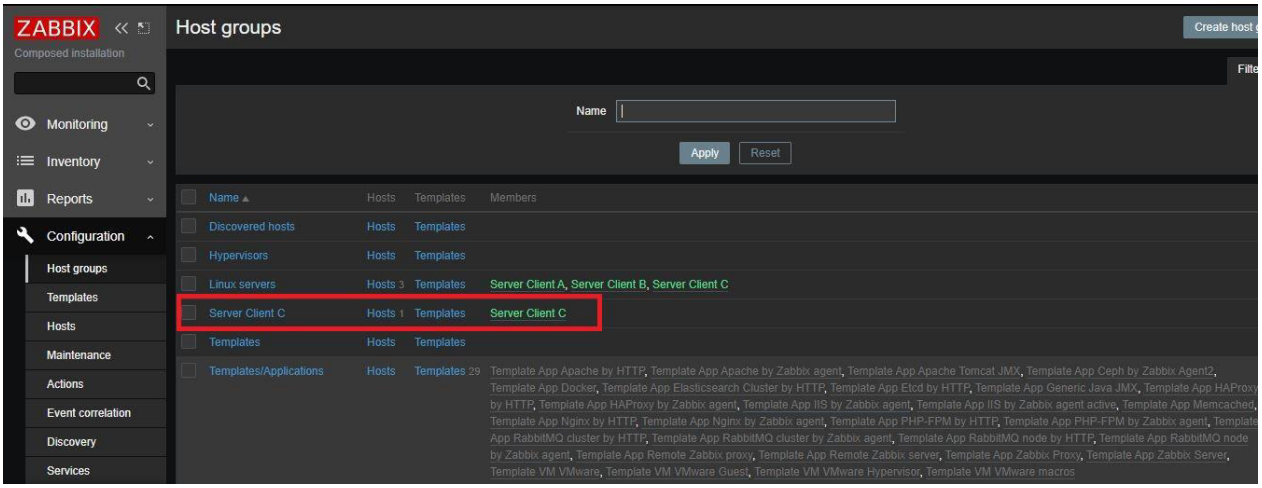


Figura D.47 Verificación de la creación del host group Server Client C

D.4.2 CONFIGURACIÓN DE UN HOST

Para la configuración de un host en Zabbix se va a utilizar el modo pasivo debido a que este es el más utilizado en la empresa TRESCLOUD CIA LTDA.

1. Para configurar un host en modo pasivo primero nos dirigimos al host que va a ser monitoreado y ejecutamos el siguiente comando:

   mkdir $HOME/zabbix_agent

2. Creamos una llave PSK para su cifrado.

   openssl rand -hex 32 > $HOME/zabbix_agent/zabbix_agentd.psk

3. Verificamos la creación de la llave PSK.

cat $HOME/zabbix_agent/zabbix_agentd.psk

Nota: Tener copiado el cifrado PSK para luego ponerlo en la configuración de host en la interfaz de Zabbix.

4. Instalamos el siguiente repositorio:

https://repo.zabbix.com/zabbix/5.2/ubuntu/pool/main/z/zabbix-release/zabbix-release_5.2-1+ubuntu18.04_all.deb

5. Actualizamos los repositorios del sistema.

sudo apt update

6. Descomprimimos la imagen del repositorio anterior.

sudo dpkg -i zabbix-release_5.2-1+ubuntu18.04_all.deb

7. Volvemos a actualizar los repositorios del sistema.

sudo apt update

8. Instalamos el agente de Zabbix.

sudo apt-get install zabbix-agent2

9. Una vez instalado se modifica el archivo zabbix_agentd2.conf

sudo nano /etc/zabbix/zabbix_agent2.conf

10. Al final del archivo zabbix_agentd2.conf se debe añadir lo siguiente y guardar el archivo.

Hostname=riki-testingpasivo

Server=192.168.1.103,127.0.0.1

TLSConnect=psk

TLSAccept=psk

TLSPSKIdentity=PSK 001

TLSPSKFile=/home/ubuntu/zabbix_agent/zabbix_agentd.psk

Nota: En el parámetro *Server* se debe colocar la dirección IP del servidor Zabbix que tenga instalado.

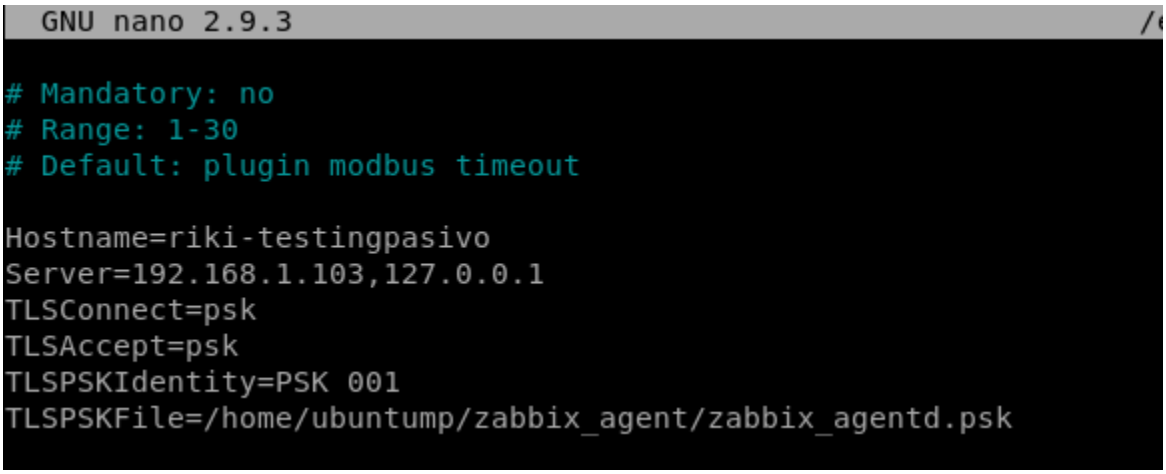


Figura D.48 Modificación del archivo zabbix_agentd2.conf

11. Reiniciamos y volvemos a habilitar el agente de Zabbix.

sudo systemctl restart zabbix-agent2.service

sudo systemctl enable zabbix-agent2

12. Una vez configurado todo en el host que se va a monitorear nos dirigimos a la interfaz del servidor Zabbix y nos vamos a la pestaña *Configuration* y damos clic en Hosts.

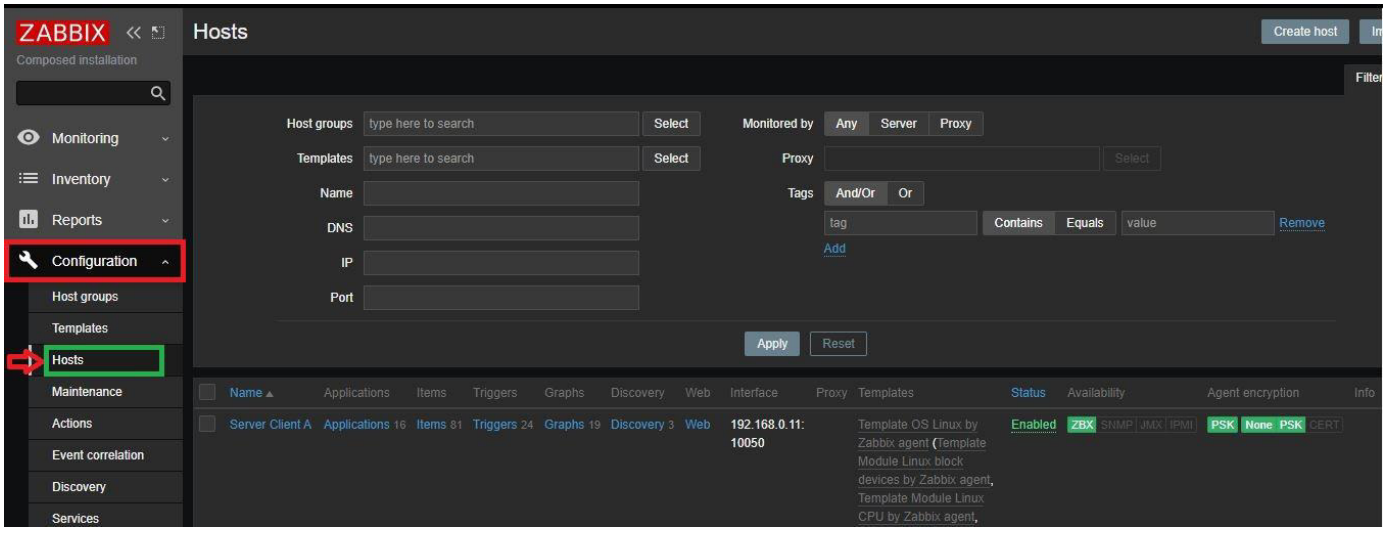


Figura D.49 Pestaña de Hosts

13. Una vez en esta pestaña damos clic en *Create host*

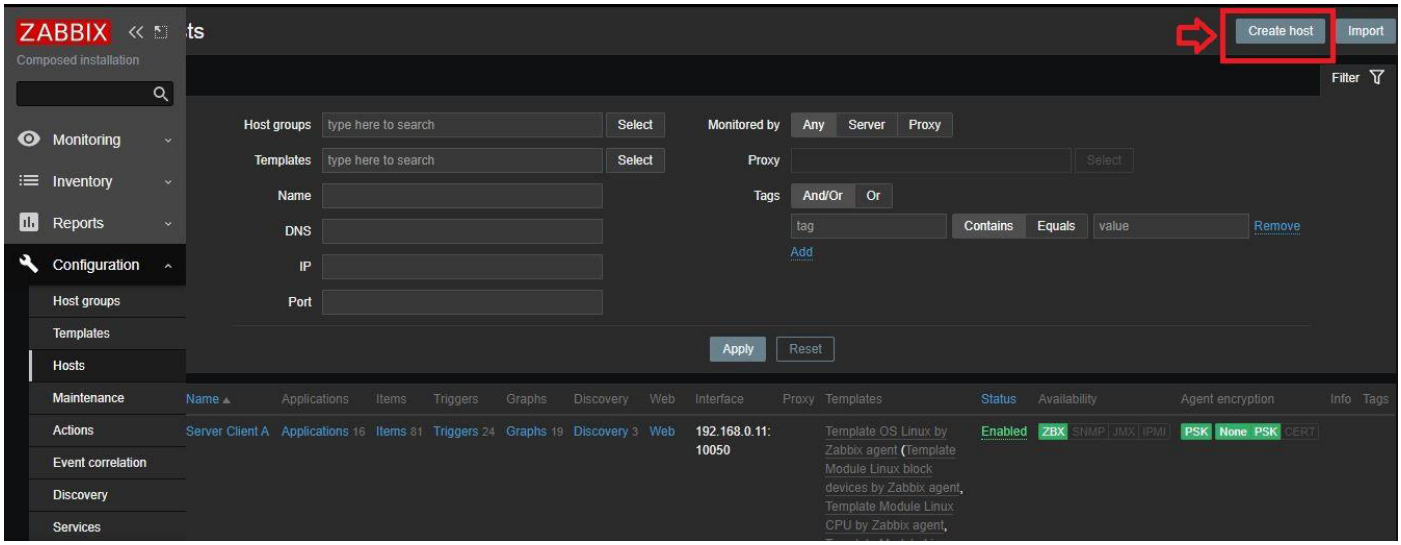


Figura D.50 Botón Create host

14. Colocamos todos los campos que nos pide llenar, cabe recalcar que en el parámetro *Host name* se debe colocar el mismo nombre que se colocó en la configuración del agente en el host, en este caso el nombre es *riki-testingpasivo.* Además, en el parámetro *Intefaces* se debe colocar la dirección IP del host que va a ser monitoreado en este caso la dirección es 192.168.1.108 y mantener el puerto en el 10050.

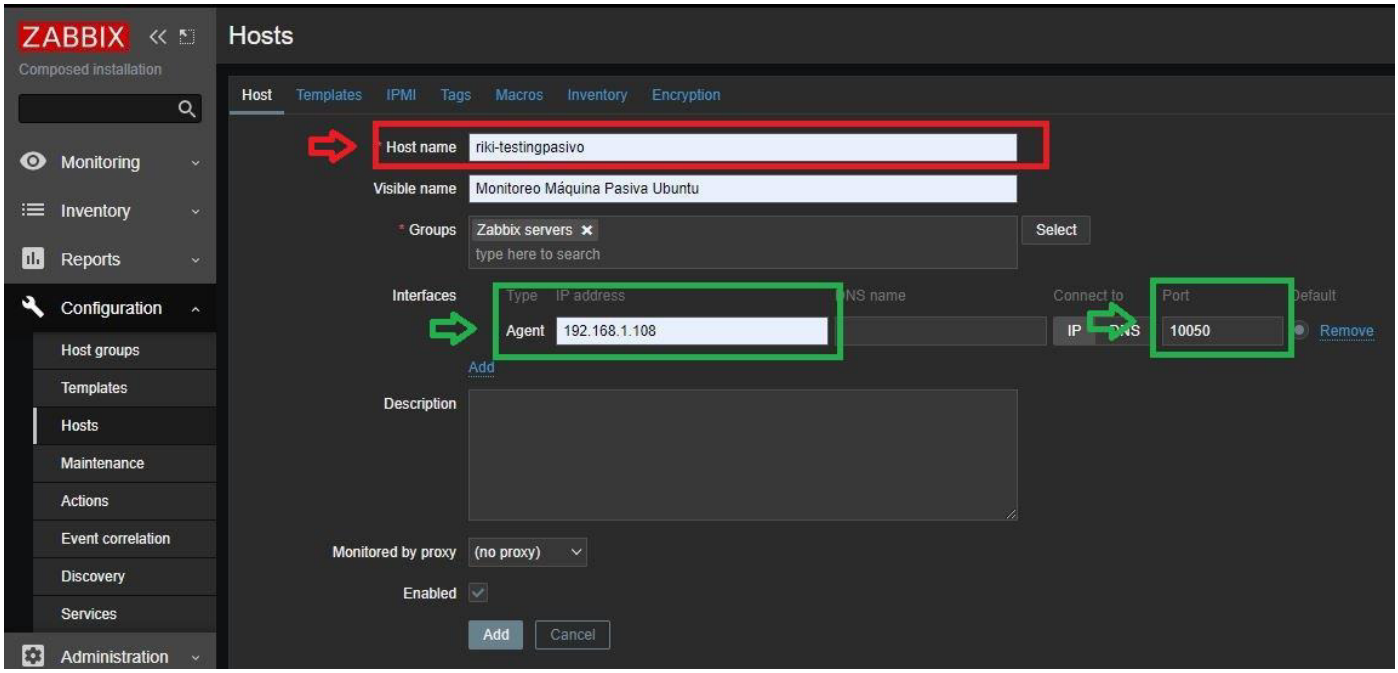


Figura D.51 Creación de un host

15. Ahora damos clic en la pestaña *Templates* y seleccionamos el tipo de agente para el modo pasivo.

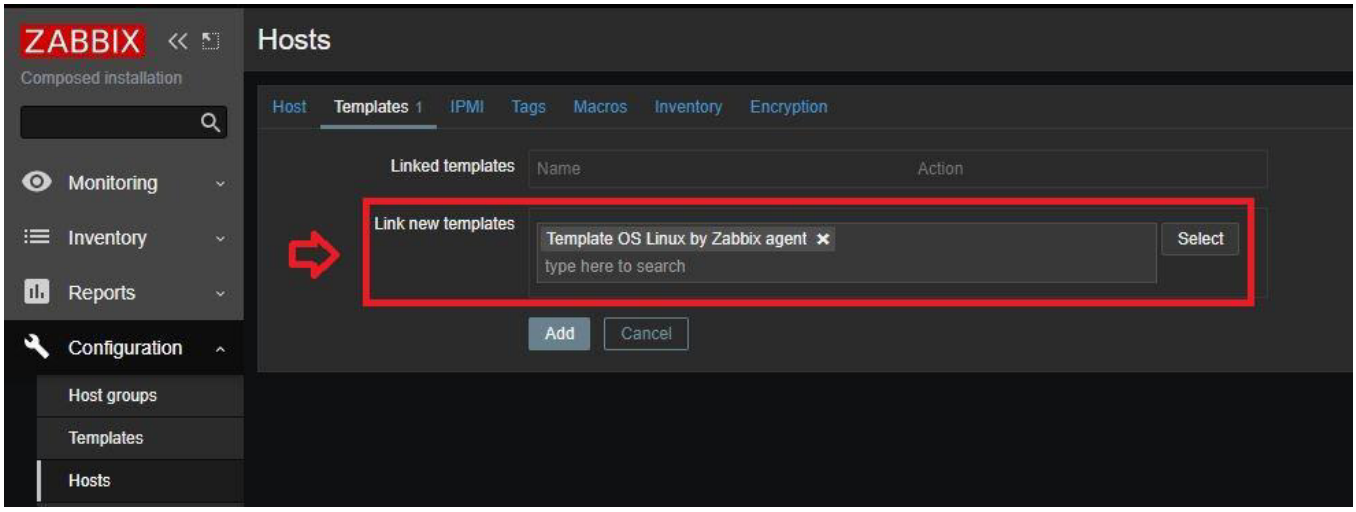


Figura D.52 Configuración del agente para el monitoreo en el host

16. Una vez configurado el *Template* nos hacemos clic en la pestaña *Encryption* donde vamos a configurar el cifrado entre el host y el servidor Zabbix. Una vez allí seleccionamos la opción PSK en *Connections to host,* en la opción Connections from host seleccionamos la opción PSK. Además, en el parámetro PSK identity se debe colocar el nombre que se configuró en el agente, para este caso el nombre es PSK 00. Por último en el parámetro PSK se debe copiar la llave PSK que se copió previamente en el paso 3.

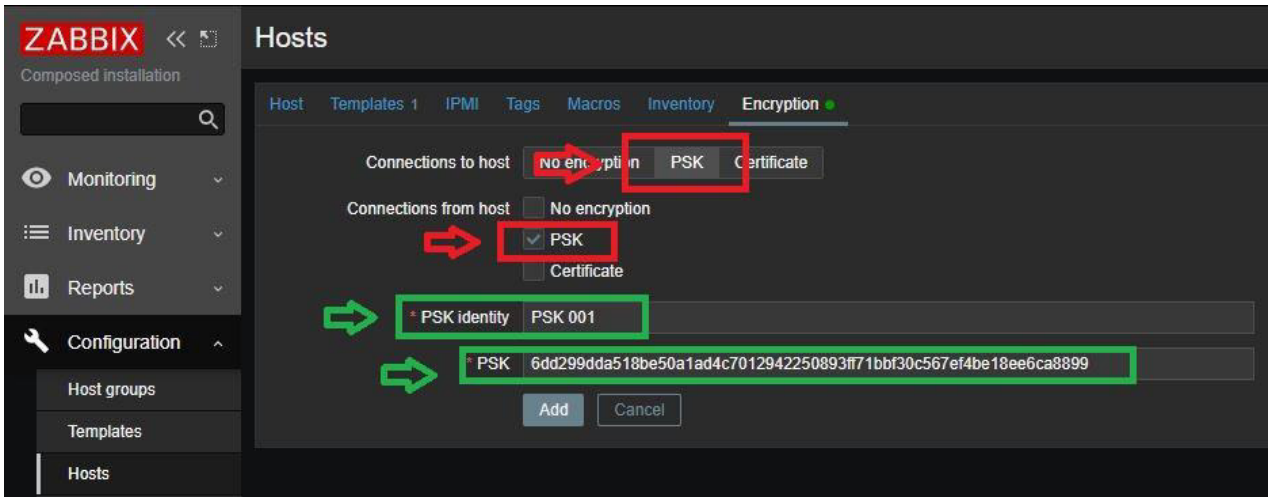


Figura D.53 Configuración del cifrado entre el host y Zabbix

17. Finalmente damos clic en el botón Add.

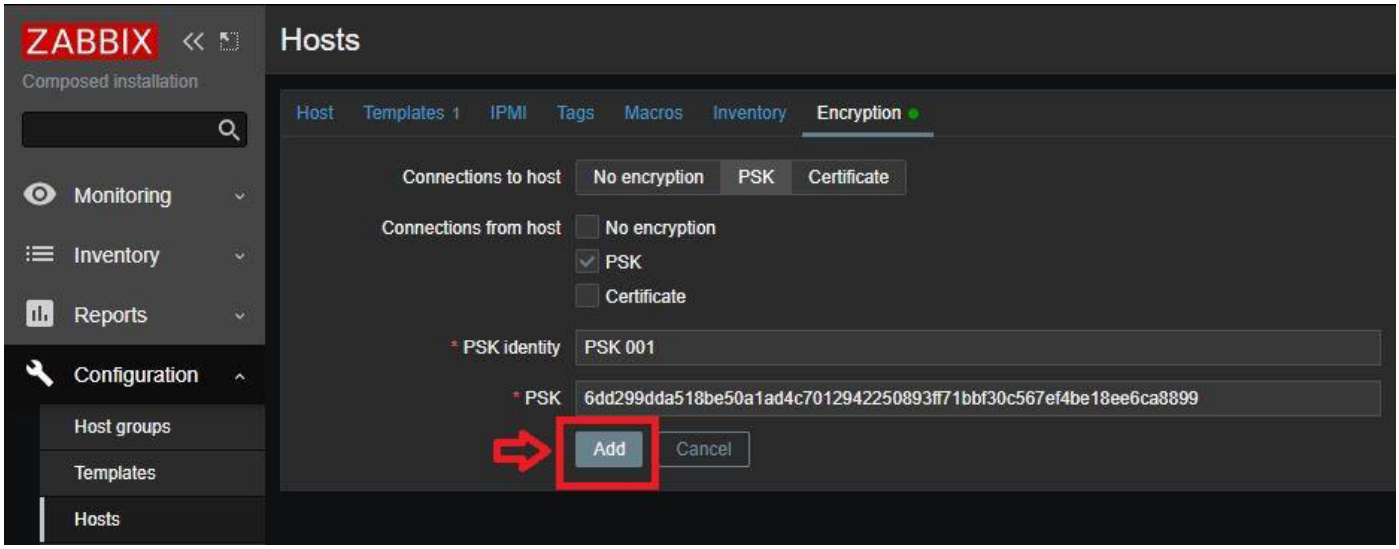


Figura D.54 Finalización de la creación del host

18. Verificamos que se haya creado bien el host.

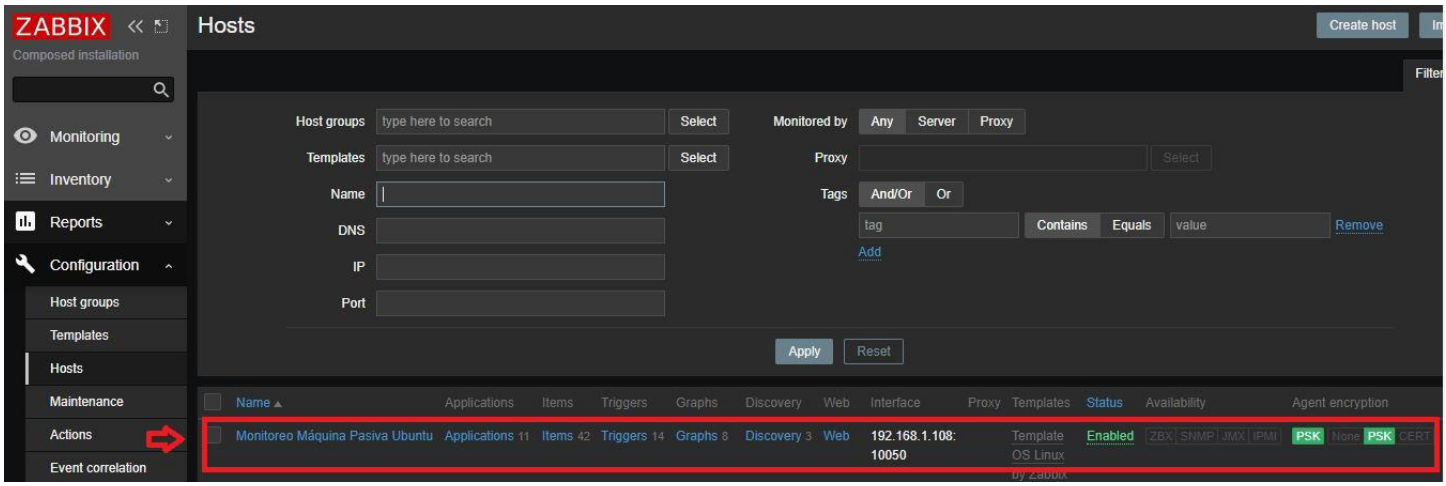


Figura D.55 Verificación de la creación del host

19. Una vez creado el host, este tarda un poco en realizar la comunicación entre los dos, en la Figura D.55 se muestra los estados que se dan cuando ambos están conectados entre sí, esto quiere decir que el host está siendo monitoreado.

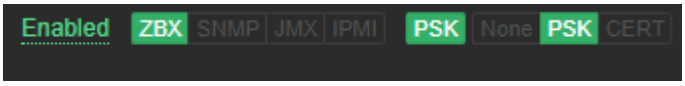


Figura D.56 Estados de conexión cuando existe la comunicación entre host y Zabbix

D.4.3 CONFIGURACIÓN DE UN USER GROUP

1. Para configurar un User group primero vamos a la pestaña de *Administration* y hacemos clic en *User groups.*

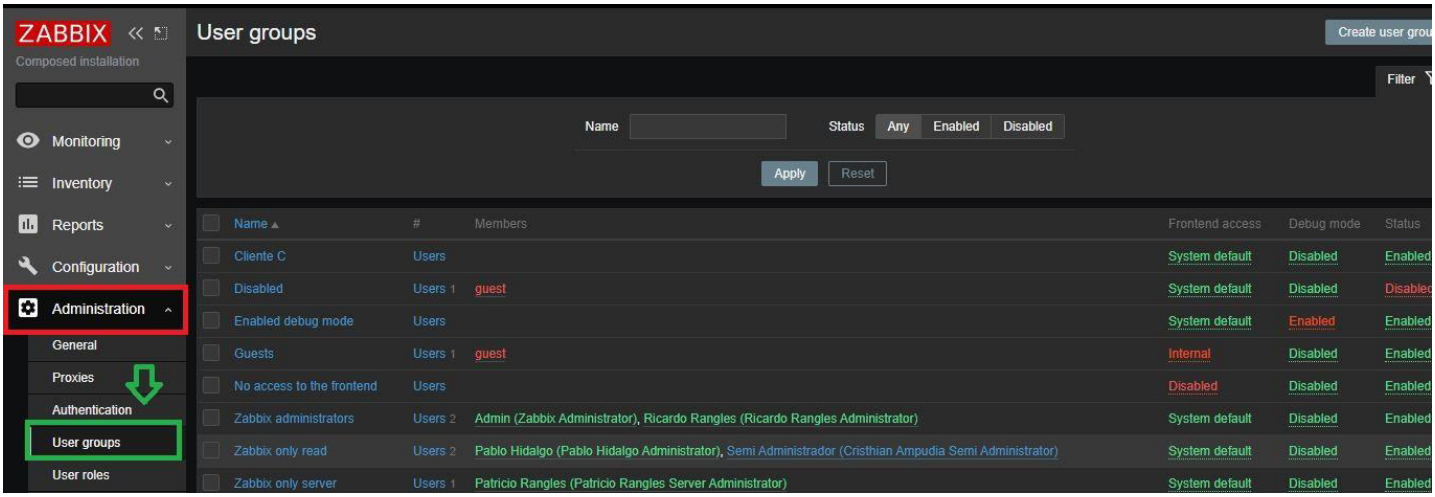


Figura D.57 Pestaña User groups

2. Luego hacemos clic en el botón Create *User group.*

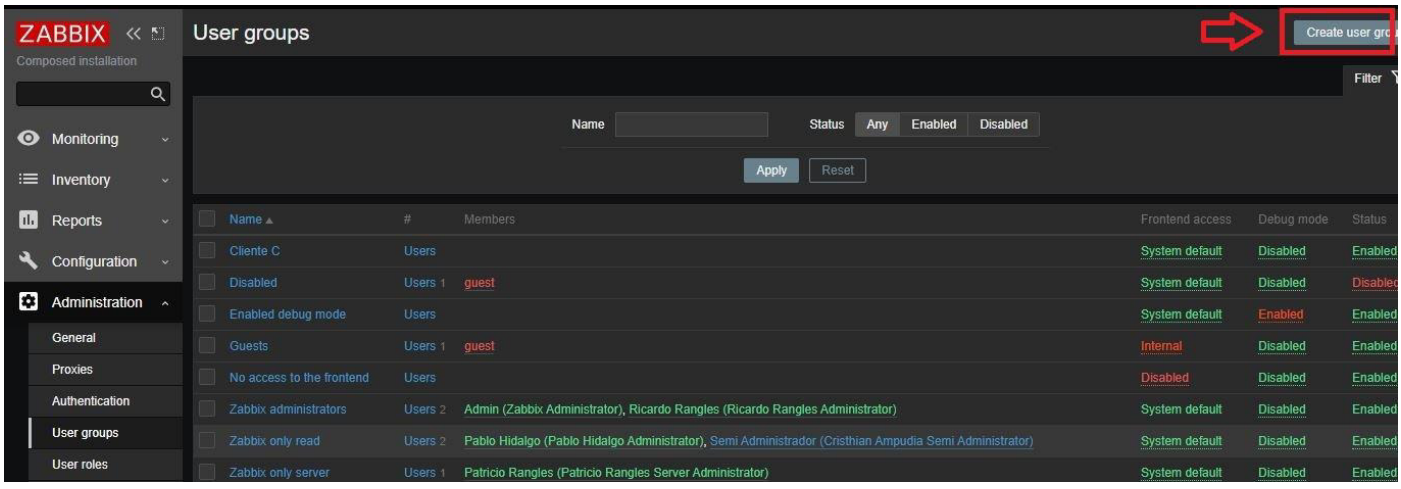


Figura D.58 Botón User groups

3. Colocamos todos los parámetros que pide para su creación, en el parámetro *Users* se debe colocar los usuarios de Zabbix que desee que se encuentren en este tipo de grupo.

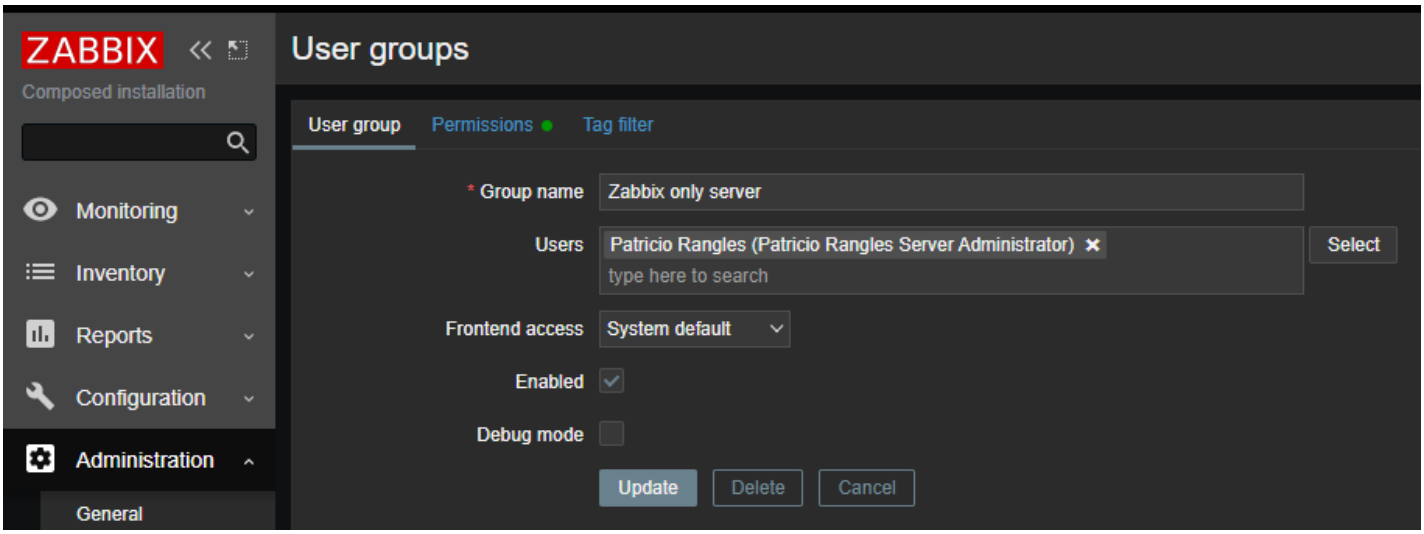


Figura D.59 Ejemplo de creación de un User group

Además, en la pestaña *Permissions* se debe añadir los permisos que se quiera tener en el grupo creado previamente.

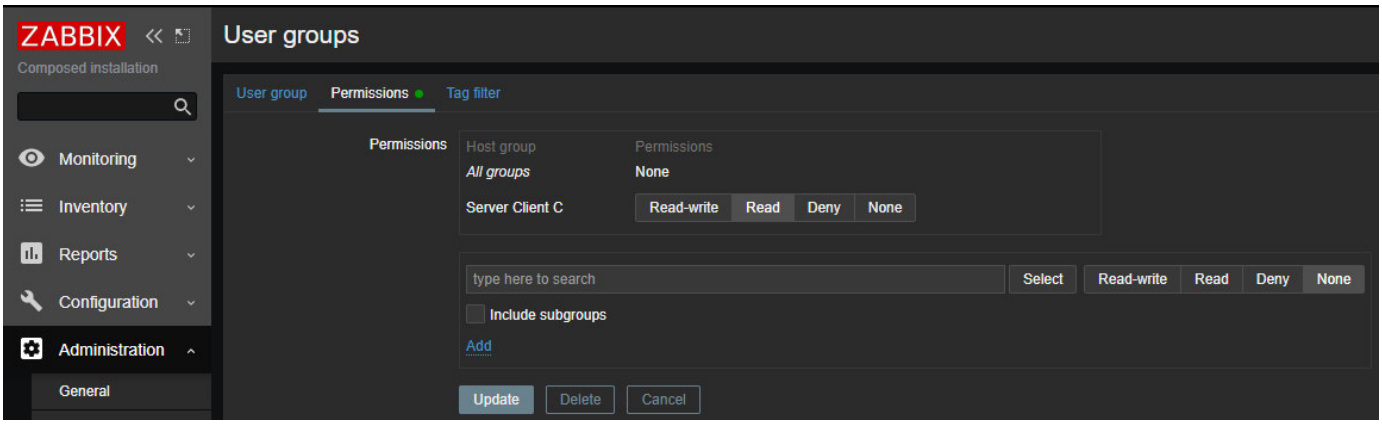


Figura D.60 Ejemplo de creación de permisos para el grupo creado

4. Una vez finalizado hacemos clic en el botón *Add* y listo comprobamos que ya exista en el sistema Zabbix.

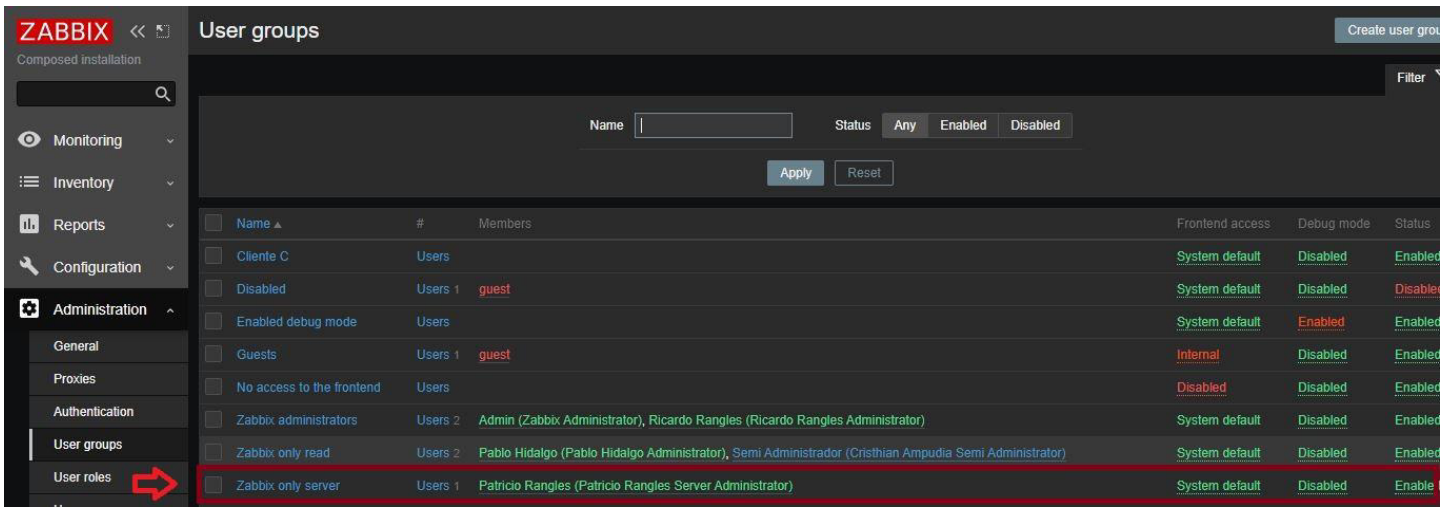


Figura D.61 Verificación de la creación de permisos para el grupo creado

Para configurar Users en Zabbix vamos a crear 2 tipos de Usuarios que tendrán diferentes permisos esto con el fin de demostrar que algunos usuarios pueden realizar ciertas tareas que otras.

1. Primero vamos a crear un usuario que tenga todos los permisos como super administrador, para ello vamos a la pestaña *Administration* y hacemos clic en *User*s.

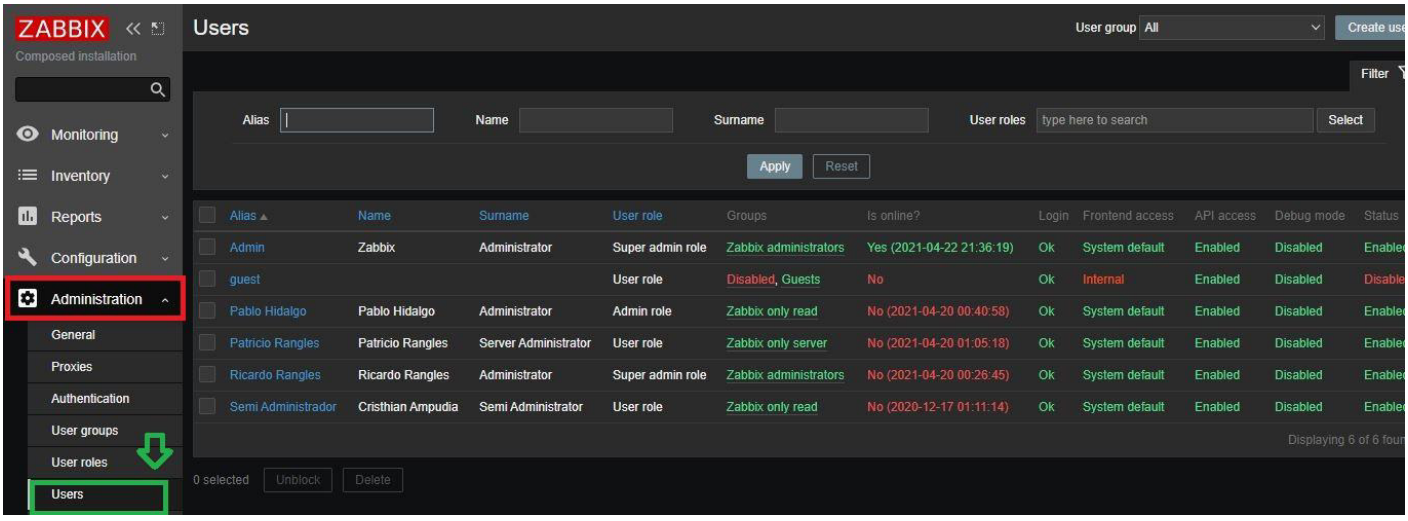


Figura D.62 Pestaña de Users

2. Hacemos clic en el botón Create user.

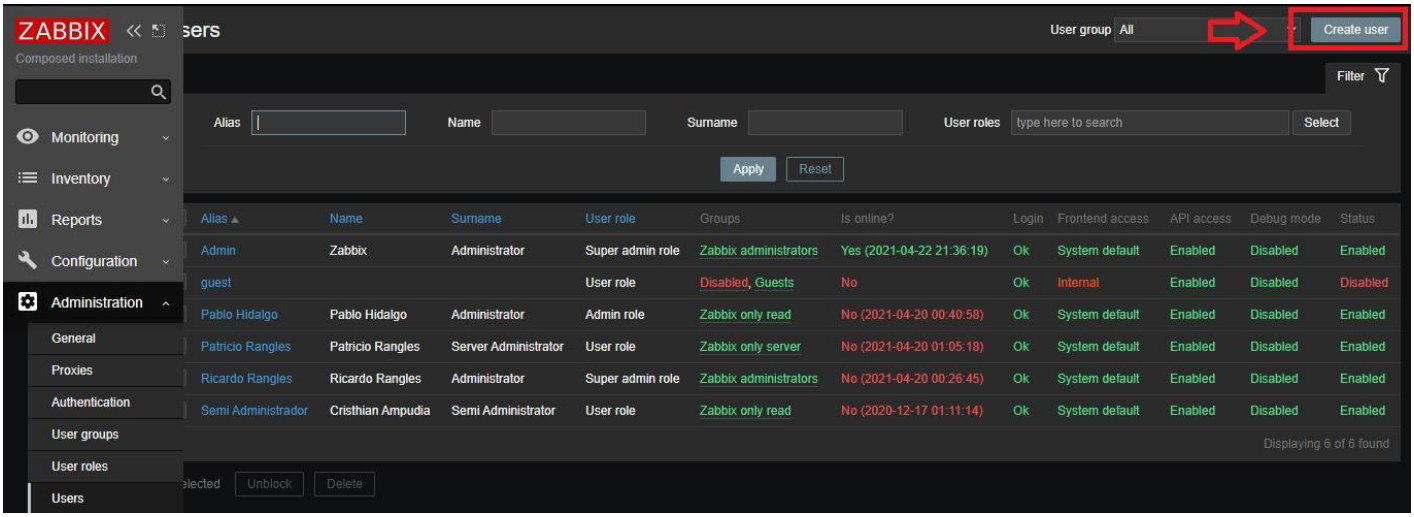


Figura D.63 Botón de Create user

3. Llenar los parámetros que pide Zabbix para crear un usuario.

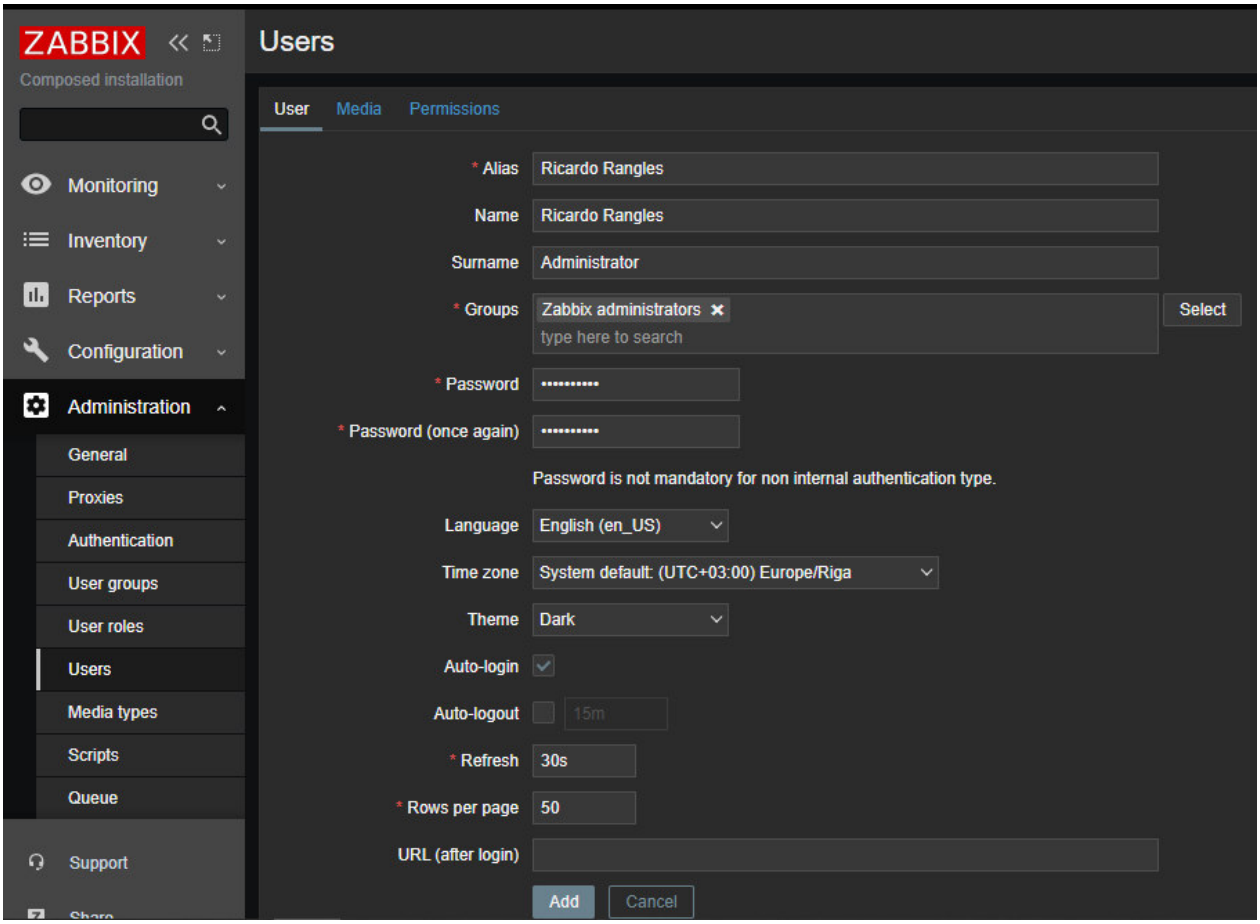


Figura D.64 Creación del usuario Ricardo Rangles

4. Vamos a la pestaña de *Media* y añadimos todos los media types necesarios para notificar a este usuario de cualquier inconveniente con algún servidor o cliente Odoo.

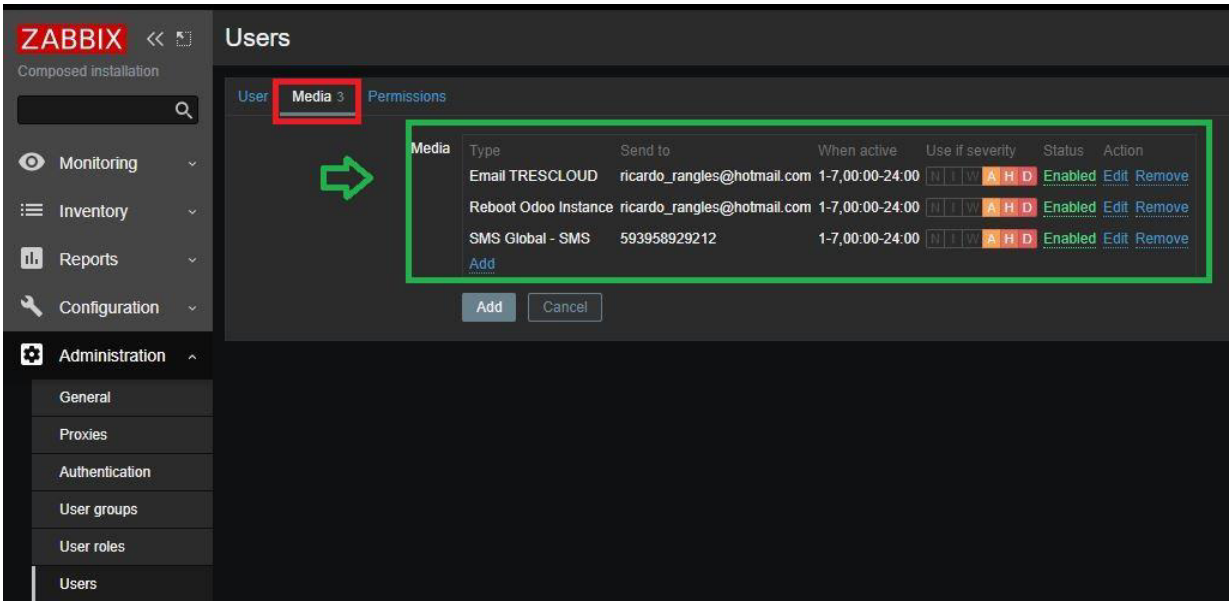


Figura D.65 Configuración de media types para el usuario Ricardo Rangles

5. Vamos a la pestaña de *Permissions* y configuramos el rol de usuario y los permisos que va a tener para utilizar el sistema Zabbix, para este usuario sus permisos serán de super administrador.

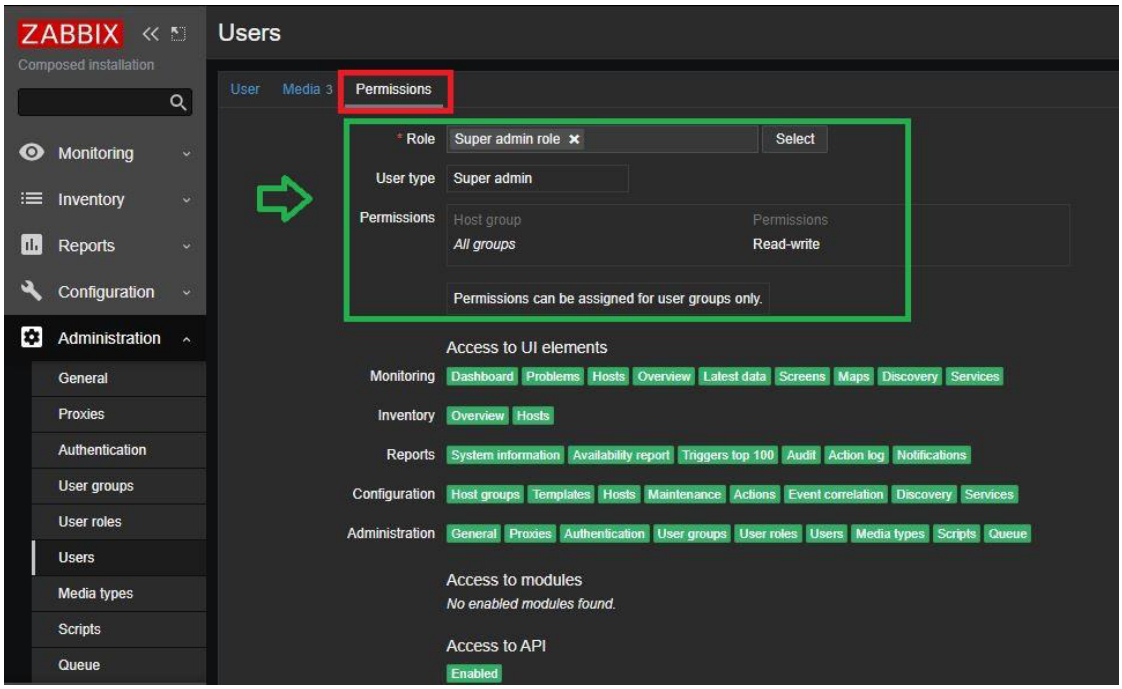


Figura D.66 Configuración de permisos para el usuario Ricardo Rangles

6. Hacemos clic en el botón *Add* y se guarda el usuario.

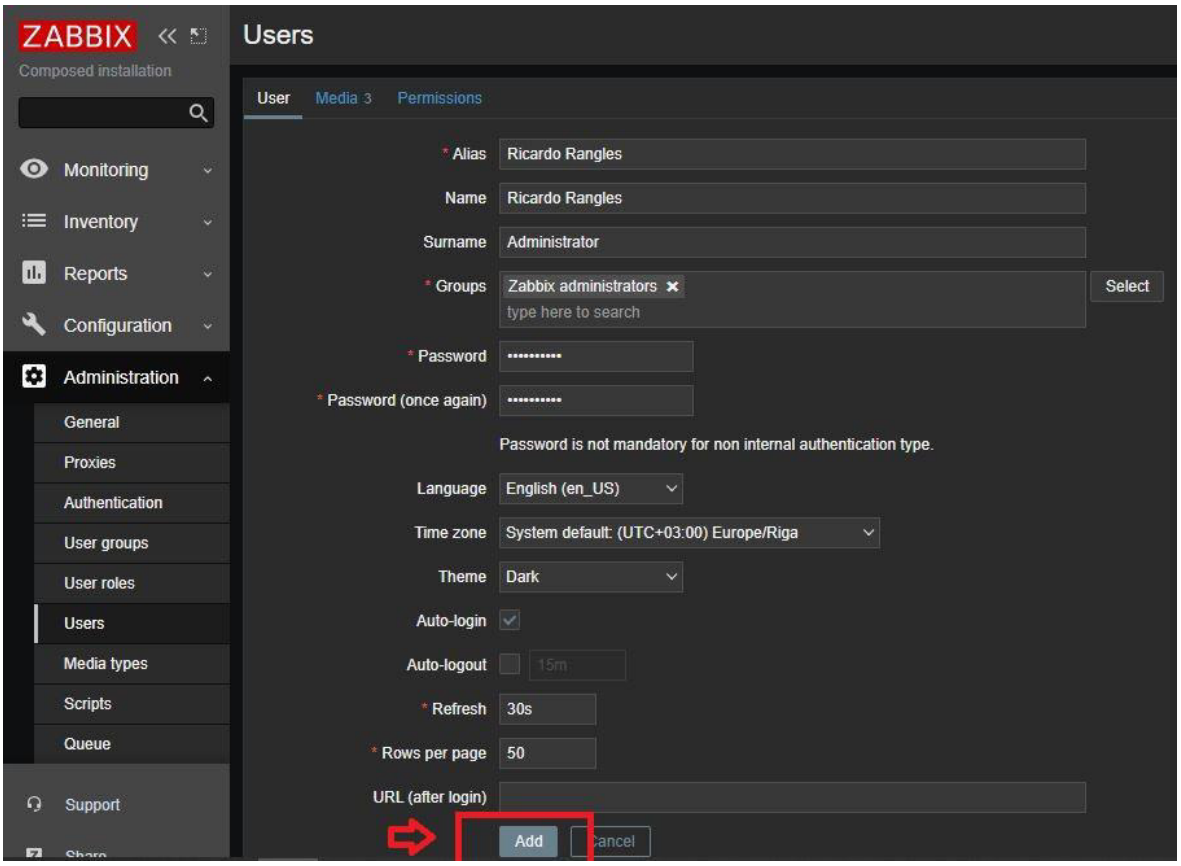


Figura D.67 Finalización de la creación del usuario Ricardo Rangles

7. Verificamos que se haya creado el usuario en el sistema Zabbix.

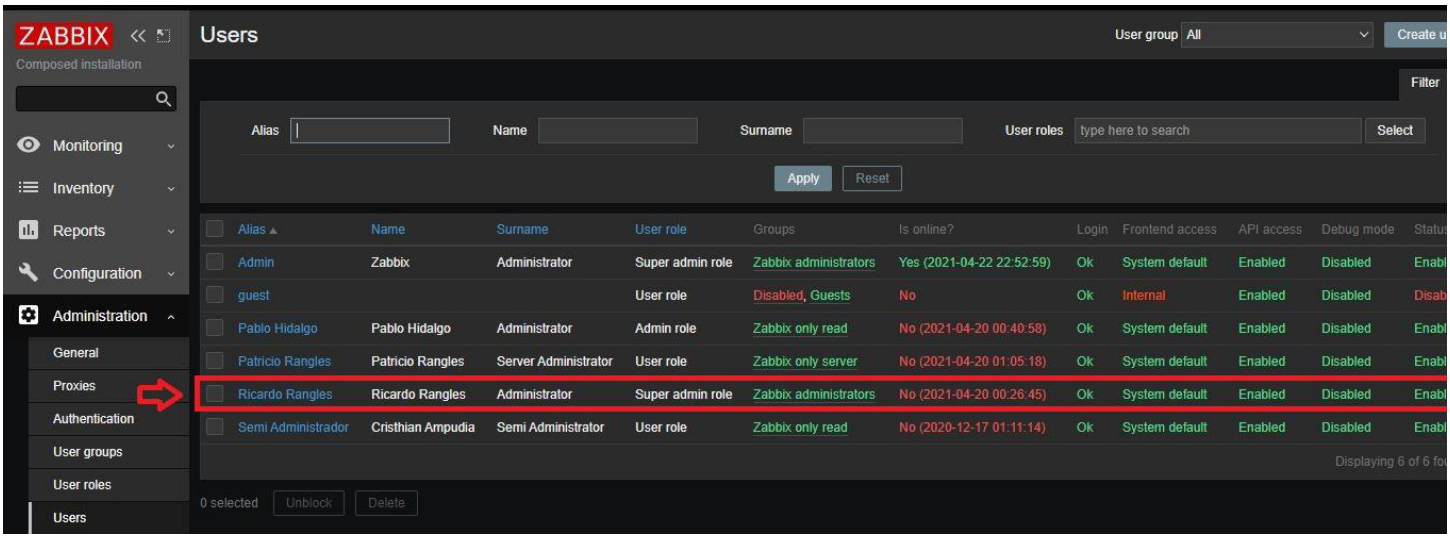


Figura D.68 Verificación de la creación del usuario Ricardo Rangles

8. Ahora vamos a crear un usuario que tenga solo permisos para que pueda ver solo el servidor que tenga por interés. En este caso vamos primero a crear un *Host Group* llamado *Server Client C* como se había explicado antes.

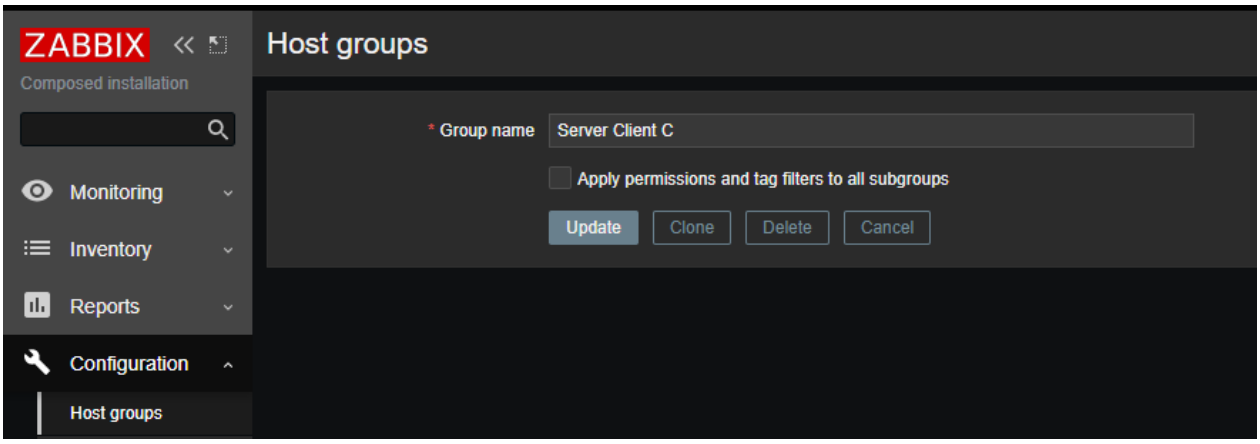


Figura D.69 Creación del host group Server Client C

9. Luego vamos a crear un *User group* llamado *Zabbix only server* como se explicó anteriormente. En el parámetro Users se escoge el usuario que vamos a crear, en este caso el usuario es Patricio Rangles.

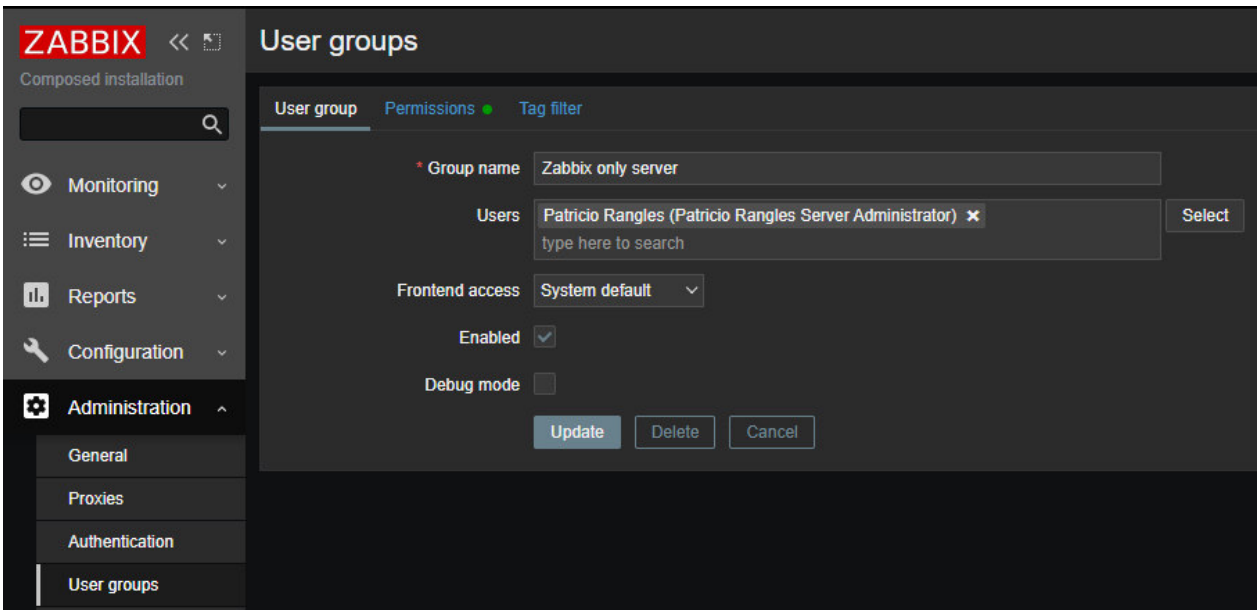


Figura D.70 Creación del user group Zabbix only server

10. Vamos a la pestaña de *Permissions* y seleccionamos el *host group Server Client C* creado antes, esto significa que solo podrá ver este servidor el usuario.

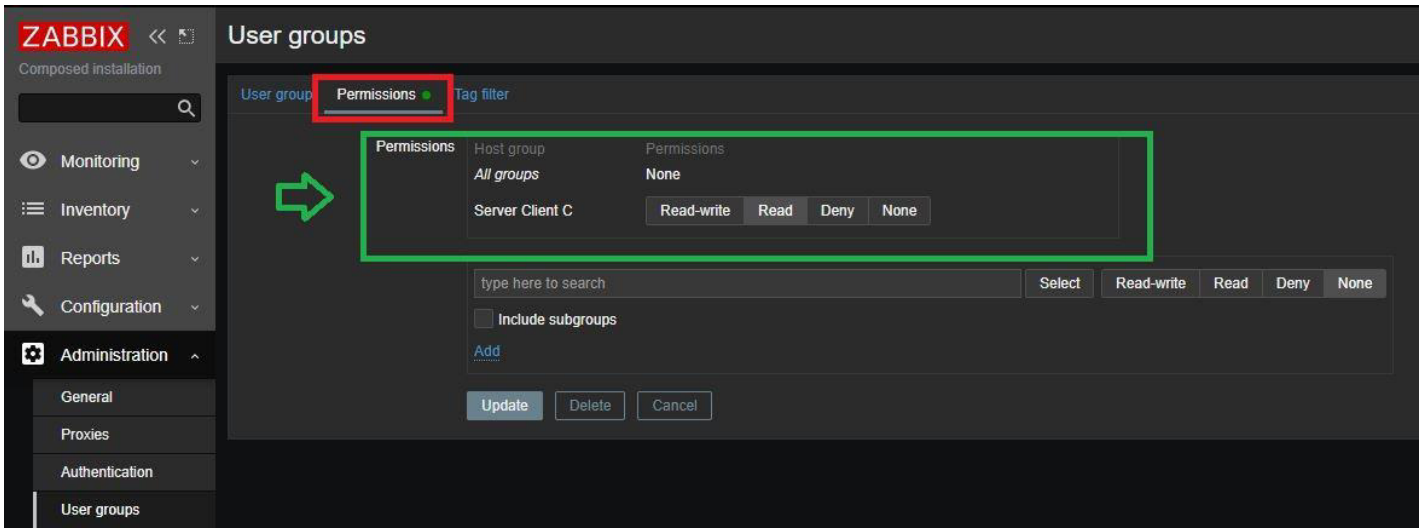


Figura D.71 Configuración de los permisos para el user group Zabbix only server

11. Ahora vamos a crear el usuario Patricio Rangles de la misma manera como se creó el usuario anterior. La única diferencia es que en el parámetro *Groups* se debe colocar el *User Group* creado antes llamado *Zabbix only server.*

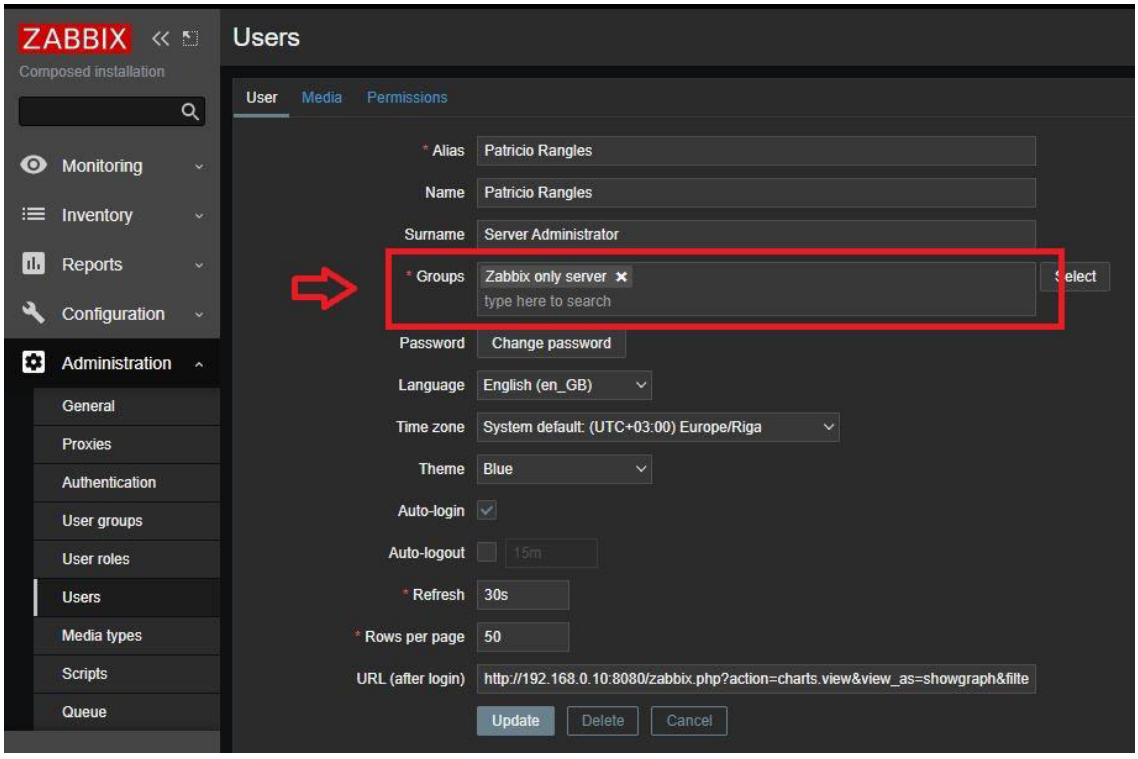


Figura D.72 Creación del usuario Patricio Rangles

12. Ahora vamos a la pestaña *Media* y configuramos todos los *Media types* para que el usuario pueda ser notificado mediante su correo electrónico y su número de teléfono.

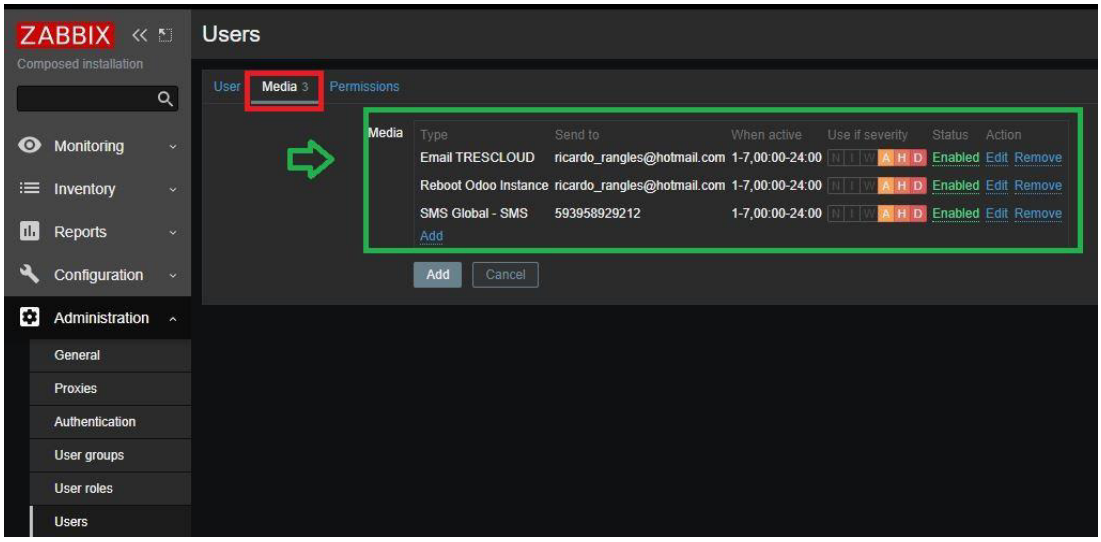


Figura D.73 Configuración de los media types del usuario Patricio Rangles

13. Vamos a la pestaña Permissions y configuramos los permisos necesarios, para este usuario sus permisos serán del tipo usuario.

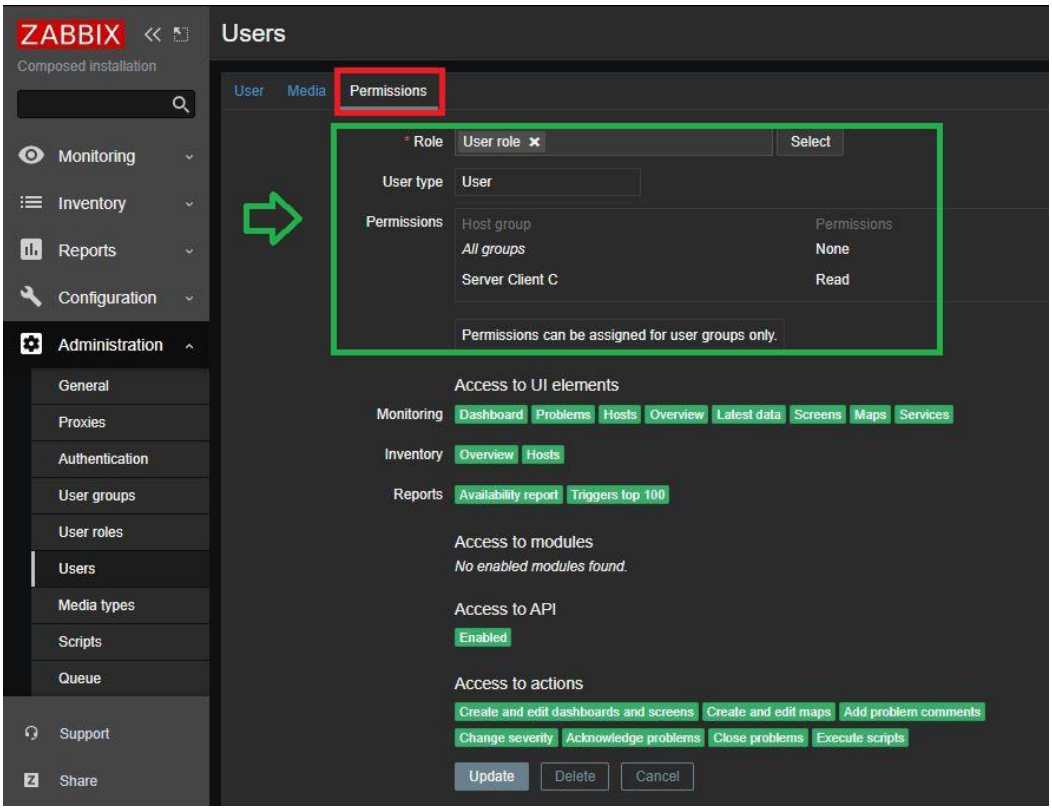


Figura D.74 Configuración de los permisos del usuario Patricio Rangles

14. Por último verificamos que se haya creado el usuario.

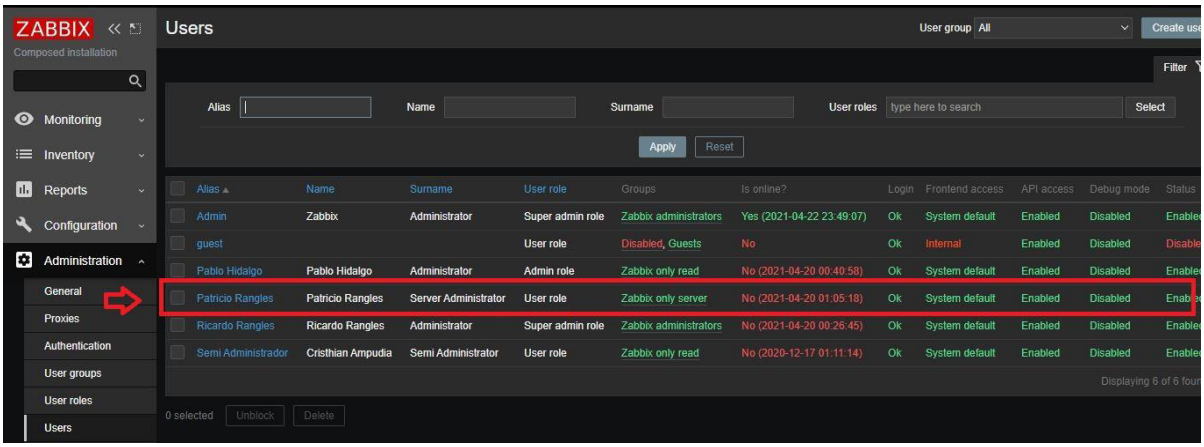


Figura D.75 Verificación de la creación del usuario Patricio Rangles

D.4.4 CONFIGURACIÓN DE MEDIA TYPES

Para el desarrollo del proyecto se crearon 3 media types para avisar al usuario cuando existen problemas tanto con los clientes como los servidores Odoo, A continuación se explica cómo se crearon estos media types para enviar al usuario notificaciones por vía correo electrónico, mensajes SMS y aviso de que un cliente Odoo tuvo problemas.

D.4.4.1 CONFIGURACIÓN DE MEDIA TYPES PARA ENVIO DE NOTIFICACIONES POR CORREO ELECTRÓNICO

Antes de configurar cualquier media types primero debemos tener habilitada la acción "*Report problems to Zabbix administrators"*, esta acción se encuentra en la pestaña de *Configuration* pestaña *Actions.*

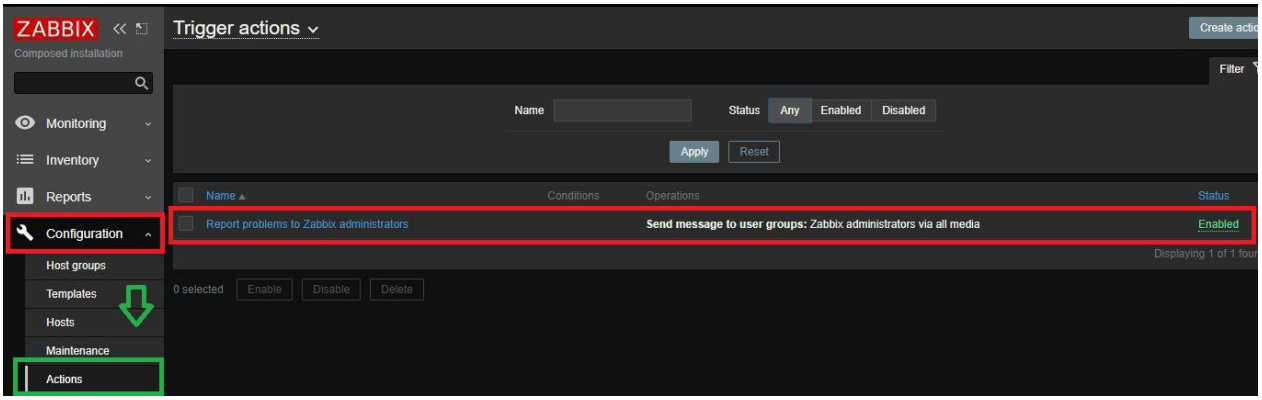


Figura D.76 Verificación de habilitación del Trigger action "*Report problems to Zabbix administrators"*

1. Vamos a la interfaz web de Zabbix y nos dirigimos a la pestaña *Administration* y hacemos clic en la pestaña *Media types.*

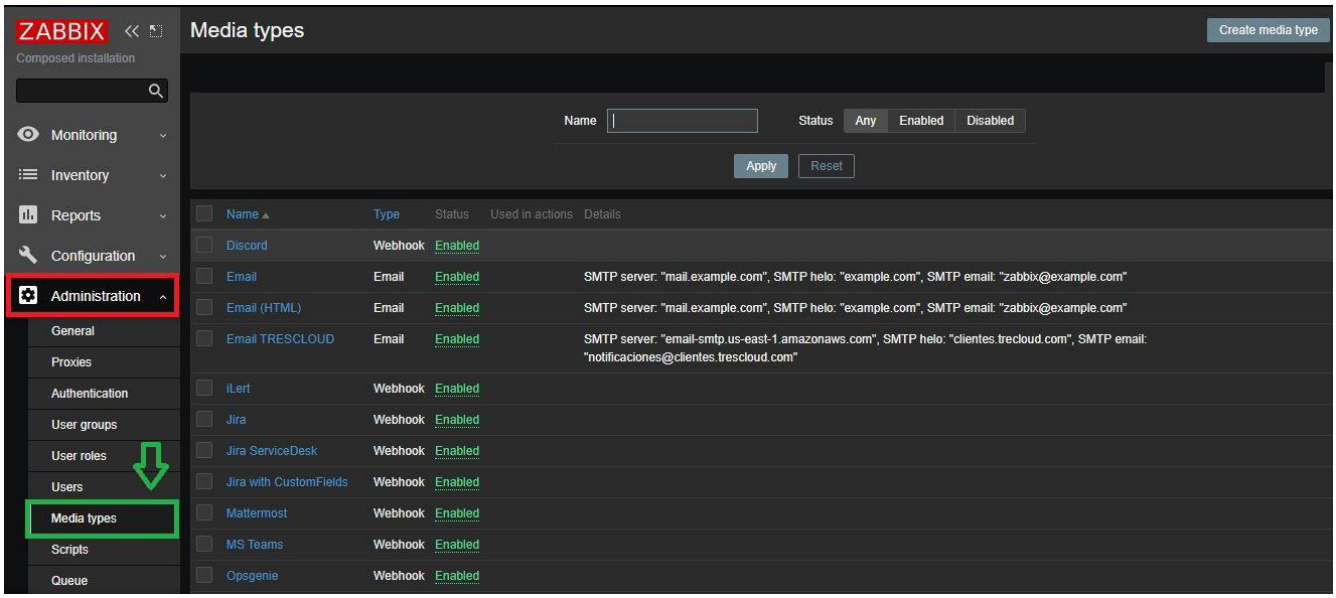


Figura D.77 Pestaña Media types

2. Damos clic en el botón Create media type para crear uno nuevo.

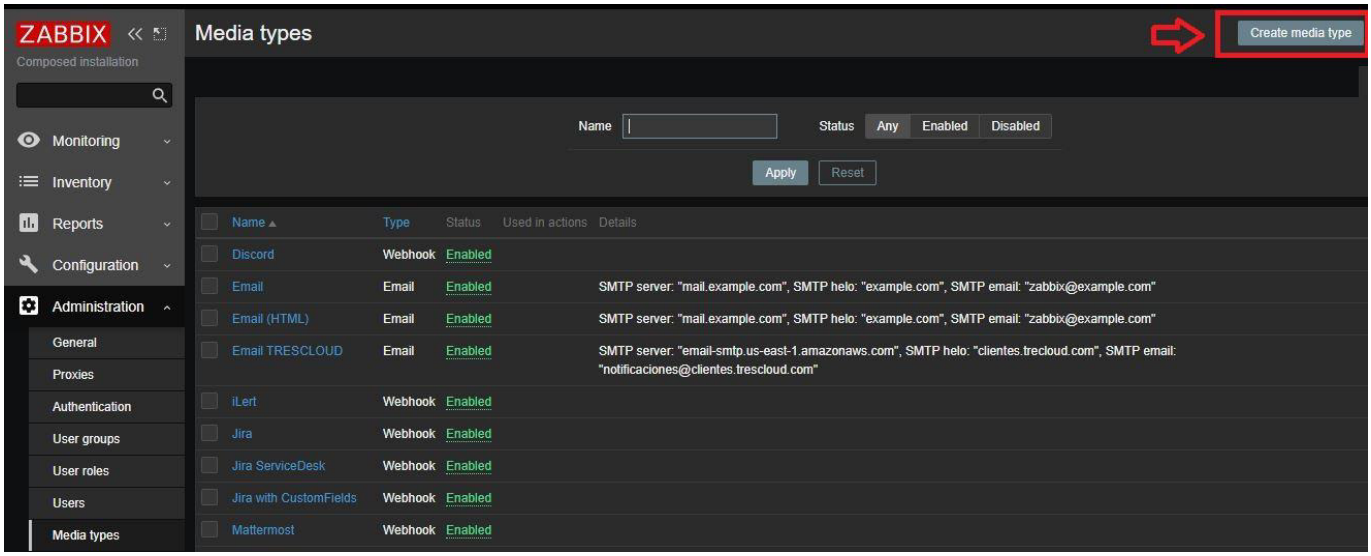


Figura D.78 Botón Create media type

3. Creamos el media type para enviar al usuario las notificaciones por correo electrónico, en este caso se colocaron todos los datos de conexión al servidor de correo de la empresa TRESCLOUD. Por tal motivo, no se puede indicar ya que es de uso privado de la empresa.

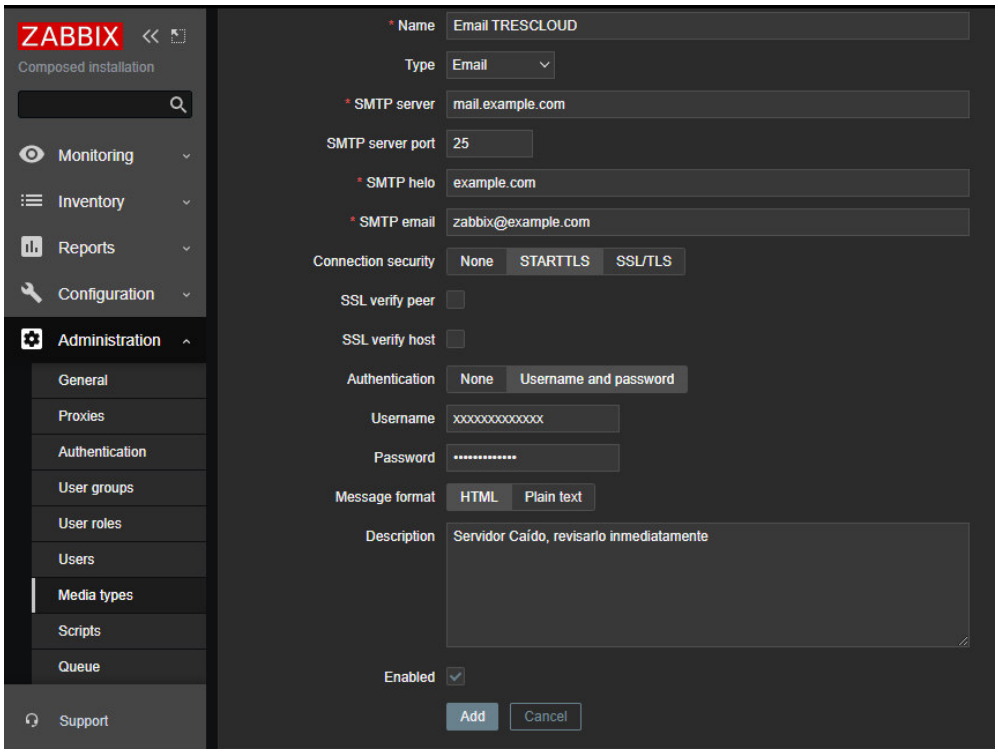


Figura D.79 Creación del media type Email TRESCLOUD

Nota: Los parámetros *SMTP server, SMTP server port, SMTP hello, SMTP email, Username y Password* deben ser llenados con los valores que necesite para conectarse con su servidor de correo electrónico.

4. Vamos a la pestaña de *Message templates* y añadimos las siguientes plantillas para que funcione correctamente el envío de notificaciones por vía correo electrónico.

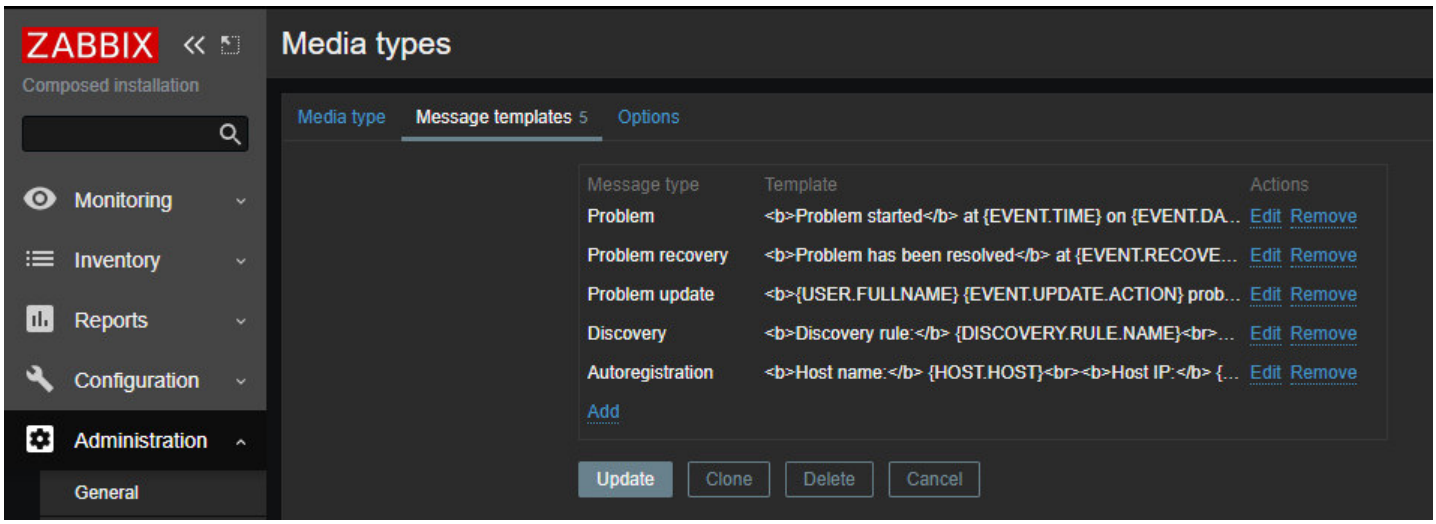


Figura D.80 Configuración del message template del media type Email TRESCLOUD

5. Una vez seleccionados las plantillas que se indican hacemos clic en el botón Add para terminar la creación del media types.

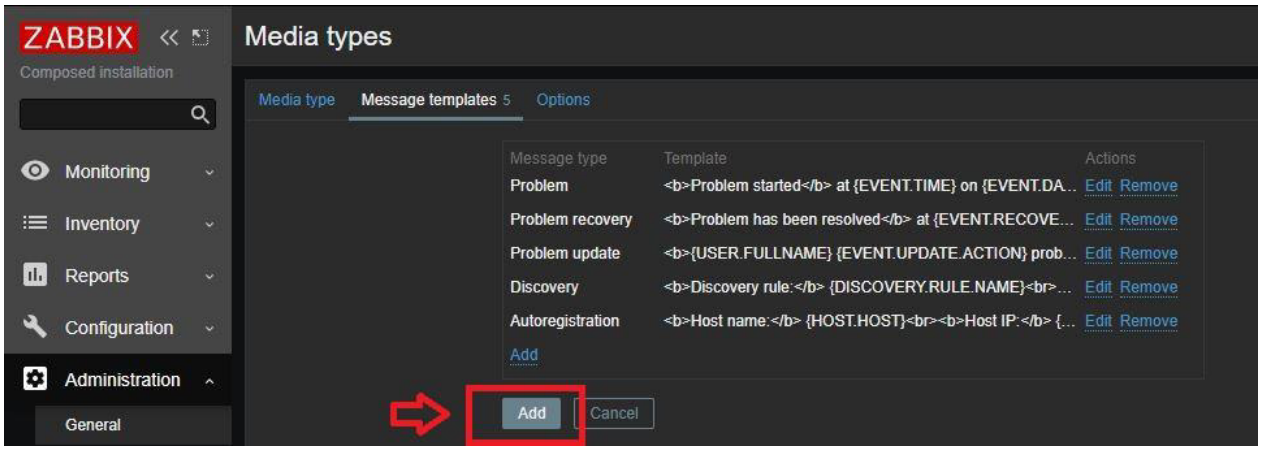


Figura D.81 Finalización de la creación del media type Email TRESCLOUD

6. Verificamos que se haya creado correctamente.

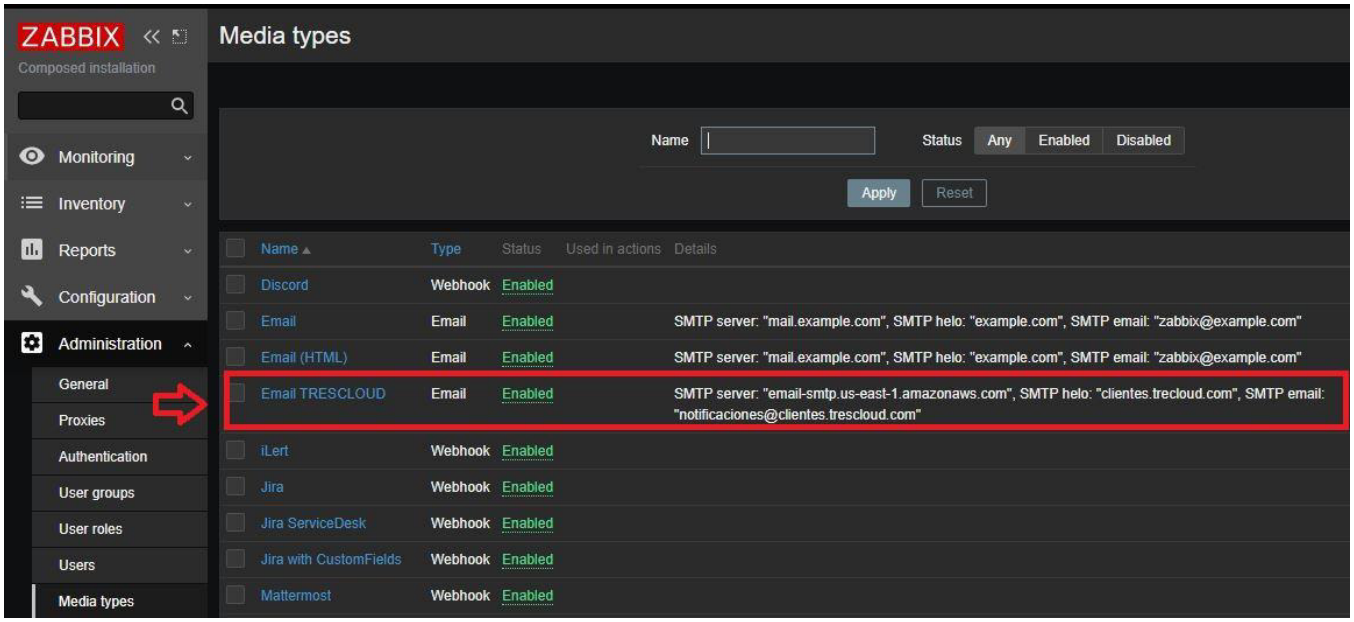


Figura D.82 Verificación de la creación del media type Email TRESCLOUD

7. Para saber si el media type funciona correctamente damos clic en la palabra *Test* del media type Email TRESCLOUD y colocamos los parámetros que nos indica la ventana, por último damos clic en el botón Test y verificamos que funcione correctamente.

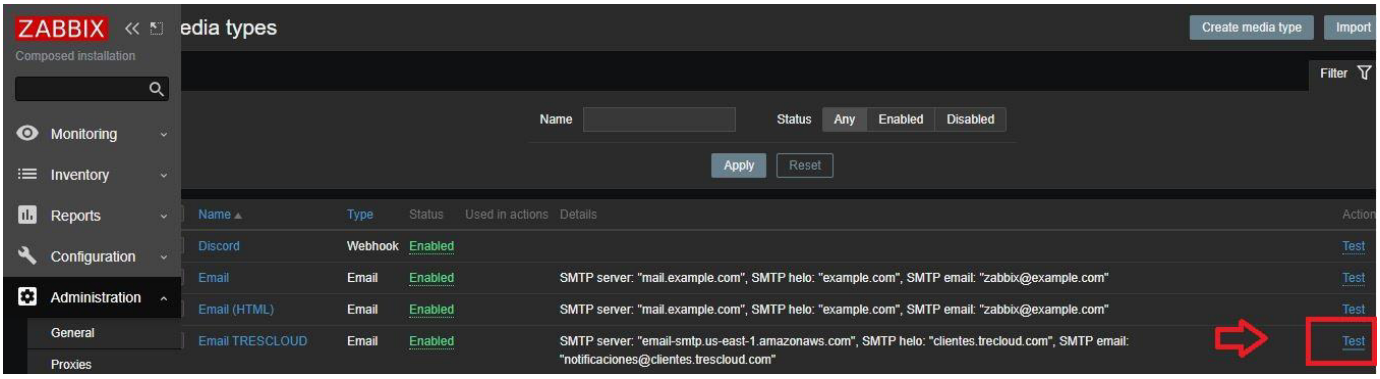


Figura D.83 Verificación del funcionamiento del media type Email TRESCLOUD

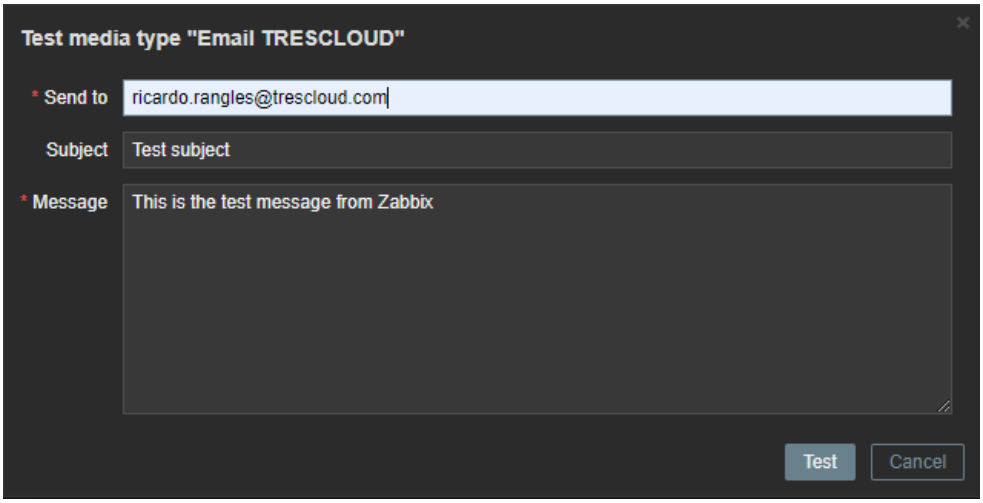


Figura D.84 Verificación del funcionamiento del media type Email TRESCLOUD

Figura D.85 Verificación del funcionamiento del media type Email TRESCLOUD

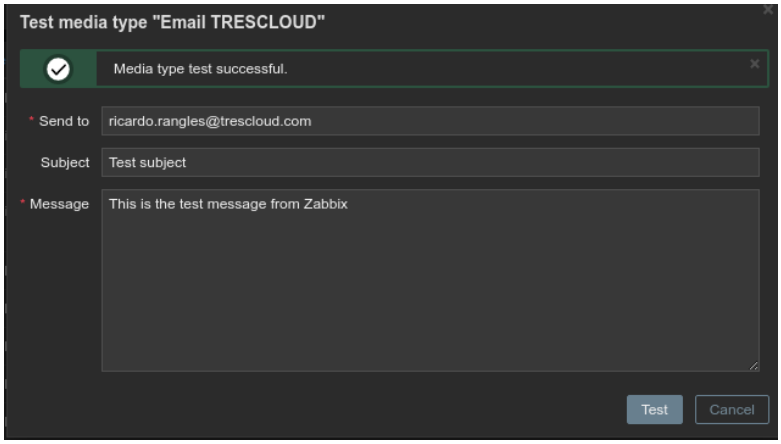


Figura D.86 Verificación satisfactoria del funcionamiento del media type Email TRESCLOUD

D.4.4.2 CONFIGURACIÓN DE MEDIA TYPES PARA ENVIO DE NOTIFICACIONES POR MENSAJES SMS

1. Vamos al servidor de Zabbix y copiamos el script *send_sms.sh* que se menciona en el Anexo D sección D.3 para que configurarlo dentro de la interfaz web de Zabbix. Este script debe ser copiado en la siguiente dirección:

   cd zabbix docker/zbx_env/usr/lib/zabbix/alertscripts

2. Una vez copiado el script dentro del servidor Zabbix, vamos a la interfaz web del servidor y creamos un nuevo media type como se creó antes, para este nuevo media type su nombre será "*SMS Global – SMS*".

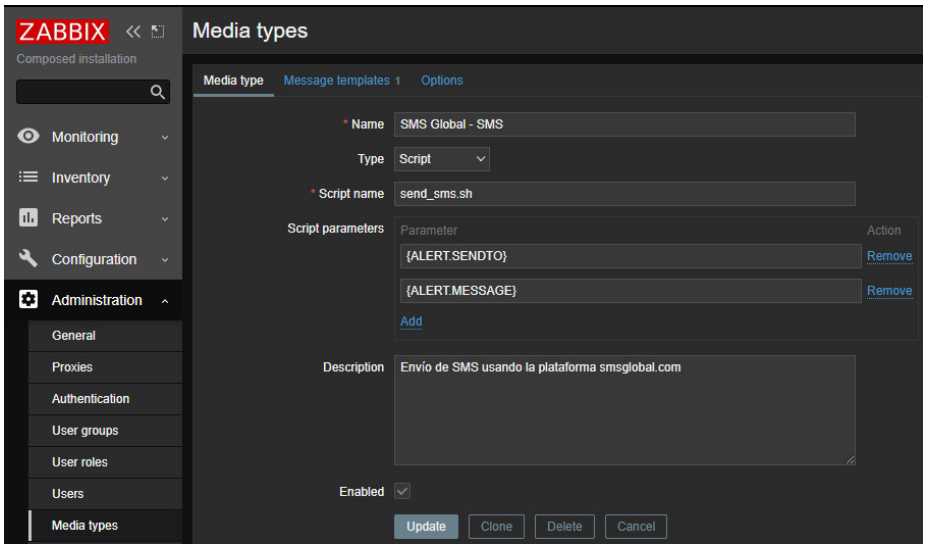


Figura D.87 Creación del media types SMS Global – SMS

Nota: En el parámetro *Type* se debe colocar Script y en el parámetro *Script name* se debe colocar el nombre del script *send_sms.sh* creado y copiado en el servidor Zabbix.

3. Vamos a la pestaña *Message templates* y añadimos está vez solo una plantilla para que funcione correctamente el media types.

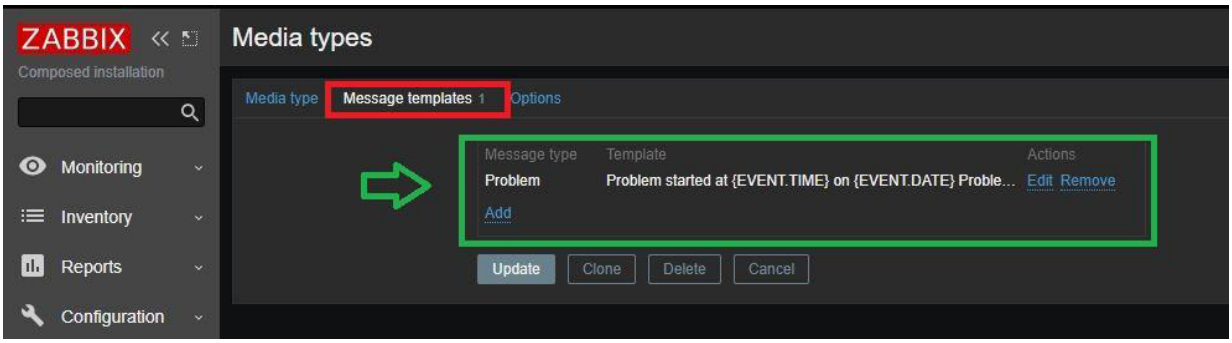


Figura D.88 Configuración del message templates del media types SMS Global – SMS

4. Una vez finalizado damos clic en el botón de *Add* y verificamos que se haya creado correctamente.

Figura D.89 Verificación de la creación del media types SMS Global – SMS

5. Una vez creado verificamos el funcionamiento correcto del media types "*SMS Global – SMS"* como el anterior media types.

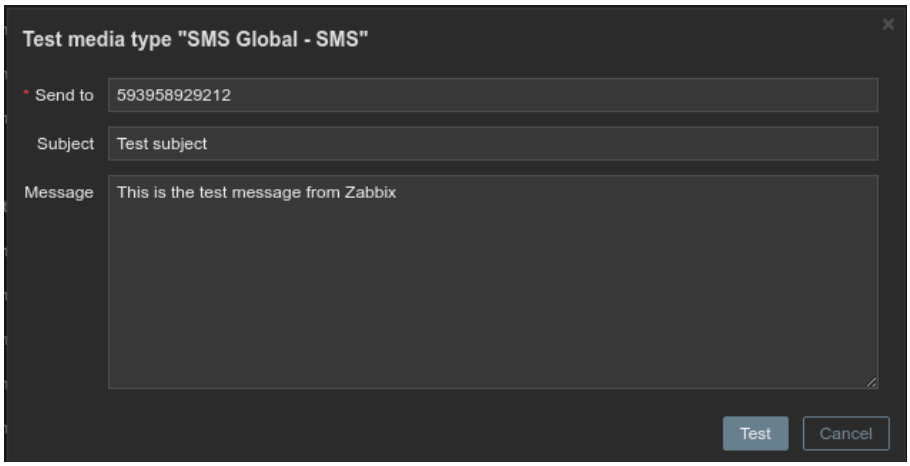


Figura D.90 Verificación del funcionamiento del media types SMS Global – SMS

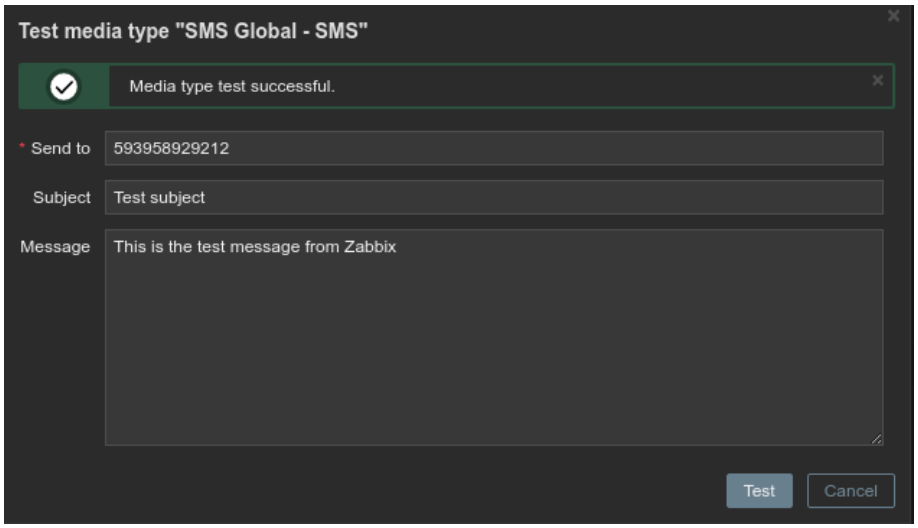


Figura D.91 Verificación del funcionamiento del media types SMS Global – SMS

D.4.4.3 CONFIGURACIÓN DE MEDIA TYPES PARA ENVIO DE NOTIFICACIONES CUANDO UN CLIENTE ODOO FALLA

1. Vamos al servidor de Zabbix y copiamos los scripts instance_reboot.py y call_function_python.sh que se mencionan en el Anexo D sección D.4 e D.5 para luego configurarlo dentro de la interfaz web de Zabbix. Ambos scripts deben ser copiados en la siguiente dirección:

cd zabbix docker/zbx_env/usr/lib/zabbix/alertscripts

2. Una vez copiado los scripts dentro del servidor Zabbix, vamos a la interfaz web del servidor y creamos un nuevo media type como se ha venido creando los otros, para este nuevo media type su nombre será "*Reboot Odoo Instance".*

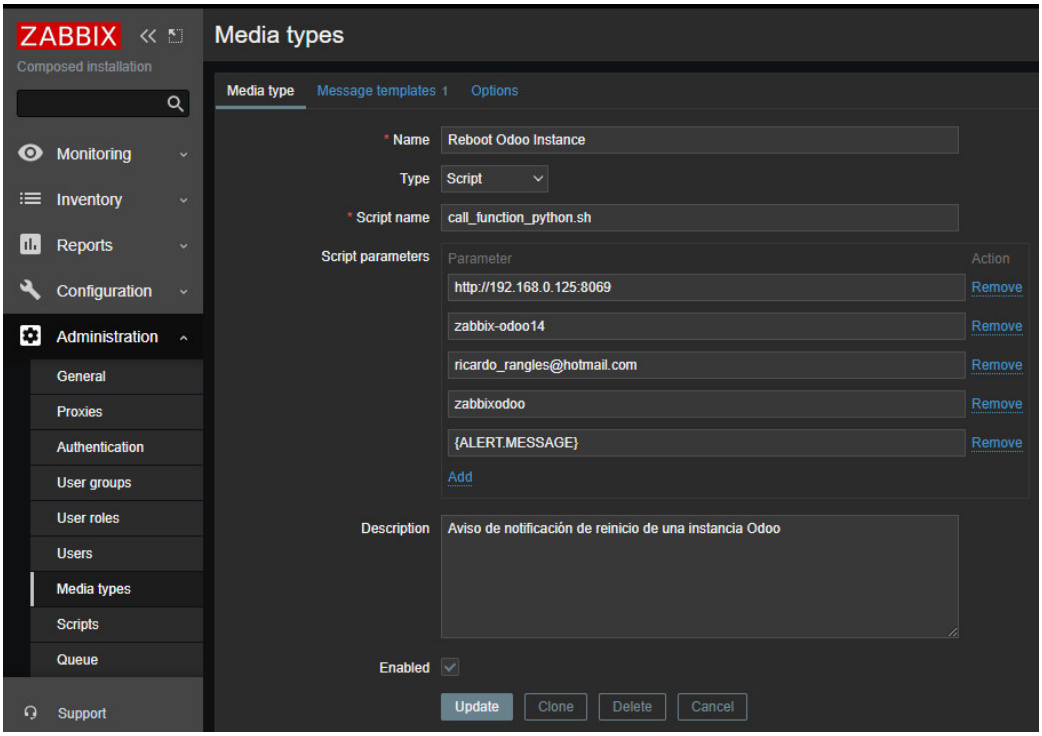


Figura D.92 Creación del media types Reboot Odoo Instance

Nota: En el parámetro *Type* se debe poner la opción Script y en el *parámetro Script* name el nombre del script *call_function_python.sh* debido a que Zabbix trabaja con script en Bash. En el parámetro *Script parameters* se debe colocar los parámetros para conectarse al servidor Odoo que va a servir para enviarle la información de que cliente Odoo ha fallado.

3. En la pestaña de Message templates copiamos la misma plantilla que se colocó en el media types *"SMS Global – SMS".*

4. Una vez finalizado todo damos clic en el botón *Add* y verificamos que se haya creado correctamente.

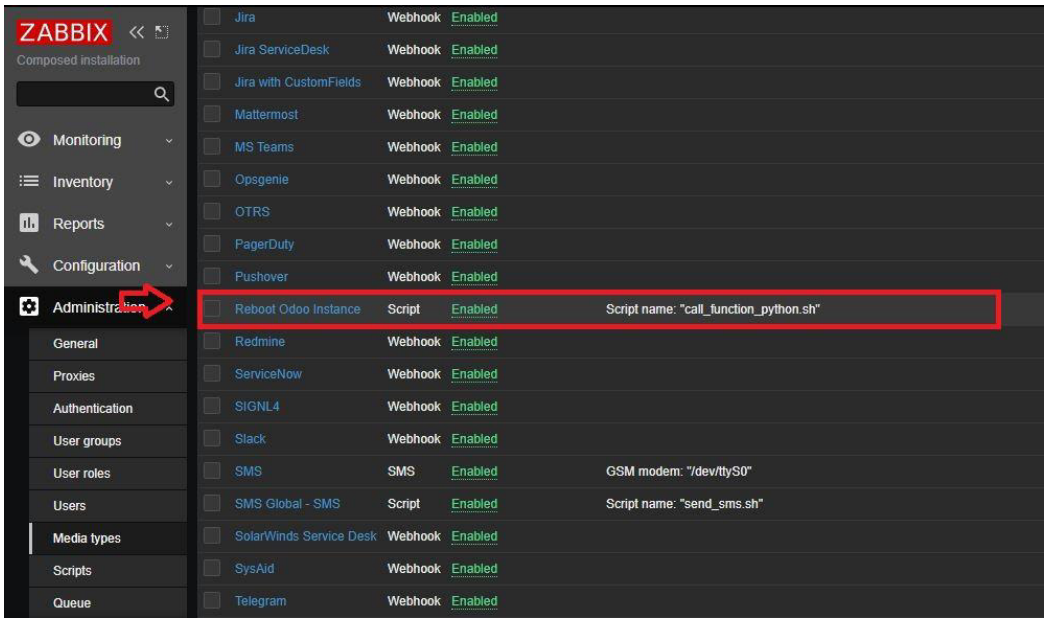


Figura D.93 Verificación de la creación del media types Reboot Odoo Instance

Nota: Para realizar la verificación de funcionamiento del media types Reboot Odoo Instance, el servidor Odoo al cuál se va a conectar Zabbix debe estar prendido.

D.5. USO DEL MÓDULO DE PRUEBA MODULO_PRUEBA_BOTONES

1. Para utilizar el módulo de prueba modulo_prueba_botones, lo que se debe hacer es copiar la carpeta llamada *modulo_prueba_botones* a su máquina, esta carpeta se encuentra ubicada en: https://github.com/TRESCLOUD/odoo-zabbix-monitoring.git
2. Una vez copiada la carpeta se debe añadir la ruta de la carpeta copiada al archivo odoo-server.conf, en la sección addons_path.

Figura D.94 Ejemplo de añadir la ruta de una carpeta en addons_path

3. Una vez colocado el path se reinicia el servicio de Odoo que tenga instalado en su máquina.

4. Al reiniciar el servicio Odoo, se debe ir a la interfaz web de Odoo e ir a la sección aplicaciones y buscar el nombre del módulo modulo_prueba_botones para poder instalar el módulo.

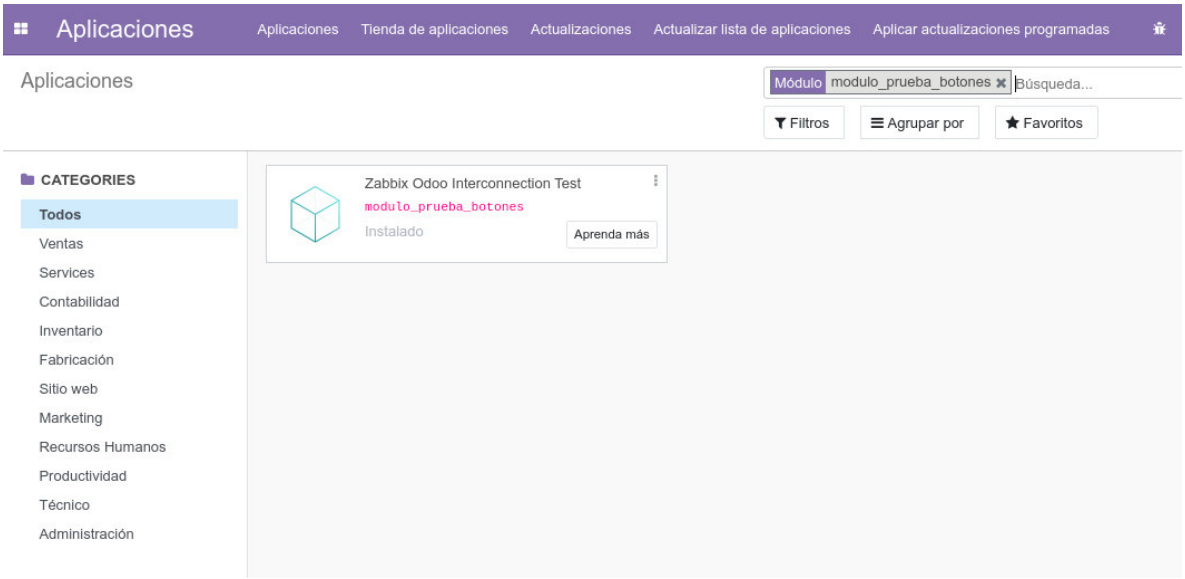


Figura D.95 Instalación del módulo zabbix_odoo_interconnection

5. Finalmente cuando se instale el módulo se lo puede utilizar sin problemas, cabe recalcar que este módulo es solo para probar las funcionalidades de crear y eliminar un cliente Odoo al sistema de monitoreo Zabbix; además todo el módulo puede ser modificado de acorde a su necesidad.