

Automatización De Entornos Productivos Con Uso De Software Libre



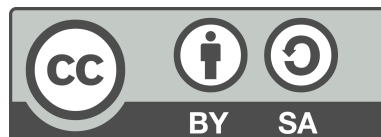
Francisco Javier Aguilera Aguilera

IES Medina Azahara

Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en Red

Córdoba, España

Junio de 2023



This work is licensed under CC BY-SA 4.0. To view a copy of this license, visit

<https://creativecommons.org/licenses/by-sa/4.0/>

| | |
|--|-----------|
| 1. DEFINICIÓN Y ANÁLISIS CONTEXTUAL | 4 |
| CONTEXTO Y JUSTIFICACIÓN | 4 |
| CLASIFICACIÓN DE EMPRESAS DEL SECTOR: | 4 |
| IDENTIFICACIÓN DE NECESIDADES MÁS DEMANDADAS: | 5 |
| CARACTERÍSTICAS ESPECÍFICAS DE PROYECTO: | 5 |
| MARCO LEGAL | 6 |
| ALCANCE DEL PROYECTO | 6 |
| ALTERNATIVAS Y PROPUESTA DE SOLUCIÓN | 7 |
| 2. ANÁLISIS DE REQUISITOS | 9 |
| 3. TEMPORALIZACIÓN | 10 |
| IDENTIFICACIÓN DE FASES Y TAREAS | 11 |
| CRONOGRAMA | 12 |
| 4. DOCUMENTACIÓN TÉCNICA | 12 |
| PREPARACIÓN DE PROXMOX | 13 |
| INSTALACIÓN DE SALT EN EL SERVIDOR | 15 |
| PREPARACIÓN DEL REPOSITORIO EN GITHUB | 18 |
| ESTRUCTURA DE FICHEROS DE SALT | 19 |
| STATES CREADOS | 21 |
| RESULTADOS DE LA INSTALACIÓN | 39 |
| 5. ANÁLISIS ECONÓMICO | 43 |
| 6. SEGUIMIENTO Y CONTROL | 45 |
| POSIBLES INCIDENCIAS | 45 |
| SOLUCIONES | 45 |
| ANÁLISIS DAFO | 46 |
| 7. FUENTES DE DOCUMENTACIÓN | 48 |
| ANEXO I GUÍA DE CLONACIÓN DEL REPOSITORIO | 49 |
| ANEXO II GUÍA DE APLICACIÓN DEL PROYECTO | 50 |
| ANEXO III GUÍA DE BORRADO DE CONTENEDORES | 51 |

1. DEFINICIÓN Y ANÁLISIS CONTEXTUAL

CONTEXTO Y JUSTIFICACIÓN

El movimiento del software libre ha transformado la administración de sistemas al ofrecer cuatro principios fundamentales: Ejecutar, estudiar, redistribuir y mejorar el software, proporcionando un control total sobre la infraestructura. La elección de software libre en este proyecto refleja el compromiso con una administración de sistemas flexible, adaptable y transparente, explorando los límites de automatización en este contexto.

Actualmente, se observa una marcada tendencia hacia la adopción de soluciones de código abierto en la administración de sistemas. Este cambio refleja la filosofía del software libre, promoviendo la transparencia y la colaboración en el desarrollo de infraestructuras tecnológicas. La virtualización, esencial en la administración moderna, ha experimentado un crecimiento considerable, con plataformas como Proxmox.

CLASIFICACIÓN DE EMPRESAS DEL SECTOR:

Las empresas del sector tecnológico que adoptan soluciones de software libre pueden clasificarse según sus características organizativas y el tipo de producto o servicio que ofrecen. Estas empresas incluyen desde startups innovadoras hasta grandes corporaciones establecidas. La estructura organizativa varía, pero comúnmente incluye departamentos de desarrollo, DevOps, soporte técnico y seguridad informática.

El departamento de desarrollo de software está encargado de la creación y mejora del código.

El departamento de DevOps se encarga de la implementación y mantenimiento de los

sistemas. El departamento de soporte técnico ofrece asistencia a los usuarios. El departamento de seguridad informática se encarga de proteger la infraestructura y los datos de la empresa.

IDENTIFICACIÓN DE NECESIDADES MÁS DEMANDADAS:

Las necesidades más demandadas por las empresas del sector incluyen soluciones sencillas y rápidas en su implementación. La capacidad de automatizar tareas repetitivas y complejas es altamente valorada.

Las oportunidades de negocio en el sector del software libre y la virtualización son significativas. Con el creciente interés en soluciones de código abierto, las empresas pueden aprovechar la reducción de costos de licencias y la capacidad de personalizar y adaptar software a sus necesidades específicas. Además, la demanda de servicios de soporte y consultoría en software libre está en aumento.

CARACTERÍSTICAS ESPECÍFICAS DE PROYECTO:

Las características específicas requeridas para el proyecto incluyen la implementación de herramientas de automatización para la administración de sistemas, el uso de plataformas de virtualización de código abierto como Proxmox y el desarrollo de procedimientos y políticas para garantizar la seguridad y eficiencia de la infraestructura.

Se ha invertido tiempo de este proyecto en investigar de forma detallada sobre las tecnologías de software libre y virtualización disponibles, incluyendo sus beneficios, limitaciones y casos de éxito. También se han investigado las mejores prácticas en automatización y administración de sistemas para asegurar que el proyecto esté alineado con los estándares actuales del sector.

MARCO LEGAL

Dado que el proyecto está dirigido a pequeñas y medianas empresas (PYMEs) y autónomos, es crucial considerar las particularidades y posibles ayudas que se pueden aplicar:

Obligaciones Fiscales:

Impuesto sobre Sociedades Simplificado: Las PYMEs pueden beneficiarse de regímenes fiscales simplificados y tipos impositivos reducidos en el Impuesto sobre Sociedades, siempre y cuando cumplan con los requisitos establecidos para ello.

Régimen Especial de IVA para PYMEs: Existen regímenes especiales de IVA que simplifican la declaración y pago de este impuesto para las PYMEs y autónomos, lo que les permite reducir la carga administrativa y fiscal.

Subvenciones y Ayudas:

Recientemente, el Gobierno ha ofrecido una ayuda de 1000 euros para autónomos con la que se puede comprar un equipo para la empresa. Esta ayuda es ideal para la implantación de este software, ya que con este presupuesto podemos alcanzar para un equipo de sobremesa que podemos usar a modo de servidor.

ALCANCE DEL PROYECTO

Objetivo General:

Implementar servicios como WordPress, MySQL, Prometheus y Apache en un entorno Proxmox, utilizando Salt como herramienta de automatización, con el propósito de optimizar la administración de entornos de producción de software libre.

Objetivos Específicos:

- Determinar la viabilidad y coherencia de la infraestructura como código (IaC) utilizando Salt para la gestión de servicios en entornos Proxmox.
- Evaluar el rendimiento, la escalabilidad y la velocidad de la implementación de servicios independientes bajo el enfoque de automatización con Salt.
- Identificar y abordar posibles desafíos de seguridad asociados con la automatización de servicios en entornos de producción de software libre.
- Aislar los servicios que usa un CMS como WordPress, de forma que su gestión posterior sea modular y mucho más clara de cara a facilitar el servicio técnico.
- Publicar el código utilizado en un repositorio como GitHub de forma pública, de forma que cualquier persona pueda ver, utilizar y aportar a este proyecto.

ALTERNATIVAS Y PROPUESTA DE SOLUCIÓN

ALTERNATIVAS:

Implementación Manual: Consiste en configurar y desplegar cada servicio de forma manual en el entorno Proxmox, lo que implica un alto grado de intervención humana y un mayor riesgo de errores.

Utilización de Herramientas de Automatización Alternativas: Se podría considerar el uso de otras herramientas de automatización como Ansible, Chef o Puppet en lugar de Salt, cada una con sus propias características y enfoques.

Contenedores Docker: Otra alternativa sería utilizar contenedores Docker para la implementación de los servicios, lo que podría simplificar la gestión y escalabilidad de los mismos.

PROPUESTA DE SOLUCIÓN:

Dado el objetivo de optimizar la administración de entornos de producción de software libre, se propone la siguiente solución:

Infraestructura como Código (IaC) con Salt para la Gestión de Servicios en Entornos Proxmox: Se utilizará SaltStack como herramienta principal de automatización, aprovechando su capacidad para gestionar la configuración y el estado de los sistemas de forma eficiente y escalable. La infraestructura se definirá como código, lo que permitirá una gestión más ágil y reproducible de los servicios, facilitando su despliegue y mantenimiento.

Se implementarán los servicios de WordPress, MySQL, Prometheus y Apache en Proxmox utilizando recetas predefinidas de Salt, que permitirán una configuración uniforme y coherente de los mismos.

Aislamiento y Modularidad: Se diseñará la arquitectura de los servicios de forma modular, aislando cada componente para facilitar su gestión posterior y garantizar la claridad y la independencia entre ellos. Esto se logrará utilizando contenedores LXC en lugar de máquinas virtuales tradicionales, lo que proporcionará mayor flexibilidad y eficiencia en la gestión de recursos.

Publicación del Código en GitHub: Todo el código utilizado en el proyecto, incluyendo las recetas de Salt y los archivos de configuración, se publicará en un repositorio público en GitHub. Esto permitirá que otros usuarios puedan acceder, revisar, contribuir y utilizar el proyecto, fomentando la colaboración y el intercambio de conocimientos en la comunidad de software libre.

Fundamentación: La elección de SaltStack como herramienta de automatización se fundamenta en su robustez, flexibilidad y amplia adopción en la comunidad de DevOps. Salt permite definir la infraestructura como código de manera eficiente, ofreciendo una gestión centralizada y escalable de los servicios en entornos Proxmox. La publicación del código en GitHub responderá a los principios de transparencia y colaboración que se buscan usando software libre.

2. ANÁLISIS DE REQUISITOS

RECURSOS HUMANOS

Para la implantación de este proyecto en una empresa real, sólo sería necesaria una persona. Si bien, esta persona debe estar formada profesionalmente en el ámbito DevOps y de sistemas operativos, ya que en caso de falla de los scripts, debe saber reaccionar, depurar y corregir el software.

REQUISITOS HARDWARE

- Será necesario un equipo con capacidades suficientes para levantar cuatro contenedores LXC y aguantar la instalación de un wordpress y el tráfico previsto. Como requerimientos mínimos diremos 4GB de RAM y cuatro núcleos de procesador.
- También es necesaria la capacidad de virtualización de la placa base y el procesador.
- Conexión a Internet para clonar el repositorio del proyecto y utilizar los states de SaltStack

REQUISITOS SOFTWARE

- El sistema operativo Proxmox instalado
- La instalación de SaltStack (master y minion) en el servidor
- Instalación de Git para poder clonar el repositorio

3. TEMPORALIZACIÓN

IDENTIFICACIÓN DE FASES Y TAREAS

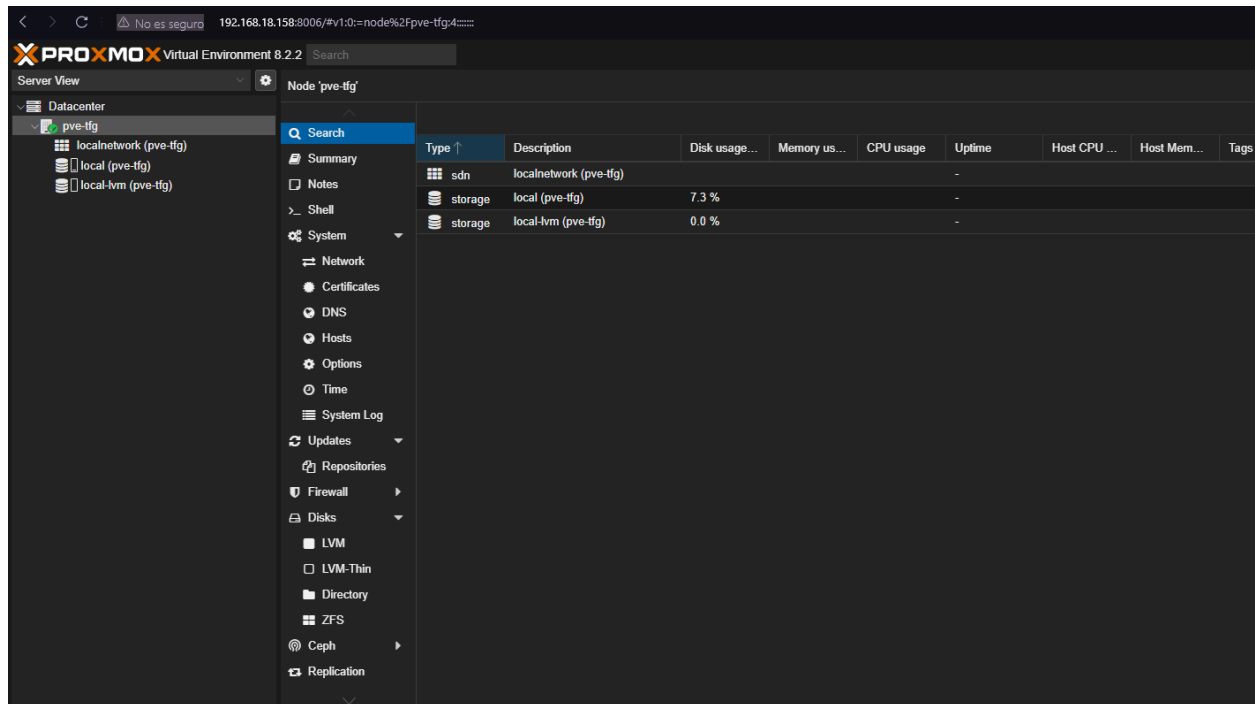
- Preparación del Entorno y Evaluación de Servidor: En esta fase se determinan los recursos disponibles para llevar el proyecto y se marcan los objetivos finales de este proyecto.
- Integración de Salt con Proxmox: Para llevar a cabo este proyecto, es necesario instalar el software de Salt. Primeramente se estudiará y comprenderá el software y posteriormente se realizará la instalación de Salt en el servidor. Se harán las pruebas necesarias para comprender su uso mediante states y orquestación.
- Implementación de Salt: Se van a desarrollar los states, que son un método de configuración remota de Salt que define el estado en el que debe estar un minion. Esta tarea será la labor principal de este proyecto y, por tanto, la tarea en la que más tiempo se va a invertir
- Orquestación de states de Salt: Una vez estén los states definidos e implementados, se va a organizar un sistema de orquestación para automatizar estos procesos y facilitar el uso de ellos
- Pruebas y Ajustes: Rendimiento y Escalabilidad: Se comprobará el funcionamiento de todo el código desarrollado y se contemplarán posibles mejoras para el proyecto
- Reposición del proyecto para hacerlo accesible a todo el mundo: El proyecto será colgado en la página web github.com para dotar de accesibilidad al mismo

CRONOGRAMA

| Actividad | Tiempo empleado |
|---|--------------------------|
| Preparación del Entorno, Evaluación de Servidor | 1 semana |
| Integración de Salt con Proxmox | 1 semana |
| Implementación de Salt: Instalación y Configuración. Desarrollo de Recetas y Estados | 3 semanas |
| Orquestación de Salt | 1 semana |
| Pruebas y Ajustes: Rendimiento y Escalabilidad | 1 semana |
| Reposición del proyecto para hacerlo accesible a todo el mundo | Durante todo el proyecto |

4. DOCUMENTACIÓN TÉCNICA

PREPARACIÓN DE PROXMOX



Partiremos de un servidor con Proxmox recién instalado. Merece la pena recordar que el servidor debe estar capacitado para soportar virtualización y debe tener capacidad de cómputo y memoria suficiente para aguantar cuatro contenedores con una imagen base de Debian y el servicio que se les asigne.

```
Linux pve-tfg 6.8.4-2-pve #1 SMP PREEMPT_DYNAMIC PMX 6.8.4-2 (2024-04-10T17:36Z) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May 27 19:05:57 CEST 2024 on pts/0
root@pve-tfg:~# apt update
Hit:1 http://security.debian.org bookworm-security InRelease
Hit:2 http://ftp.es.debian.org/debian bookworm InRelease
Hit:3 http://ftp.es.debian.org/debian bookworm-updates InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
6 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@pve-tfg:~#
```

A continuación, actualizaremos los repositorios de nuestro servidor. De esta manera, nos libramos de las posibles vulnerabilidades en nuestro sistema relacionadas con paquetes antiguos.

```
root@pve-tfg:~# apt install git
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.39.2-1.1).
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
root@pve-tfg:~#
```

Instalaremos git, nos será de utilidad más adelante para la clonación del repositorio donde guardaremos el código y será público. Para un servidor nuevo en el que queramos descargar el proyecto y usarlo también nos será necesario para conservar la estructura de los archivos.

INSTALACIÓN DE SALT EN EL SERVIDOR

Salt Project es una plataforma de automatización y gestión de configuración que permite automatizar la administración de infraestructura y aplicaciones en entornos distribuidos.

Salt proporciona herramientas para la ejecución remota, la gestión de configuraciones, el monitoreo en tiempo real y la orquestación de tareas complejas a gran escala, basándose en el principio de Infraestructura como Código (IaC). Nosotros haremos uso de estas herramientas para gestionar un servidor de forma remota y segura.

Para la instalación, haremos uso del proyecto Salt Bootstrap, un proyecto que automatiza la instalación mediante la ejecución de un script:

```
root@pve-tfg:~# curl -o bootstrap-salt.sh -L https://bootstrap.saltproject.io
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 346k 100 346k    0     0 1156k      0 --:--:-- --:--:-- --:--:-- 1159k
root@pve-tfg:~# chmod +x bootstrap-salt.sh
root@pve-tfg:~# ./bootstrap-salt.sh -M
```

```
curl -o bootstrap-salt.sh -L https://bootstrap.saltproject.io
```

Este comando nos descargará el script necesario, para ejecutarlo le daremos permisos con

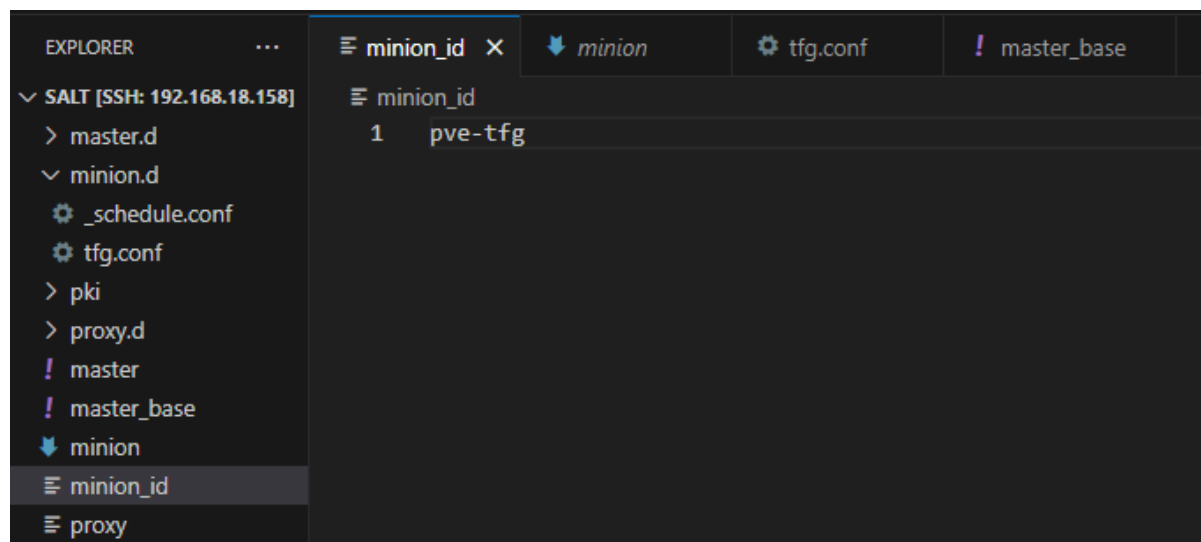
```
chmod +x bootstrap-salt.sh
```

Finalmente, lo ejecutaremos usando `./bootstrap-salt.sh -M`

El parámetro -M nos incluye en la instalación el salt-master, necesario para la estructura maestro - minions que utiliza Salt internamente

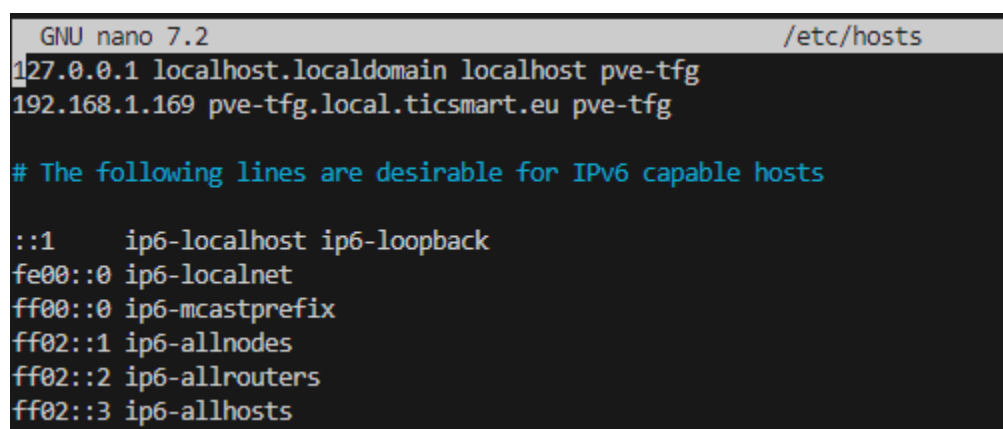
Para configurar Salt, tenemos la carpeta `/etc/salt` creada automáticamente. Las configuraciones restantes para empezar a usar Salt son:

- Asignación de nombre al equipo como salt-minion:

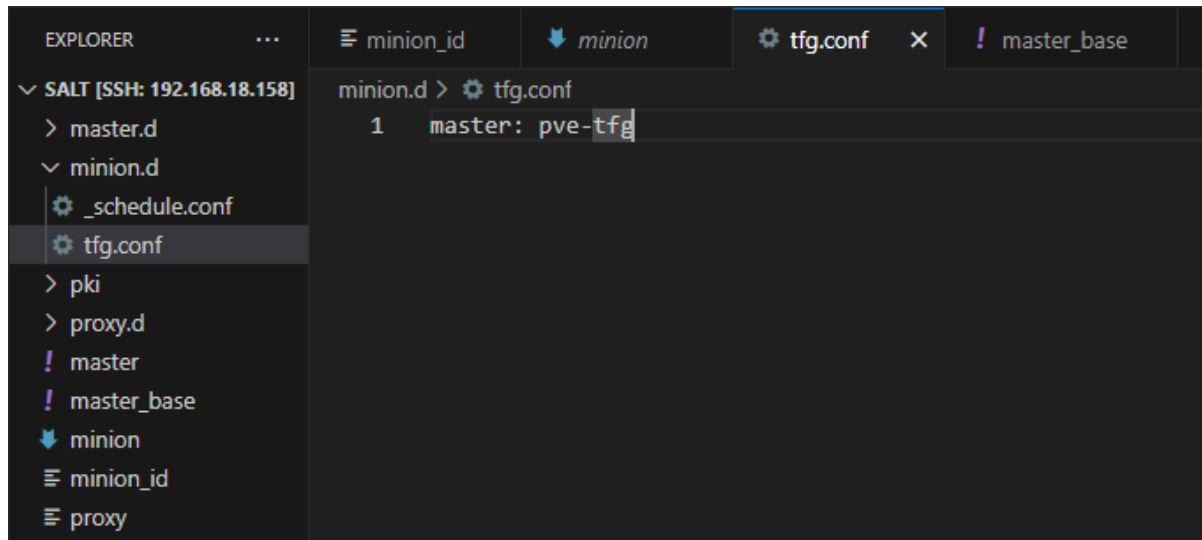


Para este caso, se le ha otorgado el nombre pve-tfg

- Asignación de un master al minion(en nuestro caso, apuntarse a sí mismo):



Apuntamos el nombre pve-tfg desde el archivo `/etc/hosts`



En /etc/salt/master.d/ creamos un archivo de configuración adicional donde definamos al master como pve-tfg, en cuanto reiniciemos el servicio salt-minion, este se pondrá en contacto con el master para intentar darle su key

- Admisión de la clave como salt-master:

```

root@pve-tfg:/etc/salt# systemctl status salt-minion
● salt-minion.service - The Salt Minion
   Loaded: loaded (/lib/systemd/system/salt-minion.service; enabled; preset: enabled)
   Active: active (running) since Mon 2024-05-27 20:00:48 CEST; 2s ago
     Docs: man:salt-minion(1)
           file:///usr/share/doc/salt/html/contents.html
           https://docs.saltproject.io/en/latest/contents.html
  Main PID: 14114 (python3.10)
    Tasks: 7 (limit: 9319)
   Memory: 56.8M
      CPU: 1.268s
   CGroup: /system.slice/salt-minion.service
           └─14114 /opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion
              14123 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion MultiMinionProcessManager MinionProc

May 27 20:00:47 pve-tfg systemd[1]: Starting salt-minion.service - The Salt Minion...
May 27 20:00:48 pve-tfg systemd[1]: Started salt-minion.service - The Salt Minion.
May 27 20:00:49 pve-tfg salt-minion[14123]: [ERROR ] The Salt Master has cached the public key for this node, th

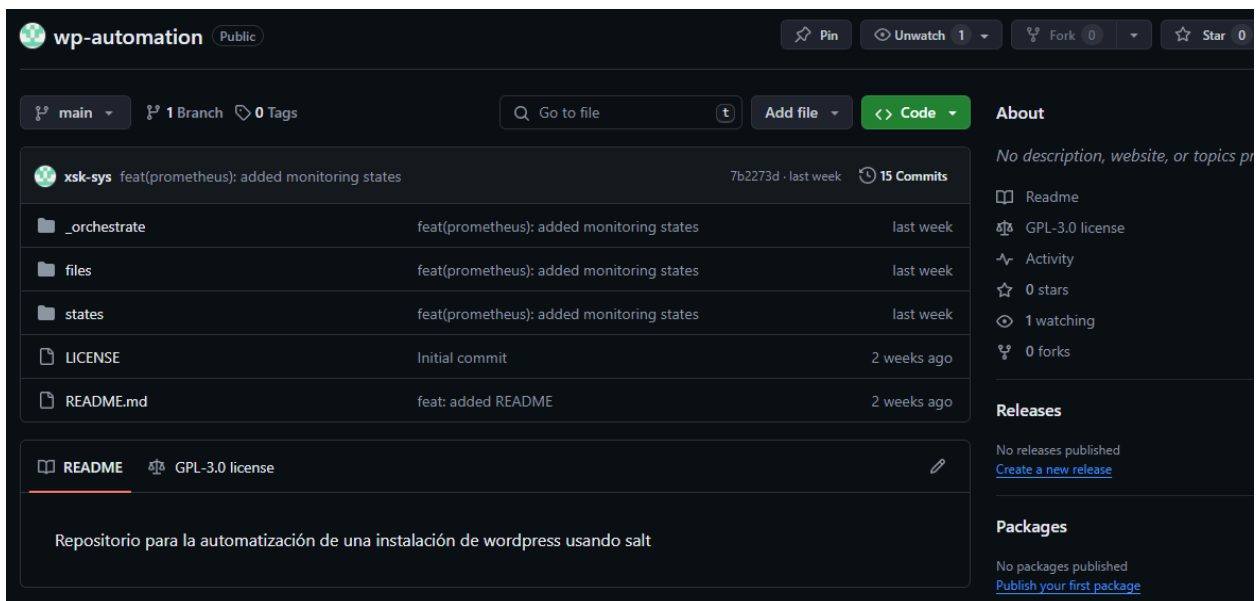
```

Revisando el servicio, vemos un error que nos dice que salt-minion ha encontrado a salt-master, pero está esperando que acepte nuestra clave

```
root@pve-tfg:/etc/salt# salt-key -A
The following keys are going to be accepted:
Unaccepted Keys:
pve-tfg
Proceed? [n/Y]
Key for minion pve-tfg accepted.
root@pve-tfg:/etc/salt#
```

Usando el comando salt-key -A aceptamos la clave del minion y salt queda configurado para poder atacarse a si mismo

PREPARACIÓN DEL REPOSITORIO EN GITHUB



The screenshot shows the GitHub interface for a repository named 'wp-automation' by the user 'xsk-sys'. The repository is public and has 15 commits. The main branch is selected. The repository contains several files and folders: '_orchestrate', 'files', 'states', 'LICENSE', and 'README.md'. The README file is highlighted, showing its content: 'Repositorio para la automatización de una instalación de wordpress usando salt'. The repository is licensed under GPL-3.0. The right sidebar shows the repository's activity, including 0 stars, 1 watching, and 0 forks. The 'Releases' and 'Packages' sections indicate that no releases or packages have been published yet.

Se ha inicializado un repositorio en GitHub que contendrá nuestro proyecto. Se encuentra publicado en <https://github.com/xsk-sys/wp-automation> y cualquier persona puede pedir nuevas features o fixes creando Pull Requests.

Para empezar a trabajar en él, nos debemos clonar el proyecto en /srv/salt, identificandonos antes usando

```
git config --global user.name "PacoAguil"
```

```
git config --global user.email "pacoaguil@outlook.es"
```

```
git clone https://github.com/PacoAguil/wp-automation.git
```

Desde aquí, podemos empezar a trabajar en el proyecto

ESTRUCTURA DE FICHEROS DE SALT

En /srv/salt/wp-automation hemos creado la siguiente estructura de carpetas

```
root@pve-tfg:/srv/salt/wp-automation# tree -d
.
├── files
├── _orchestrate
└── states

4 directories
root@pve-tfg:/srv/salt/wp-automation#
```

- En states, crearemos todos los states personalizados que usaremos en el despliegue.

Un ejemplo de su estructura puede ser:

```
create_prometheus_dir:
  file.directory:
    - name: /etc/prometheus
    - user: prometheus
    - group: prometheus
    - mode: 755
    - require:
      - user: prometheus
```

Desglosado, el state haría lo siguiente:

Usa el módulo `file.directory`, que asegura que exista un directorio con las características determinadas. El nombre es `/etc/prometheus/`. El usuario y grupo propietarios son `prometheus`. Los permisos son `755`. Si este directorio existe, el state se da por satisfecho, si por el contrario no existe lo crea. Para que se ejecute requiere que el usuario `prometheus` esté presente, en caso contrario, este fallará. Cada módulo de salt tiene documentadas sus posibilidades en la documentación oficial

<https://docs.saltproject.io/en/latest/ref/states/all/index.html>

- En files, colocaremos los archivos de configuración preconstruidos de la forma que nos interese, de forma que cuando estemos configurando los servicios, solo tengamos que reemplazar los archivos por los que nos interesan, estando ya precargados en el servidor.

Un ejemplo de archivo que usaremos puede ser la configuración de `mysql`:

```
[mysqld]
user      = mysql
pid-file  = /var/run/mysqld/mysqld.pid
socket    = /var/run/mysqld/mysqld.sock
datadir   = /var/lib/mysql
log-error = /var/log/mysql/error.log
symbolic-links = 0

# Allow external connections
bind-address = 0.0.0.0
```

- En `_orchestrate`, tenemos los ficheros de automatización. Dirigen los comandos a los hosts que deben y los secuencian, facilitando la implementación y despliegue.

STATES CREADOS

ct_init.sls

El primer paso que debemos dar es la creación de contenedores. En proxmox no debemos tener ninguna plantilla de contenedor inicialmente, por lo que descargamos la plantilla de Debian 11, a menos que ya exista.

```
# Descarga la plantilla de los contenedores inicial
download_debian_11_template:
  cmd.run:
    - name: pveam download local debian-11-standard_11.7-1_amd64.tar.zst
    - unless: ls /var/lib/vz/template/cache/ | grep debian-11
```

Posteriormente, crearemos los cuatro contenedores que utilizaremos. Además activaremos el nesting para que funcionen correctamente los servicios.

El contenedor 101 se encargará de la base de datos

```
# Inicialización de los contenedores
ct_mysql_init:
  cmd.run:
    - name: |
        pct create 101
        /var/lib/vz/template/cache/debian-11-standard_11.7-1_amd64.tar.zst \
        --hostname mysql \
        --net0
        name=eth0,bridge=vbr0,firewall=1,gw=192.168.18.1,ip=192.168.18.151/24,type=veth \
        --password serverftg \
        --storage local-lvm \
        --rootfs local-lvm:5 \
        --memory 1024 \
        --swap 0
    - unless: pct list |grep 101

set_nesting_mysql:
  cmd.run:
    - name: pct set 101 -features nesting=1
    - require:
      - cmd: ct_mysql_init
```

El contenedor 102 se encargará de wordpress

```
ct_wp_init:
  cmd.run:
    - name: |
        pct create 102
/var/lib/vz/template/cache/debian-11-standard_11.7-1_amd64.tar.zst \
        --hostname wp \
        --net0
name=eth0,bridge=vbr0,firewall=1,gw=192.168.18.1,ip=192.168.18.152/24,type=veth \
        --password serverftg \
        --storage local-lvm \
        --rootfs local-lvm:10 \
        --memory 2048 \
        --swap 0
    - unless: pct list |grep 102

set_nesting_wp:
  cmd.run:
    - name: pct set 102 -features nesting=1
    - require:
      - cmd: ct_wp_init
```

El contenedor 103 se encargará de usar un apache que haga de proxy inverso hacia el wordpress, en caso

```
ct_apache_init:
  cmd.run:
    - name: |
        pct create 103
/var/lib/vz/template/cache/debian-11-standard_11.7-1_amd64.tar.zst \
        --hostname apache \
        --net0
name=eth0,bridge=vbr0,firewall=1,gw=192.168.18.1,ip=192.168.18.153/24,type=veth \
        --password serverftg \
        --storage local-lvm \
        --rootfs local-lvm:5 \
        --memory 2048 \
        --swap 0
    - unless: pct list |grep 103

set_nesting_apache:
  cmd.run:
    - name: pct set 103 -features nesting=1
    - require:
      - cmd: ct_apache_init
```

El contenedor 104 se encargará de la monitorización con prometheus

```
ct_prometheus_init:
  cmd.run:
    - name: |
        pct create 104
/var/lib/vz/template/cache/debian-11-standard_11.7-1_amd64.tar.zst \
        --hostname prometheus \
        --net0
name=eth0,bridge=vmbro0,firewall=1,gw=192.168.18.1,ip=192.168.18.154/24,type=veth \
        --password servertfg \
        --storage local-lvm \
        --rootfs local-lvm:5 \
        --memory 2048 \
        --swap 0
    - unless: pct list |grep 104

set_nesting_prometheus:
  cmd.run:
    - name: pct set 104 -features nesting=1
    - require:
      - cmd: ct_prometheus_init
```

Cuando este state termine, los contenedores quedarán montados. Sin embargo, queremos manejarlos de forma sencilla y haciendo uso de salt, por lo que aún tenemos que instalarles el salt-minion para que nuestro servidor pueda reconocerlos de forma remota. De esta forma, atacaremos directamente a cada contenedor.

salt_install.sls

En primer lugar, arrancaremos los contenedores

```
ensure_mysql_running:
  cmd.run:
    - name: |
        if [ "$(pct status 101 | awk '{print $2}')" != "running" ]; then
          pct start 101
          # Wait for the container to start
          while [ "$(pct status 101 | awk '{print $2}')" != "running" ]; do
            sleep 1
          done
        fi
    - shell: /bin/bash

ensure_wp_running:
  cmd.run:
    - name: |
        if [ "$(pct status 102 | awk '{print $2}')" != "running" ]; then
          pct start 102
          # Wait for the container to start
          while [ "$(pct status 102 | awk '{print $2}')" != "running" ]; do
            sleep 1
          done
        fi
    - shell: /bin/bash

ensure_apache_running:
  cmd.run:
    - name: |
        if [ "$(pct status 103 | awk '{print $2}')" != "running" ]; then
          pct start 103
          # Wait for the container to start
          while [ "$(pct status 103 | awk '{print $2}')" != "running" ]; do
            sleep 1
          done
        fi
    - shell: /bin/bash

ensure_prometheus_running:
  cmd.run:
    - name: |
        if [ "$(pct status 104 | awk '{print $2}')" != "running" ]; then
          pct start 104
          # Wait for the container to start
          while [ "$(pct status 104 | awk '{print $2}')" != "running" ]; do
```



```
        sleep 1
    done
fi
- shell: /bin/bash
```

A continuación, entraremos a cada contenedor ejecutando el script de Salt Bootstrap que ejecutamos anteriormente, sin pasar el parámetro de instalación de salt-master.

El parámetro del que sí haremos uso es -A, que pregunta el nombre del master al que vamos a atacar.

```
mysql_install_salt:
  cmd.run:
    - name: |
        pct exec 101 -- bash -c "apt update && wget -O bootstrap-salt.sh
https://bootstrap.saltproject.io && sh bootstrap-salt.sh -A 192.168.18.158 stable
3007.1"
    - onlyif: pct exec 101 -- bash -c "which salt-minion || echo 'not_installed'"
    - unless: pct exec 101 -- bash -c "which salt-minion"
    - require:
        - cmd: ensure_mysql_running
    - shell: /bin/bash

wp_install_salt:
  cmd.run:
    - name: |
        pct exec 102 -- bash -c "apt update && wget -O bootstrap-salt.sh
https://bootstrap.saltproject.io && sh bootstrap-salt.sh -A 192.168.18.158 stable
3007.1"
    - onlyif: pct exec 102 -- bash -c "which salt-minion || echo 'not_installed'"
    - unless: pct exec 102 -- bash -c "which salt-minion"
    - require:
        - cmd: ensure_wp_running
    - shell: /bin/bash

apache_install_salt:
  cmd.run:
    - name: |
        pct exec 103 -- bash -c "apt update && wget -O bootstrap-salt.sh
https://bootstrap.saltproject.io && sh bootstrap-salt.sh -A 192.168.18.158 stable
3007.1"
    - onlyif: pct exec 103 -- bash -c "which salt-minion || echo 'not_installed'"
    - unless: pct exec 103 -- bash -c "which salt-minion"
    - require:
```

```

- cmd: ensure_apache_running
- shell: /bin/bash

prometheus_install_salt:
  cmd.run:
    - name: |
        pct exec 104 -- bash -c "apt update && wget -O bootstrap-salt.sh
        https://bootstrap.saltproject.io && sh bootstrap-salt.sh -A 192.168.18.158 stable
        3007.1"
    - onlyif: pct exec 104 -- bash -c "which salt-minion || echo 'not_installed'"
    - unless: pct exec 104 -- bash -c "which salt-minion"
    - require:
        - cmd: ensure_prometheus_running
        - shell: /bin/bash

```

Por último, aceptaremos directamente todas las claves que hayan llegado al salt-master del servidor

```

accept_keys_on_master_srv:
  cmd.run:
    - name: salt-key -Ay

```

Una vez se ha inicializado todo, podemos crear el primer state de orquestación, que llamaremos init.sls

orchestrate/init.sls

```

#Ejecutor del state ct_init en el minion pve-tfg
execute_ct_init:
  salt.state:
    - tgt: 'pve-tfg'
    - sls: wp-automation.states.ct_init

#Ejecutor del state salt_install en el minion pve-tfg
execute_salt_install:
  salt.state:
    - tgt: 'pve-tfg'
    - sls: wp-automation.states.salt_install

```

Todos los contenedores están listos, por lo que vamos a empezar a ejecutar las instalaciones de sus servicios

mysql.sls

```
# Se comprueba que el paquete está instalado, si no
# lo estuviera, lo instala
install-mariadb:
  pkg.installed:
    - name: mariadb-server
# Se lleva el archivo de configuración de mariadb en files
# para usar configuración personalizada
configure_mariadb:
  file.managed:
    - name: /etc/mysql/my.cnf
    - source: salt://wp-automation/files/my.cnf
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: install-mariadb

# Inicia y habilita el servicio
start_mariadb_service:
  service.running:
    - name: mariadb
    - enable: True
    - require:
      - pkg: install-mariadb
      - file: configure_mariadb

# Establece contraseña para el admin de mariadb
set_mysql_root_password:
  cmd.run:
    - name: |
        mysqladmin -u root password 'root_passwd'
    - unless: mysql -u root -p'root_passwd' -e "SHOW DATABASES;"
    - require:
      - service: start_mariadb_service

# Crea una base de datos vacía
create_wp_database:
  cmd.run:
    - name: mysql -u root -e "CREATE DATABASE IF NOT EXISTS wpdb;"
```

```
# Crea un usuario de la base de datos que solo puede conectar desde la ip del
contenedor de wordpress
create_wp_user:
  cmd.run:
    - name: mysql -u root -e "CREATE USER IF NOT EXISTS 'wpuser'@'192.168.18.152'
IDENTIFIED BY 'MyStrongPassword';"
    - require:
      - cmd: create_wp_database

# Da privilegios al nuevo usuario en la base de datos que nos interesa
grant_wp_user:
  cmd.run:
    - name: mysql -u root -e "GRANT ALL PRIVILEGES ON wpdb.* TO
'wpuser'@'192.168.18.152';"

# Refresca la tabla de privilegios
flush_priv:
  cmd.run:
    - name: mysql -u root -e "FLUSH PRIVILEGES;"

# Reinicia el servicio
restart_mariadb:
  cmd.run:
    - name: systemctl restart mariadb
```

apache_init.sls

```
# Instala el paquete de apache
install_apache:
  pkg.installed:
    - name: apache2

# Lleva el archivo de configuración de apache de files a sites-available
apache_config:
  file.managed:
    - name: /etc/apache2/sites-available/wp.conf
    - source: salt://wp-automation/files/wp-apache.conf
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: install_apache
```

```
# Configura apache para leer los ficheros index.php primero
index_config:
  file.managed:
    - name: /etc/apache2/mods-available/dir.conf
    - source: salt://wp-automation/files/dirmod.conf
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: install_apache

# instala los modulos necesarios para un proxy inverso, wordpress y ssl
enable_modules:
  cmd.run:
    - name: >
      a2enmod ssl rewrite proxy proxy_http

# Quita el archivo de configuración predeterminada de apache
apache_default_disabled:
  cmd.run:
    - name: a2dissite 000-default
    - unless: ls /etc/apache2/sites-enabled/ |grep wp.conf

# Habilita el archivo de configuración de wordpress
apache_site_enabled:
  file.symlink:
    - name: /etc/apache2/sites-enabled/wp.conf
    - target: /etc/apache2/sites-available/wp.conf
    - require:
      - file: apache_config

# Reinicia el servicio de apache
apache_on:
  service.running:
    - name: apache2
    - enable: True
    - require:
      - pkg: install_apache
      - file: apache_site_enabled

restart_apache:
  cmd.run:
    - name: systemctl restart apache2
```

wp_init.sls

```
# Instala los paquetes necesarios para un wordpress en Debian
install_packages:
  pkg.installed:
    - pkgs:
      - apache2
      - php
      - libapache2-mod-php
      - php-mysql
      - curl

# Gestiona el endpoint de wordpress
apache_config:
  file.managed:
    - name: /etc/apache2/sites-available/wp.conf
    - source: salt://wp-automation/files/wp-endpoint.conf
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: install_packages

# Configura apache para leer los ficheros index.php primero
index_config:
  file.managed:
    - name: /etc/apache2/mods-available/dir.conf
    - source: salt://wp-automation/files/dirmod.conf
    - user: root
    - group: root
    - mode: 644
    - require:
      - pkg: install_packages

# Habilita los modulos necesarios
enable_modules:
  cmd.run:
    - name: >
      a2enmod ssl rewrite proxy proxy_http

# Desactiva la página por defecto
apache_default_disabled:
  cmd.run:
    - name: a2dissite 000-default
    - unless: ls /etc/apache2/sites-enabled/ |grep wp.conf

# Habilita la de wordpress
apache_site_enabled:
  file.symlink:
    - name: /etc/apache2/sites-enabled/wp.conf
```

```
- target: /etc/apache2/sites-available/wp.conf
- require:
  - file: apache_config
# Reinicia el servicio de apache
apache_on:
  service.running:
    - name: apache2
    - enable: True
    - require:
      - pkg: install_packages
      - file: apache_site_enabled

restart_apache:
  cmd.run:
    - name: systemctl restart apache2
# Descarga los archivos de wordpress
download_wp:
  cmd.run:
    - name: curl -o /root/latest.tar.gz https://wordpress.org/latest.tar.gz
    - unless: ls /root | grep latest.tar.gz
# Extrae los archivos de wordpress
extract_wordpress:
  cmd.run:
    - name: tar -xzf /root/latest.tar.gz -C /var/www/
    - unless: ls /var/www |grep wordpress
# Cambio de permisos a los archivos de wordpress
chown_wp:
  cmd.run:
    - name: chown -R www-data:www-data /var/www/wordpress

chmodd_wp:
  cmd.run:
    - name: find /var/www/wordpress/ -type d -exec chmod 750 {} \;

chmodf_wp:
  cmd.run:
    - name: find /var/www/wordpress/ -type f -exec chmod 640 {} \;
# Gestiona el archivo de configuración de wordpress
copy_wp_config:
  file.managed:
    - name: /var/www/wordpress/wp-config.php
    - source: salt://wp-automation/files/wp-config.php
```

prometheus.sls

```
# Descarga los archivos de prometheus
download_prometheus:
  file.managed:
    - name: /root/prometheus-2.52.0.linux-amd64.tar.gz
    - source:
      https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.l
      inux-amd64.tar.gz
    - skip_verify: True
    - makedirs: True
    - unless: test -f /root/prometheus-2.52.0.linux-amd64.tar.gz
# Extrae los archivos
extract_prometheus:
  cmd.run:
    - name: tar -xzf /root/prometheus-2.52.0.linux-amd64.tar.gz -C /root/
    - creates: /root/prometheus-2.52.0.linux-amd64
    - require:
      - file: download_prometheus
# Crea el usuario prometheus
create_prometheus_user:
  user.present:
    - name: prometheus
    - createhome: False
    - shell: /bin/false
# Crea los directorios de configuración de prometheus
create_prometheus_dir:
  file.directory:
    - name: /etc/prometheus
    - user: prometheus
    - group: prometheus
    - mode: 755
    - require:
      - user: prometheus

another_prometheus_dir:
  file.directory:
    - name: /var/lib/prometheus
    - user: prometheus
    - group: prometheus
    - mode: 755
    - require:
      - user: prometheus
```



```
# Crea el comando prometheus
copy_prometheus_binary:
  file.managed:
    - name: /usr/local/bin/prometheus
    - source: /root/prometheus-2.52.0.linux-amd64/prometheus
    - user: prometheus
    - group: prometheus
    - mode: 755

# Crea el comando promtool
copy_promtool_binary:
  file.managed:
    - name: /usr/local/bin/promtool
    - source: /root/prometheus-2.52.0.linux-amd64/promtool
    - user: prometheus
    - group: prometheus
    - mode: 755

# Crea el directorio consoles
copy_consoles_directory:
  cmd.run:
    - name: 'cp -r /root/prometheus-2.52.0.linux-amd64/consoles /etc/prometheus'
    - unless: 'test -d /etc/prometheus/consoles'
    - user: root
    - require:
      - user: prometheus

# Crea el directorio console_libraries
copy_console_libraries_directory:
  cmd.run:
    - name: 'cp -r /root/prometheus-2.52.0.linux-amd64/console_libraries
/etc/prometheus'
    - unless: 'test -d /etc/prometheus/console_libraries'
    - user: root
    - require:
      - user: prometheus
```

```
# Añade configuración personalizada al final del archivo de configuración
append_prometheus_config:
  file.append:
    - name: /etc/prometheus/prometheus.yml
    - text: |
        global:
          scrape_interval: 10s

        scrape_configs:
          - job_name: 'prometheus'
            scrape_interval: 5s
            static_configs:
              - targets: ['localhost:9200']
          - job_name: 'mysql'
            scrape_interval: 5s
            static_configs:
              - targets: ['192.168.18.151:9200']

          - job_name: 'wp'
            scrape_interval: 5s
            static_configs:
              - targets: ['192.168.18.152:9200']

          - job_name: 'apache'
            scrape_interval: 5s
            static_configs:
              - targets: ['192.168.18.153:9200']
# Asignación de permisos al archivo de configuración
set_prometheus_config_permissions:
  file.managed:
    - name: /etc/prometheus/prometheus.yml
    - user: prometheus
    - group: prometheus
    - mode: 644
    - require:
      - file: append_prometheus_config
# Crea el servicio desde el archivo que tenemos en files
deploy_prometheus_service:
  file.managed:
    - name: /etc/systemd/system/prometheus.service
    - source: salt://wp-automation/files/prometheus.service
    - user: root
    - group: root
    - mode: 644
```

```
# Recarga los demonios del sistema
reload_systemd:
  cmd.run:
    - name: systemctl daemon-reload
    - require:
      - file: deploy_prometheus_service
# Habilita el servicio
enable_prometheus_service:
  service.enabled:
    - name: prometheus
    - require:
      - cmd: reload_systemd
# Inicia el servicio
start_prometheus_service:
  service.running:
    - name: prometheus
    - enable: True
    - require:
      - file: /etc/systemd/system/prometheus.service
```

Todos estos states pueden ser englobados en un archivo del orquestador. Lo hemos llamado `deploy.sls`

`orchestrate/deploy.sls`

```
# Inicia mysql.sls en el minion mysql
execute_mysql:
  salt.state:
    - tgt: 'mysql'
    - sls: wp-automation.states.mysql
# Inicia apache_init en el minion apache
execute_apache_init:
  salt.state:
    - tgt: 'apache'
    - sls: wp-automation.states.apache_init
# Inicia wp_init.sls en el minion wp
execute_wp_init:
  salt.state:
    - tgt: 'wp'
    - sls: wp-automation.states.wp_init
# Inicia prometheus.sls en el minion prometheus
execute_state_prometheus:
  salt.state:
    - tgt: 'prometheus'
    - sls: wp-automation.states.prometheus
```

A pesar de que hemos instalado prometheus, aún no tenemos ningún servicio en los minion que exporte las métricas a monitorizar, por lo que vamos a implementar un nuevo state

node_exporter_install.sls

```
# Descarga node_exporter, un servicio exportador de métricas
download_node_exporter:
  file.managed:
    - name: /root/node_exporter-1.8.1.linux-amd64.tar.gz
    - source:
      https://github.com/prometheus/node_exporter/releases/download/v1.8.1/node_exporter-1.8.1.linux-amd64.tar.gz
    - skip_verify: True
    - unless: ls /root | grep node_exporter
# Crea un usuario y un grupo node_exporter
create_user:
  cmd.run:
    - name: groupadd -f node_exporter && useradd -g node_exporter --no-create-home --shell /bin/false node_exporter
    - unless: cat /etc/passwd | grep node_exporter
# Extrae los archivos del tar.gz
extract_node_exporter:
  archive.extracted:
    - name: /root/node_exporter-1.8.1.linux-amd64
    - source: /root/node_exporter-1.8.1.linux-amd64.tar.gz
    - unless: ls /root/node_exporter-1.8.1.linux-amd64/ | grep node_exporter
# Mueve el archivo a /usr/bin a menos que ya exista
move_node_exporter:
  file.managed:
    - name: /usr/bin/node_exporter
    - source:
      /root/node_exporter-1.8.1.linux-amd64/node_exporter-1.8.1.linux-amd64/node_exporter
    - user: root
    - group: root
    - mode: '0755'
    - replace: True
    - unless: ls /usr/bin/node_exporter | grep node_exporter
# Crea el servicio node_exporter
node_exporter_service:
  file.managed:
    - name: /usr/lib/systemd/system/node_exporter.service
    - source: salt://wp-automation/files/node_exporter.service
    - user: root
    - group: root
    - mode: '0755'
    - replace: True
```

```
- unless: ls /usr/lib/systemd/system | grep node_exporter
# Otorga permisos al archivo node_exporter.service
permissions_exporter:
  cmd.run:
    - name: chmod 664 /usr/lib/systemd/system/node_exporter.service
# Reinicia los demonios y servicios
reload_and_start:
  cmd.run:
    - name: systemctl daemon-reload && systemctl start node_exporter && systemctl
enable node_exporter
```

Crearemos en el orquestador un state nuevo para que se ejecute en los minion que toca

_orchestrate/node_exporter.sls

```
node_exporter_srv:
  salt.state:
    - tgt: 'pve-tfg'
    - sls: wp-automation.states.node_exporter_install

node_exporter_mysql:
  salt.state:
    - tgt: 'mysql'
    - sls: wp-automation.states.node_exporter_install

node_exporter_wp:
  salt.state:
    - tgt: 'wp'
    - sls: wp-automation.states.node_exporter_install

node_exporter_apache:
  salt.state:
    - tgt: 'apache'
    - sls: wp-automation.states.node_exporter_install

node_exporter_prometheus:
  salt.state:
    - tgt: 'prometheus'
    - sls: wp-automation.states.node_exporter_install
```

Para facilitar aún más las cosas, podemos automatizar procesos del orquestador, de forma que solo tengamos que ejecutar un state

`orchestrate/main.sls`

```
include:
  - wp-automation._orchestrate.deploy
  - wp-automation._orchestrate.node_exporter
```

Otros states que no se llegan a utilizar en la automatización, pero son interesantes para dejar en el proyecto por la utilidad que presentan para el desarrollo de código son:

`ping.sls`

```
ping_all_minions:
  module.run:
    - name: test.ping
```

Ping para comprobar que los minion estén funcionando y conectados al master

`delete_ct.sls`

```
delete_ct_101:
  cmd.run:
    - name: pct stop 101 && pct destroy 101
    - ignore_errors: True

delete_ct_102:
  cmd.run:
    - name: pct stop 102 && pct destroy 102
    - ignore_errors: True

delete_ct_103:
  cmd.run:
    - name: pct stop 103 && pct destroy 103
    - ignore_errors: True

delete_ct_104:
  cmd.run:
    - name: pct stop 104 && pct destroy 104
    - ignore_errors: True
```

Borra los contenedores para poder relanzar el proyecto

rm_saltkeys.sls

```
delete_mysql_key:
  cmd.run:
    - name: salt-key -d mysql -y

delete_wp_key:
  cmd.run:
    - name: salt-key -d wp -y

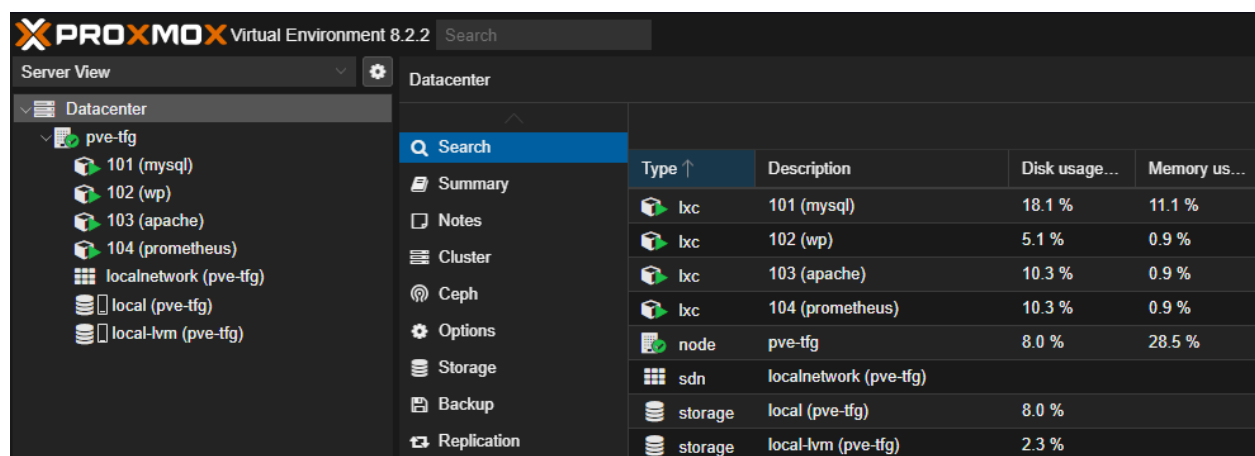
delete_apache_key:
  cmd.run:
    - name: salt-key -d apache -y

delete_prometheus_key:
  cmd.run:
    - name: salt-key -d prometheus -y
```

Borra las claves para evitar errores de contenedores antiguos

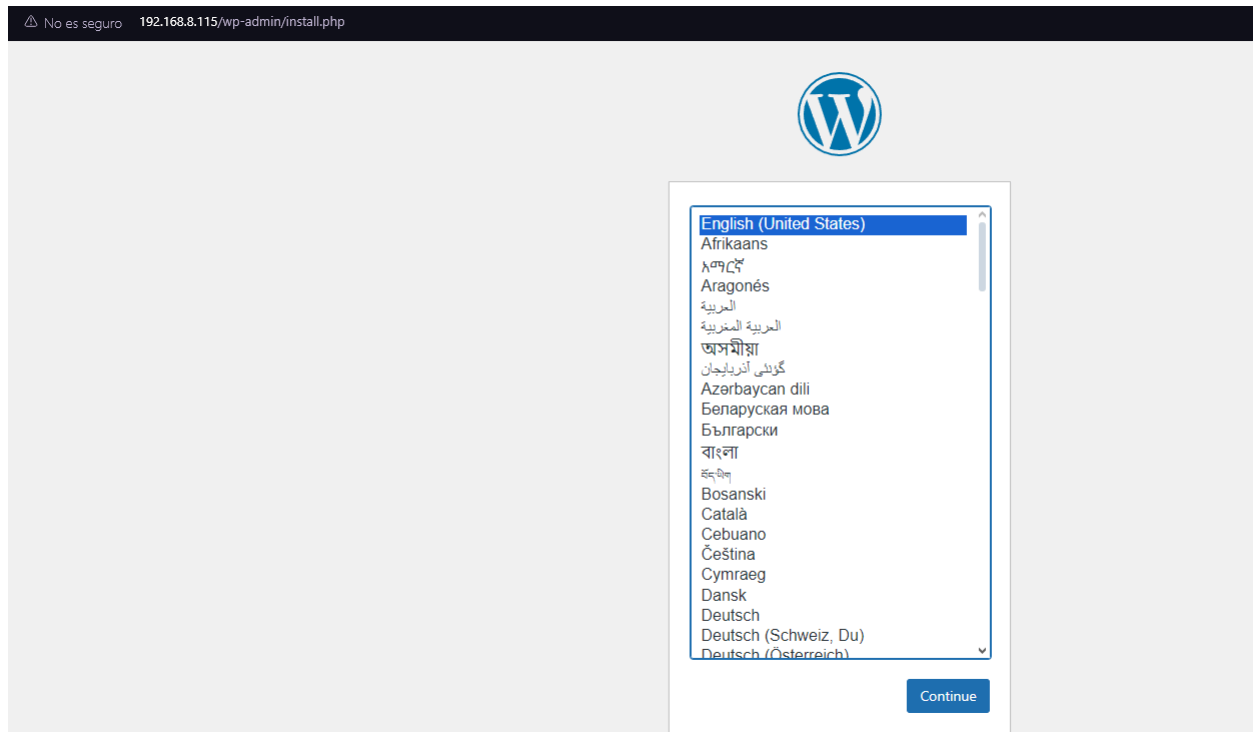
RESULTADOS DE LA INSTALACIÓN

Una vez ejecutados los states que realizan la instalación de todo el sistema que hemos propuesto, podemos ver los resultados de las máquinas creadas en Proxmox



| Type | Description | Disk usage... | Memory us... |
|---------|------------------------|---------------|--------------|
| lxc | 101 (mysql) | 18.1 % | 11.1 % |
| lxc | 102 (wp) | 5.1 % | 0.9 % |
| lxc | 103 (apache) | 10.3 % | 0.9 % |
| lxc | 104 (prometheus) | 10.3 % | 0.9 % |
| node | pve-tfg | 8.0 % | 28.5 % |
| sdn | localnetwork (pve-tfg) | | |
| storage | local (pve-tfg) | 8.0 % | |
| storage | local-lvm (pve-tfg) | 2.3 % | |

El wordpress ya ha quedado creado y conectado con su base de datos, accesible a través del servidor apache que tenemos funcionando como proxy inverso. La IP interna final sobre la que se ha servido apache es 192.168.8.115



En el caso de que la base de datos no hubiera quedado correctamente conectada con wordpress, esta página no sería visible. Ver esta página es un seguro de que la conexión es correcta.

La siguiente comprobación que podemos realizar es el sistema de monitorización, para comprobar que el exportador de métricas está funcionando en cada nodo podemos acceder a cada IP a través del puerto 9090 y la URI /metrics


```

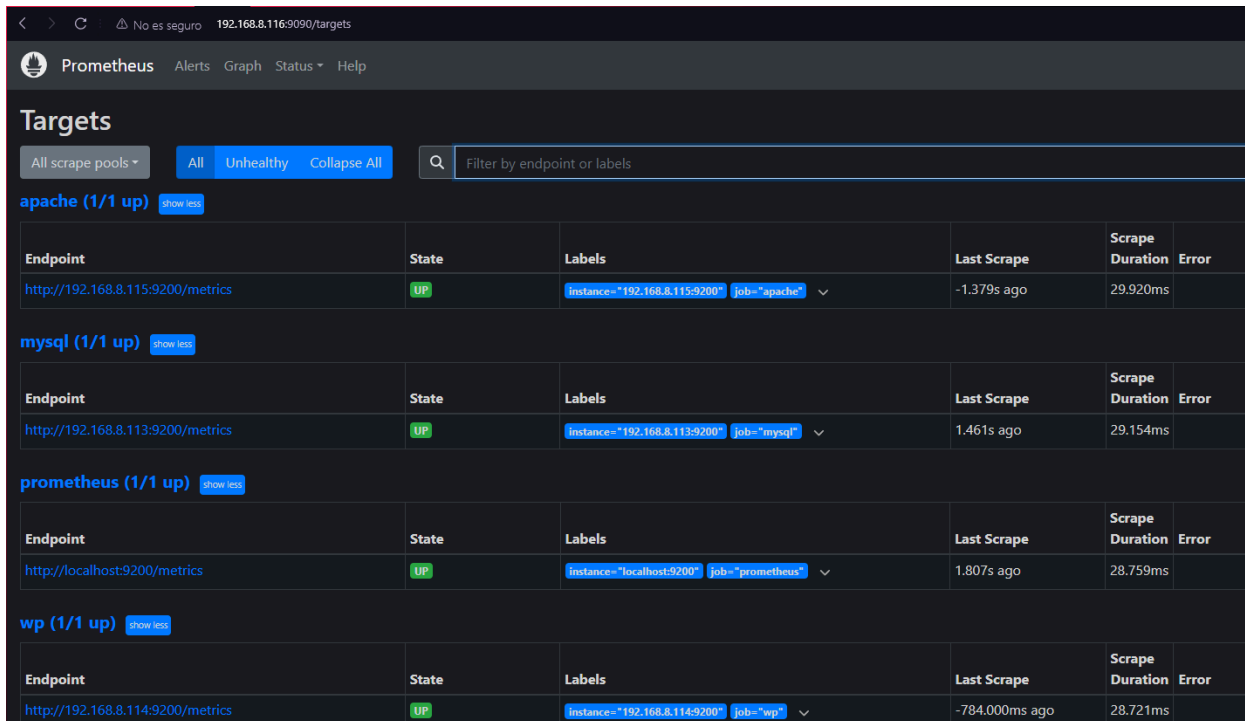
< > ↺ : ⚠ No es seguro 192.168.8.115:9200/metrics

# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 1.8081e-05
go_gc_duration_seconds{quantile="0.25"} 3.1011e-05
go_gc_duration_seconds{quantile="0.5"} 3.3041e-05
go_gc_duration_seconds{quantile="0.75"} 3.4997e-05
go_gc_duration_seconds{quantile="1"} 6.3948e-05
go_gc_duration_seconds_sum 0.004170845
go_gc_duration_seconds_count 126
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.22.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.403952e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 2.30484968e+08
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.503346e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 3.170022e+06
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 3.124448e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.403952e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 3.514368e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 4.366336e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 19033
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge

```

Esta serie de datos que se refrescan automáticamente, son las métricas del sistema que utiliza Prometheus para monitorearlos. Como no hemos configurado Prometheus junto a Grafana, no se van a pintar de forma gráfica. Sin embargo, lo que sí trae implementado de base Prometheus es el detector de estado de las máquinas, y hemos configurado las direcciones IP que queremos que revise.

Para verlo, accedemos a la IP del servicio Prometheus, puerto 9090 y URI /targets



The screenshot shows the Prometheus web interface at the /targets endpoint. The page displays four scrape pools, each with a table of targets. All targets are in an 'UP' state.

| Targets | | | | | |
|--|-------|--|----------------|-----------------|-------|
| All scrape pools ▾ All Unhealthy Collapse All 🔍 Filter by endpoint or labels | | | | | |
| apache (1/1 up) show less | | | | | |
| Endpoint | State | Labels | Last Scrape | Scrape Duration | Error |
| http://192.168.8.115:9200/metrics | UP | instance="192.168.8.115:9200" job="apache" ▾ | -1.379s ago | 29.920ms | |
| mysql (1/1 up) show less | | | | | |
| Endpoint | State | Labels | Last Scrape | Scrape Duration | Error |
| http://192.168.8.113:9200/metrics | UP | instance="192.168.8.113:9200" job="mysql" ▾ | 1.461s ago | 29.154ms | |
| prometheus (1/1 up) show less | | | | | |
| Endpoint | State | Labels | Last Scrape | Scrape Duration | Error |
| http://localhost:9200/metrics | UP | instance="localhost:9200" job="prometheus" ▾ | 1.807s ago | 28.759ms | |
| wp (1/1 up) show less | | | | | |
| Endpoint | State | Labels | Last Scrape | Scrape Duration | Error |
| http://192.168.8.114:9200/metrics | UP | instance="192.168.8.114:9200" job="wp" ▾ | -784.000ms ago | 28.721ms | |

Aquí vemos que todos los servicios se encuentran activos

5. ANÁLISIS ECONÓMICO

El equipo que se ha utilizado para el desarrollo del proyecto tiene las siguientes características:

| | |
|--|---------|
| Intel(R) Core(TM) i5-2400 | 40€ |
| Gigabyte H510M H V2 | 68.99€ |
| Kingston FURY Beast DDR4 3200 MHz 8GB CL16 | 24€ |
| Tacens Imperator II USB 3.0 Negra | 37.99€ |
| Tempest PSU 750W Fuente de Alimentación | 50.99€ |
| Patriot P210 2.5" SSD 128GB SATA 3 | 18.28€ |
| Subtotal | 240.28€ |

Con este presupuesto, nos alcanza de sobra para poder lanzar este proyecto, y utilizarlo como servidor de producción. A este presupuesto se le añadiría la mano de obra, si cobramos 20 euros/hora de mano de obra sin tener que desplazarnos, y añadimos soporte del servidor durante un año:

| | |
|---|-----------------|
| Preparación del servidor y puesta en marcha | 40€ |
| Ejecución del proyecto | 20€ |
| Soporte y gestión del servidor | 12 x 50€ = 600€ |
| Subtotal | 900.28€ |
| Total (21% IVA) | 1089.34€ |

Otra solución de implantación que podemos ofrecer es un VPS contratado con alguna empresa externa, si el cliente no quiere tener un servidor físico. Se desglosa a continuación un presupuesto donde contrataremos un servidor con Contabo, ofreciendo soporte durante 12 meses:

Cloud VPS 1

Monthly Base Price €4.50

50% off Location Fee in IN

| CPU | RAM | STORAGE | SNAPSHOT |
|--------------|----------|------------------------------|------------|
| 4 vCPU Cores | 6 GB RAM | 100 GB NVMe or 400 GB SSD | 1 Snapshot |

1. Select your term length

12 Months

2. Region

☒ European Union

Good Latency 80 ms

☐ United States (Central)

Good Latency 134 ms

€0.90

☐ United Kingdom

Good Latency 81 ms

€0.90

Order Summary

Cloud VPS 1

Server Quantity: 1

Details

- Contract Period: 12 Months
- European Union
- 400 GB SSD
- Use your existing Custom Image Storage
- 32 TB Out + Unlimited In

50% off Location Fee in IN

Monthly €4.50
€5.45 incl. 21% VAT

One-Time €0.00
Setup Fee

Due Today €54.00
Prepaid (12 months)
€65.34 incl. 21% VAT

| | |
|--------------------------------|-----------------|
| Servidor | 54€ |
| Configuración de Proxmox | 40€ |
| Ejecución del proyecto | 20€ |
| Soporte y gestión del servidor | 12 x 50€ = 600€ |
| Subtotal | 714€ |
| Total (21% IVA) | 863.94€ |

6. SEGUIMIENTO Y CONTROL

POSIBLES INCIDENCIAS

- Problemas con las redes: cuando se despliega el proyecto, se definen en los contenedores una serie de IP que tendremos que definir manualmente cada vez que lo implementemos.
- Tráfico: la red del cliente debe estar preparada para soportar el tráfico que entre al wordpress que nosotros implementamos, además de tener una IP pública
- Firewall: en el proyecto no se ha contemplado que el firewall del cliente bloquee las peticiones que le lleguen a los puertos que exponemos en el servidor
- Desarrollo: en este proyecto el wordpress queda instalado, pero de forma mínima. No se contempla en este proyecto ser desarrolladores de la web que el cliente pide

SOLUCIONES

- Al inicializar cada proyecto, debemos cambiar estas IP manualmente en cada fichero. Una medida que suavizará este problema sería crear variables en el código, de forma que no tendríamos que modificar estos parámetros varias veces en cada archivo
- El día de inicialización del proyecto se investigará el router y la velocidad de red del cliente para comprobar la viabilidad con la red actual o proponer un cambio al cliente. Se abrirán los puertos necesarios (80 y 443) en el router del cliente. Los puertos de monitorización no es necesario examinarlos desde la red externa. En caso necesario, se puede explorar la posibilidad de abrir una VPN contra esta red.
- El desarrollo no se explora en el proyecto, si bien el cliente quiere contratar este servicio, se contemplaría la viabilidad y se añadiría a presupuesto.

ANÁLISIS DAFO

- Debilidades:

- Inicio Complejo: Configurar todo inicialmente, especialmente SaltStack y su integración con Proxmox, fue complicado y requiere aprender mucho, lo que puede ser frustrante y exigente para no iniciados.
- Necesidad de Experiencia Técnica: El éxito del proyecto depende de tener personas con conocimientos avanzados en las herramientas que usamos, como SaltStack y Proxmox, lo que puede ser un obstáculo si no tenemos a las personas adecuadas en el equipo.
- Mantenimiento Constante: Mantener y actualizar las configuraciones y scripts es una tarea continua que puede requerir bastante trabajo, lo que puede ser abrumador y tedioso.

- Amenazas:

- Riesgos de Seguridad: Los ataques y vulnerabilidades pueden ser una amenaza importante, especialmente en entornos automatizados, lo que me puede hacer sentir inseguro y preocupado.
- Resistencia al Cambio: Implementar nuevas tecnologías y procesos puede encontrar resistencia en el personal, lo que podría retrasar o complicar el proyecto en su implementación.
- Competencia: Otras empresas pueden adoptar soluciones similares, reduciendo nuestra ventaja competitiva.

- Fortalezas:
 - Ahorro de Tiempo y Esfuerzo: Al usar SaltStack para automatizar tareas repetitivas, se reduce el tiempo y el esfuerzo necesarios para gestionar los servicios, minimizando también los errores humanos.
 - Crecimiento Fácil: Con Proxmox, la infraestructura puede crecer junto con la empresa sin grandes complicaciones, lo que permite adaptación a los cambios y crecer junto con la empresa.
 - Adaptabilidad: Definir la infraestructura como código nos permite cambiar y ajustar la configuración rápidamente según las necesidades del negocio.
 - Ahorro en Costos: Al optar por software libre y herramientas de código abierto, se ahorra mucho en licencias y mantenimiento.
 - Colaboración Abierta: Publicar el código en GitHub invita a otros a colaborar, aprender y mejorar el proyecto, creando una comunidad de apoyo.
- Oportunidades:
 - Innovación y Mejora Competitiva: Usar estas soluciones avanzadas puede poner a las PYMEs y autónomos a la vanguardia tecnológica, mejorando su posición en el mercado.
 - Facilidad para Integrar Nuevas Tecnologías: La modularidad y automatización facilitan la incorporación de nuevas tecnologías y servicios en el futuro.
 - Apoyo Comunitario: Participar en la comunidad de software libre nos da acceso a recursos adicionales, soporte y mejores prácticas.

7. FUENTES DE DOCUMENTACIÓN

Prometheus. GitHub - prometheus/prometheus: The Prometheus monitoring system and time series database. GitHub. <https://github.com/prometheus/prometheus>.

Salt Module Index. <https://docs.saltproject.io/en/latest/py-modindex.html>.

Devopscube. How to install and configure Prometheus Monitoring System on Linux. *DevopsCube*. diciembre 2023. <https://devopscube.com/install-configure-prometheus-linux/>.

Developer Portal | Couchbase.
<https://developer.couchbase.com/tutorial-node-exporter-setup>.

Proxmox Guide. Proxmox VE Administration Guide.
<https://pve.proxmox.com/pve-docs/pve-admin-guide.html>.

Tezer OS. How To Use Apache HTTP Server As Reverse-Proxy Using mod_proxy Extension. DigitalOcean.
https://www.digitalocean.com/community/tutorials/how-to-use-apache-http-server-as-reverse-proxy-using-mod_proxy-extension. Published 14 de febrero de 2014.

Yang K, Heidi E. How To Install Linux, Apache, MySQL, PHP (LAMP) Stack on Ubuntu. DigitalOcean.
<https://www.digitalocean.com/community/tutorials/how-to-install-lamp-stack-on-ubuntu>.

Published 27 de febrero de 2024.

ANEXO I GUÍA DE CLONACIÓN DEL REPOSITORIO

En el servidor con salt instalado y preparado, cambiar de directorio a /srv/salt

Clonar el repositorio con git clone

```
root@pve-tfg:~# cd /srv/salt/  
root@pve-tfg:/srv/salt# git clone https://github.com/xsk-sys/wp-automation.git
```

Con el repositorio clonado, tenemos los states que se han desarrollado

```
root@pve-tfg:/srv/salt# tree  
.  
├── wp-automation  
│   ├── files  
│   │   ├── dirmod.conf  
│   │   ├── my.cnf  
│   │   ├── node_exporter.service  
│   │   ├── prometheus.service  
│   │   ├── wp-apache.conf  
│   │   ├── wp-config.php  
│   │   └── wp-endpoint.conf  
│   ├── LICENSE  
│   ├── orchestrate  
│   │   ├── deploy.sls  
│   │   ├── init.sls  
│   │   ├── main.sls  
│   │   └── node_exporter.sls  
│   ├── README.md  
│   └── states  
│       ├── apache_init.sls  
│       ├── ct_init.sls  
│       ├── delete_ct.sls  
│       ├── mysql.sls  
│       ├── node_exporter_install.sls  
│       ├── ping.sls  
│       ├── prometheus.sls  
│       ├── rm_saltkeys.sls  
│       ├── salt_install.sls  
│       └── wp_init.sls  
5 directories, 23 files  
root@pve-tfg:/srv/salt#
```

ANEXO II GUÍA DE APLICACIÓN DEL PROYECTO

Con el servidor preparado y el repositorio clonado en el servidor del cliente, los pasos a seguir para aplicar el proyecto son los siguientes:

Usar el comando `salt-run state.orchestrate wp-automation. orchestrate.init` para crear los contenedores y preparar el servicio salt-minion

```
Summary for pve-tfg
-----
Succeeded: 9 (changed=9)
Failed:    0
-----
Total states run:    9
Total run time: 227.226 s

Summary for pve-tfg_master
-----
Succeeded: 2 (changed=2)
Failed:    0
-----
Total states run:    2
Total run time: 278.192 s
root@pve-tfg:/srv/salt/wp-automation#
```

Cuando el comando devuelve una salida exitosa, usar el comando

`salt-run state.orchestrate wp-automation. orchestrate.main` que ejecutará el resto de acciones que preparan los contenedores finales

```
Summary for prometheus
-----
Succeeded: 7 (changed=7)
Failed:    0
-----
Total states run:    7
Total run time: 5.472 s

Summary for pve-tfg_master
-----
Succeeded: 9 (changed=9)
Failed:    0
-----
Total states run:    9
Total run time: 97.056 s
root@pve-tfg:/srv/salt/wp-automation#
```

ANEXO III GUÍA DE BORRADO DE CONTENEDORES

Usar el comando `salt pve-tfg state.apply wp-automation.states.delete_ct` para borrar los contenedores automáticamente

```
Summary for pve-tfg
-----
Succeeded: 4 (changed=4)
Failed:    0
-----
Total states run:    4
Total run time: 16.863 s
```

Cuando el comando devuelve una salida exitosa, usar el comando

`salt-run state.orchestrate wp-automation._orchestrate.main` borrará las claves que conectaban los antiguos salt-minion con el master