Introducción a Terraform y Salt Project

Unidad 05. Integración de Terraform y Salt Project

Autor: Sergi García

Actualizado Noviembre 2025

Licencia

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 **Importante**

❕ **Atención**

💬 **Interesante**

ÍNDICE

Unidad 05. Integración de Terraform y Salt Project

1. 📘 Introducción a la automatización de configuración

La **Infraestructura como Código (IaC)** es un enfoque que permite **definir y gestionar entornos de infraestructura mediante archivos de código**, en lugar de hacerlo manualmente.
Esto significa que servidores, redes, contenedores y configuraciones pueden describirse en archivos de texto, versionarse y reproducirse en cualquier entorno.

Terraform, **Salt**, **Ansible** o **Puppet** son herramientas que implementan esta filosofía.
Cada una cumple un propósito dentro del ciclo DevOps.

💬 **Ejemplo:**
Un archivo Terraform (main.tf) puede describir una red y varios contenedores.
Un archivo Salt (users.sls) puede definir los usuarios y servicios que deben configurarse dentro de esos contenedores.

📖 **Importante:** IaC no solo implica "automatizar", sino hacerlo de forma **declarativa**: el código describe *el estado deseado*, y la herramienta se encarga de llegar hasta él.

1.1. Diferencias entre aprovisionamiento y configuración

Concepto	Herramienta típica	Responsabilidad
Aprovisionamiento	Terraform	Crear recursos (máquinas, redes, contenedores, etc.)
Configuración	Salt, Ansible, Chef	Configurar el software dentro de esos recursos

- **Terraform** responde a: "¿Qué infraestructura necesito?"
    - Ejemplo: Terraform crea dos servidores Ubuntu.
- **Salt** responde a: "¿Cómo debe estar configurada una vez creada?"
    - Salt se encarga de instalar Apache, crear usuarios y habilitar servicios.

1.2. Cómo encajan Terraform y Salt en el ciclo DevOps

En un flujo de **Infraestructura como Código**, Terraform y Salt se complementan perfectamente:

1. **Terraform** despliega la infraestructura base (nodos, redes, volúmenes).
2. **Salt Project** entra en acción para configurar y mantener el estado interno de los sistemas.
3. Juntos forman una **cadena declarativa continua**, reproducible y controlada por código.

📖 **Importante:**
Terraform se centra en el *entorno* (el "dónde"),
Salt se centra en el *comportamiento* (el "qué").

❕ **Atención:** Aunque pueden funcionar de forma independiente, **usar ambos en conjunto mejora la eficiencia** y asegura que tanto la infraestructura como la configuración estén alineadas.

2. ⚙️¿Qué es Salt Project?

2.1. Historia y origen

Salt Project (conocido originalmente como **SaltStack**) nació en 2011 como una herramienta de **automatización rápida, flexible y escalable**.
Su creador, **Thomas S. Hatch**, buscaba una alternativa más eficiente que las soluciones existentes en aquel momento.

En 2020, **VMware** adquirió SaltStack y lo incorporó a su suite de automatización, manteniéndolo como un proyecto **open source** bajo el paraguas de la **comunidad Salt Project**.

💬 **Interesante:**
El nombre *Salt* proviene de su objetivo original: *"Spicy Automation and Lightning-fast Technology"*, destacando su velocidad y capacidad de orquestación.

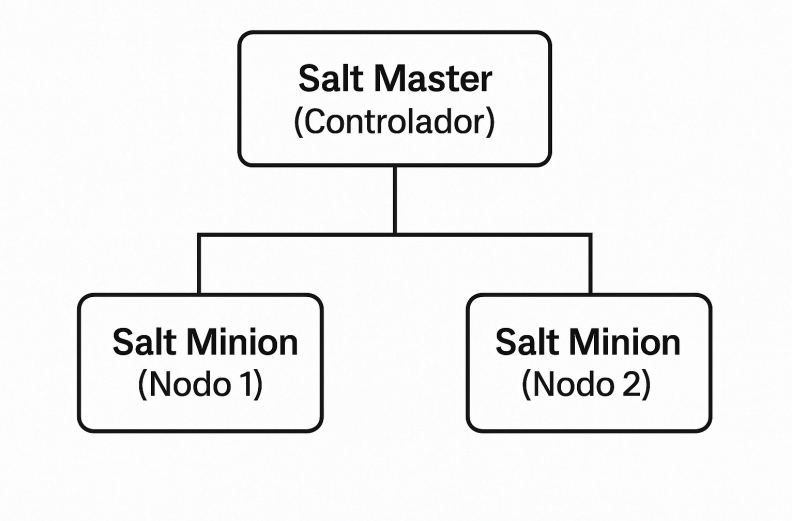
2.2. Arquitectura general

Salt utiliza una arquitectura **Maestro–Minion** (Master–Minion), basada en un modelo cliente-servidor:

- **Salt Master:** nodo central que define configuraciones y envía órdenes.
- **Salt Minion:** nodos gestionados que reciben y ejecutan las órdenes.
- **ZeroMQ:** sistema de mensajería que permite comunicación en tiempo real.

📖 **Importante:** El Master y los Minions intercambian mensajes cifrados mediante claves públicas y privadas, garantizando seguridad y confianza.

💬 **Ejemplo gráfico:**



2.3. Componentes principales

Componente	Descripción	Ejemplo de uso
Salt Master	Nodo central que coordina la infraestructura y almacena estados	Enviar salt '*' test.ping
Salt Minion	Cliente que ejecuta órdenes del Master	Instalar paquetes, crear usuarios
Salt States	Archivos YAML (.sls) que definen configuraciones deseadas	users.sls crea usuarios
Pillars	Variables seguras (datos sensibles o específicos por minion)	Contraseñas o claves API
Grains	Información sobre cada minion (SO, IP, roles)	salt '*' grains.items
Runners/Reactors	Scripts de orquestación y automatización de eventos	Automatizar acciones según logs o alertas

2.4. Comunicación entre nodos

Salt usa **ZeroMQ**, una librería de mensajería ultrarrápida, para establecer la comunicación entre Master y Minions.
Los mensajes se cifran con claves **RSA** y se validan mediante un proceso de intercambio inicial:

1. El Minion se registra enviando su clave pública.
2. El Master acepta la clave (**comando "salt-key -A"**).
3. Se establece una conexión cifrada permanente.

📖 **Importante:** Esta comunicación permite ejecutar órdenes simultáneamente en cientos o miles de minions con un solo comando.

💬 **Ejemplo de flujo:**

```
salt '*' test.ping
```

1. El Master envía la orden a todos los minions.
2. Cada minion responde con "True" si está activo.
3. Salt muestra los resultados consolidados.

2.5. Ciclo de vida de una orden en Salt

Etapa	Descripción
Etapa 1	El usuario ejecuta un comando desde el Master (salt '*' cmd.run 'uptime')
Etapa 2	El Master envía la orden a los Minions usando ZeroMQ
Etapa 3	Cada Minion ejecuta el comando localmente
Etapa 4	El resultado vuelve al Master
Etapa 5	El Master muestra la salida consolidada en la terminal

Este modelo de comunicación hace que Salt sea **extremadamente rápido** en comparación con herramientas que usan conexiones SSH independientes.

3. 🧠 Conceptos clave del ecosistema Salt

3.1. Salt Master

El **Salt Master** es el **nodo de control central** en la arquitectura de Salt. Se encarga de almacenar configuraciones, distribuir estados y coordinar la comunicación con los minions.

📖 **Importante:** El Master **no ejecuta las configuraciones directamente**, sino que envía las órdenes a los Minions, que las ejecutan localmente.

💬 **Ejemplo:** Desde el Master se puede ejecutar:

```
salt '*' test.ping
```

y obtener respuesta de todos los minions conectados.

El Master gestiona:

- Los archivos de estado (/srv/salt/).
- Los **Pillars** (datos sensibles o personalizados).
- Las **claves públicas** de los minions.
- Las políticas de orquestación y ejecución.

3.2. Salt Minions

Los **Salt Minions** son los **nodos gestionados** por el Master. Cada minion ejecuta las órdenes que recibe y devuelve los resultados.

El servicio salt-minion se instala en cada nodo controlado.

Durante el primer inicio, el minion:

1. Genera una clave pública.
2. La envía al Master.
3. Espera ser aceptado con el comando "salt-key -A".

📖 **Importante:** Un minion puede tener un nombre (ID) único y personalizable, o usar su hostname por defecto.

💬 **Ejemplo de aceptación de minions:**

```
salt-key -L        # Lista claves pendientes
salt-key -A        # Acepta todas
salt '*' test.ping # Comprueba comunicación
```

3.3. Salt Keys (Gestión de confianza)

Salt utiliza un **sistema de claves RSA** para autenticar la comunicación entre Master y Minions. Esto garantiza que solo los minions autorizados pueden recibir órdenes.

Comando	Descripción
salt-key -L	Lista claves aceptadas y pendientes
salt-key -A	Acepta todas las claves pendientes
salt-key -D minion1	Elimina la clave del minion1
salt-key -R	Rechaza claves no autorizadas

❗ **Atención:** Si un minion es comprometido, se debe **revocar su clave** inmediatamente en el Master para impedir su acceso.

3.4. Top Files (top.sls)

El archivo top.sls define qué estados se aplican a qué minions. Se considera el punto de entrada principal de la configuración en Salt.

💬 **Ejemplo:**

```
base:
  'salt-minion1':
    - users
  'salt-minion2':
    - novnc
```

📖 **Importante:**

- base: define el entorno por defecto.
- Cada minion o grupo recibe una lista de módulos de estado (.sls) a aplicar.
- Salt buscará esos archivos dentro de /srv/salt.

3.5. Módulos Salt

Salt ofrece una colección de **módulos de ejecución** que encapsulan acciones específicas.
Estos módulos permiten ejecutar operaciones remotas o aplicar configuraciones.

Módulo	Función	Ejemplo
cmd.run	Ejecuta comandos en el sistema	salt '*' cmd.run 'ls /'
pkg.installed	Instala paquetes	salt '*' pkg.installed nginx
service.running	Inicia un servicio	salt '*' service.running nginx
file.managed	Gestiona archivos	salt '*' file.managed /etc/motd
user.present	Crea usuarios	salt '*' user.present name=jose

💬 **Interesante:**
Cada módulo tiene una estructura consistente:

```
<modulo>.<función> [parámetros]
```

Ejemplo:

```
salt '*' pkg.install nginx
```

3.6. Idempotencia y convergencia

Salt, al igual que Terraform, trabaja bajo el principio de **idempotencia**: Aplicar el mismo estado varias veces no debe cambiar el sistema si ya cumple con la configuración deseada.

📖 **Ejemplo:** Si un estado indica que el paquete nginx debe estar instalado, Salt comprobará:

- Si ya lo está → no hace nada.
- Si no → lo instala.

Esto garantiza **consistencia**, **seguridad** y **predictibilidad** en la configuración.

4. 🧰Salt States (Declaración de configuración)

4.1. ¿Qué son los Salt States?

Los Salt States son archivos escritos en YAML con extensión .sls (Salt State File).
Describen el estado deseado de un sistema, no los pasos para alcanzarlo.

📖 **Importante:** Salt States funcionan de manera **declarativa**, no imperativa:
se define *qué se quiere*, no *cómo hacerlo*.

💬 **Ejemplo básico (users.sls):**

```
user_jose:
  user.present:
    - name: jose
    - shell: /bin/bash
```

Este estado garantiza que exista un usuario jose con la shell /bin/bash. Si no existe, Salt lo crea; si ya está, no hace nada.

4.2. Sintaxis y estructura

Un archivo .sls tiene tres partes esenciales:

Sección	Descripción	Ejemplo
ID	Nombre interno del recurso	nginx
Módulo.Función	Acción declarada	pkg.installed
Parámetros	Configuración específica	- name: nginx

💬 **Ejemplo completo (nginx.sls):**

```
nginx:
  pkg.installed:
    - name: nginx
  service.running:
    - enable: True
```

📖 **Importante:** Este archivo asegura que:

- El paquete nginx esté instalado
- El servicio nginx esté en ejecución y habilitado al inicio

4.3. Ejemplo práctico: creación de usuarios

Salt facilita la gestión de usuarios con el módulo user.present.

💬 **Ejemplo (users.sls):**

```
user_maria:
  user.present:
    - name: maria
    - fullname: "Maria Pérez"
    - shell: /bin/bash
```

Cuando se ejecuta:

```
salt '*' state.apply users
```

Salt realiza en esa orden:

1. Verifica si el usuario existe.
2. Si no, lo crea con los parámetros definidos.
3. Informa el resultado en pantalla.

4.4. Jerarquía de directorios /srv/salt

Por defecto, el Salt Master busca los archivos de estado en el directorio /srv/salt/.

📁 **Estructura típica:**

/srv/salt/
├── top.sls
├── users.sls
└── web/
└── nginx.sls

Cada subcarpeta puede representar un módulo o conjunto lógico de estados.

📖 **Importante:** El archivo top.sls actúa como mapa maestro que indica qué estados aplicar a cada minion.

4.5. Aplicar estados desde el Master

Una vez definidos los archivos .sls, se pueden aplicar desde el Master:

```
salt '*' state.apply users
```

💬 **Ejemplo multiple:** Aplicar múltiples estados:

```
salt '*' state.apply 'users,novnc'
```

Salt descargará y ejecutará los estados, mostrando un resumen con colores:

- **Verde:** sin cambios (ya cumple el estado).
- **Amarillo:** se aplicaron cambios.
- **Rojo:** hubo errores.

❕ **Atención:** Si un estado tiene dependencias (por ejemplo, crear un archivo antes de arrancar un servicio), Salt las gestiona automáticamente siguiendo el orden lógico.

4.6. Beneficios de usar Salt States

✔️ Reproducibilidad total del entorno.
✔️ Configuraciones versionables y auditables.
✔️ Menor riesgo de errores humanos.
✔️ Integración directa con Terraform, Ansible o CI/CD.

💬 **Interesante:** Los Salt States pueden compartirse en repositorios Git y reutilizarse en distintos proyectos, siguiendo el enfoque **GitOps**.

5. 🔐PILLARS, GRAINS Y TARGETING

5.1. Pillars

Los **Pillars** en Salt son **estructuras de datos seguras y jerárquicas** utilizadas para almacenar **información sensible o específica por minion**.
Funcionan de manera similar a las variables, pero se mantienen **en el lado del Master** y no se distribuyen a todos los nodos, solo a los que corresponda.

📖 **Importante:**
Los Pillars se renderizan en el Master y se envían de forma cifrada únicamente al minion autorizado.
Esto los hace ideales para guardar contraseñas, claves API, configuraciones privadas, etc.

💬 **Ejemplo básico (/srv/pillar/users.sls):**

```
user_passwords:
  jose: "clave_supersecreta"
  maria: "clave_oculta"
```

Y el archivo top.sls correspondiente:

```
base:
  '*':
    - users
```

Para acceder al valor dentro de un estado:

```
user_jose:
  user.present:
    - name: jose
    - password: {{ pillar['user_passwords']['jose'] }}
```

💬 **Interesante:** Gracias a los Pillars, es posible separar **código y datos confidenciales**, cumpliendo buenas prácticas de seguridad.

5.2. Grains

Los **Grains** son **metadatos estáticos o dinámicos** sobre cada minion. Incluyen información del sistema operativo, kernel, CPU, IP, hostname, roles personalizados, etc.

📖 **Ejemplo de consulta:**

```
salt '*' grains.items
```

Resultado típico:

```
os: Ubuntu
osrelease: 22.04
ip_interfaces:
  eth0:
    - 172.18.0.2
```

Los Grains se utilizan para:

- Identificar sistemas
- Crear condiciones en los estados
- Aplicar configuraciones específicas según el entorno

💬 **Ejemplo de uso en un estado:**

```
install_tools:
  pkg.installed:
    - names:
      {% if grains['os'] == 'Ubuntu' %}
      - htop
      - vim
      {% else %}
      - nano
      {% endif %}
```

❕ **Atención:** Puedes definir Grains personalizados en /etc/salt/grains si necesitas clasificar los minions por roles o entornos:

```
role: webserver
environment: production
```

5.3. Targeting (Selección de minions)

Targeting permite **filtrar qué minions recibirán una orden o estado**. Es esencial para ejecutar configuraciones específicas donde se requiera sin afectar a toda la infraestructura.

Tipo de Targeting	Ejemplo	Descripción
Por ID	*salt 'minion1' test.ping*	Selecciona por nombre exacto
Por patrón (glob)	*salt 'minion*' test.ping*	Coincidencia de nombres
Por lista	*salt -L 'minion1,minion2' test.ping*	Especificar varios nodos
Por Grains	*salt -G 'os:Ubuntu' test.ping*	Filtrar por sistema operativo
Por expresiones regulares	*salt -E 'minion[1-2]' test.ping*	Selección avanzada
Por compound matchers	*salt -C 'G@os:Ubuntu and not minion2' test.ping*	Combinaciones lógicas

💬 **Ejemplo práctico:**

```
salt -G 'role:webserver' state.apply webserver
```

Este comando aplicará el estado webserver.sls solo a los minions que tengan el grain role: webserver.

5.4. Ejemplo completo

Objetivo: Aplicar un estado de instalación de utilidades solo a los servidores Ubuntu.

```
salt -G 'os:Ubuntu' state.apply tools
```

Este comando:

- Busca los minions con os:Ubuntu
- Les aplica el archivo /srv/salt/tools.sls
- Deja intactos los demás sistemas

💬 **Interesante:** El sistema de targeting de Salt es uno de los más potentes dentro del ecosistema de automatización, permitiendo combinaciones flexibles entre **nombres, grains y expresiones**.

6. 🧩Salt Orchestration (Orquestación multinodo)

6.1. Qué es la orquestación en Salt

La **Orchestración en Salt** es una capa avanzada que permite **coordinar acciones en múltiples minions** desde el Master, siguiendo un orden y lógica definidos.

📖 **Importante:** Mientras los **Salt States** configuran un sistema individual, la **Orchestración** permite coordinar *varios sistemas* en tareas distribuidas.

💬 **Ejemplo:**

- Crear una base de datos en un minion.
- Configurar un servidor web en otro.
- Asegurar que ambos estén sincronizados.

6.2. Archivos de orquestación (/srv/salt/orch/*.sls)

Los archivos de orquestación son similares a los `.sls` normales, pero se ejecutan desde el **Master** y pueden controlar múltiples hosts.

📄 **Ejemplo (/srv/salt/orch/deploy_app.sls):**

```
deploy_database:
  salt.state:
    - tgt: 'minion-db'
    - sls: mysql

deploy_web:
  salt.state:
    - tgt: 'minion-web'
    - sls: nginx
    - require:
      - salt: deploy_database
```

💬 **Explicación:**

1. Primero se aplica el estado mysql en minion-db.
2. Luego, una vez finalizado correctamente, se aplica nginx en minion-web.

📖 **Importante**: El orden lo define el parámetro require, garantizando dependencias controladas entre nodos.

6.3. Ejecución de orquestación

Para lanzar una orquestación desde el Master:

```
salt-run state.orchestrate orch.deploy_app
```

Salida esperada:

```
----------
          ID: deploy_database
    Function: salt.state
      Result: True
     Comment: States applied successfully
----------
          ID: deploy_web
    Function: salt.state
      Result: True
     Comment: Nginx deployed after database setup
```

💬 **Interesante:** Los runners (salt-run) pueden integrarse con **cron jobs**, **eventos** o **pipelines CI/CD**, convirtiendo Salt en un **motor de orquestación automatizada**.

6.4. Integración con Terraform

Terraform y Salt se integran perfectamente dentro de un flujo automatizado.
Terraform **crea** los recursos, y Salt **los configura y orquesta**.

📄 **Ejemplo (Terraform con Salt Runner):**

```
provisioner "local-exec" {
  command = "salt-run state.orchestrate orch.deploy_app"
}
```

Esto permite que, después de un terraform apply, se ejecuten automáticamente los estados definidos en Salt.

📖 **Importante:** Esta integración permite pasar **de infraestructura a configuración** sin intervención manual, cerrando el ciclo DevOps completo.

7. 🧑‍💻 Ejecución de comandos y gestión remota

7.1. Comandos principales

Una de las características más potentes de Salt Project es la capacidad de **ejecutar comandos remotos simultáneamente en cientos o miles de sistemas** gestionados.

Todos los comandos se envían desde el Salt Master mediante la utilidad salt.

💬 **Ejemplo básico:**

```
salt '*' test.ping
```

👉 Este comando verifica la comunicación entre el Master y todos los Minions. Cada uno responde con True si está activo.

📦 **Comandos más comunes**

Comando	Descripción	Ejemplo
test.ping	Comprueba conectividad con el Master	*salt '*' test.ping*
cmd.run	Ejecuta un comando del sistema	*salt '*' cmd.run 'uptime'*
pkg.install	Instala un paquete	*salt '*' pkg.install nginx*
service.status	Verifica el estado de un servicio	*salt '*' service.status ssh*
file.read	Muestra el contenido de un archivo	*salt '*' file.read /etc/hostname*

📖 **Importante:** Salt no necesita conectarse por SSH a cada servidor. Usa su propia red de mensajería (ZeroMQ), mucho más rápida y escalable.

7.2. Salt Call: ejecución local

El comando salt-call se utiliza cuando no hay un Master disponible, o se quiere probar la configuración directamente en el Minion.

💬 **Ejemplo:** En este caso, el Minion ejecuta la orden localmente y devuelve el resultado.

```
salt-call test.ping
```

📖 **Importante:** Este modo es ideal para entornos de prueba o desarrollo, ya que permite aplicar estados localmente:

```
salt-call state.apply users
```

❕ Atención: En modo local, el minion no recibe archivos desde un Master. Debes tener los estados .sls disponibles en su propio sistema.

7.3. Salt Run: runners y tareas de control

Salt Run ejecuta runners, que son módulos especiales del Master usados para:

- Orquestar despliegues (state.orchestrate)
- Consultar información del sistema (manage.up, jobs.list_jobs)
- Ejecutar comandos administrativos globales

💬 **Ejemplo:**

```
salt-run manage.up
```

📖 **Resultado:** muestra todos los minions activos.

Otro ejemplo con orquestación:

```
salt-run state.orchestrate orch.deploy_app
```

7.4. Jobs y control de tareas

Cada comando ejecutado genera un **Job ID (JID)** que permite revisar resultados o ejecutar tareas asíncronas.

💬 **Ejemplo:**

```
salt '*' cmd.run 'uptime' --async
```

Salt devuelve un JID único:

```
Executed command with job ID: 20251016122000123456
```

Puedes consultar su progreso:

```
salt-run jobs.lookup_jid 20251016122000123456
```

📖 **Importante:** Este sistema de *jobs* permite ejecutar tareas en segundo plano, muy útil para operaciones largas o programadas.

7.5. Salt Event Bus y Reactor

El **Event Bus** de Salt es un canal interno de eventos que permite **reaccionar automáticamente a cambios en el sistema**.

💬 **Ejemplo de flujo reactivo:**

- Un minion envía un evento al iniciar.
- El Master detecta el evento.
- El **Reactor** ejecuta un estado o script en respuesta.

📄 **Ejemplo (/etc/salt/master.d/reactor.conf):**

```
reactor:
  - 'minion_start':
    - /srv/reactor/welcome.sls
```

📄 **Archivo de reacción (/srv/reactor/welcome.sls):**

```
welcome_event:
  cmd.run:
    - tgt: {{ data['id'] }}
    - arg:
      - "echo 'Bienvenido {{ data['id'] }} al sistema Salt!' > /tmp/welcome.log"
```

📖 **Importante:** Con el sistema Reactor, Salt se convierte en una herramienta **proactiva**, capaz de ejecutar tareas en respuesta a eventos sin intervención humana.

8. 🧱 Integración de Salt con Terraform

8.1. Por qué combinar Terraform y Salt

Terraform y Salt cumplen **roles distintos pero complementarios** dentro de la automatización:

Herramienta	Rol principal	Ejemplo de uso
Terraform	Aprovisionamiento (infraestructura)	Crear VMs, contenedores, redes
Salt	Configuración y orquestación	Instalar software, crear usuarios, aplicar políticas

📖 **Importante:** Terraform responde a "¿qué infraestructura quiero tener?"
Salt responde a "¿cómo debe comportarse esa infraestructura?"

💬 **Ejemplo combinado:**

1. Terraform crea 3 contenedores Ubuntu.
2. Salt instala Nginx, crea usuarios y habilita servicios dentro.

Resultado: **infraestructura + configuración = entorno funcional completo.**

8.2. Flujo de trabajo típico

Un flujo **Terraform + Salt** suele seguir los siguientes pasos:

1. **Terraform**
    a. Despliega los recursos (redes, máquinas, contenedores).
    b. Define las variables de conexión necesarias (IP, hostnames, etc.).
2. **Salt**
    a. Configura los sistemas recién creados.
    b. Aplica los estados (state.apply) y orquestaciones (state.orchestrate).
    c. Gestiona dependencias entre nodos (web ↔ db, etc.).
3. **Automatización combinada**
    a. Terraform puede invocar comandos Salt automáticamente con **provisioners**.

8.3. Métodos de integración entre Terraform y Salt

🔹 **Opción 1: local-exec provisioner**

Permite ejecutar un comando en el sistema donde se ejecuta Terraform al finalizar el apply.

📄 **Ejemplo:**

```
provisioner "local-exec" {
  command = "salt '*' state.apply users"
}
```

👉 Tras el despliegue, Terraform ejecutará automáticamente los estados de Salt.

🔹 **Opción 2: remote-exec provisioner**

Ejecuta comandos dentro de una instancia remota creada por Terraform.

📄 **Ejemplo:**

```
provisioner "remote-exec" {
  inline = [
    "sudo salt-call state.apply users"
  ]
}
```

💬 **Interesante:**
Ideal para entornos cloud (AWS, Azure, GCP) donde los nodos se crean dinámicamente y Salt puede aplicarse desde dentro.

🔹 **Opción 3: Uso de outputs como Pillars dinámicos**

Terraform puede exportar datos (outputs) que Salt usa como Pillars.

📄 **Ejemplo en Terraform:**

```
output "minion_ips" {
  value = docker_container.salt_minion[*].ip_address
}
```

📄 **Ejemplo en Salt (pillar):**

```
minions_ips: {{ pillar['terraform']['minion_ips'] }}
```

📖 **Importante:** Esta técnica permite que Salt "lea" los resultados de Terraform y actúe dinámicamente según la infraestructura creada.

8.4. Ejemplo práctico: integración completa

💬 **Escenario:** Terraform crea un Salt Master y dos Minions. Salt aplica automáticamente los estados users.sls y novnc.sls.

📄 **main.tf (resumen):**

```
resource "docker_container" "salt_master" {
  name  = "salt-master"
  image = docker_image.salt_master.image_id
  ports {
```

```
    internal = 4505
    external = 4505
  }
  ports {
    internal = 4506
    external = 4506
  }

  provisioner "local-exec" {
    command = "salt '*' state.apply users && salt '*' state.apply novnc"
  }
}
```

📖 **Resultado esperado:**

- Terraform crea la red, las imágenes y los contenedores.
- Al finalizar, ejecuta los comandos Salt desde el Master.
- Los Minions crean los usuarios y configuran el servicio noVNC automáticamente.

8.4. Beneficios de la integración Terraform + Salt

✔️ Despliegues reproducibles, consistentes y automáticos.
✔️ Configuraciones declarativas y seguras.
✔️ Menor tiempo de puesta en marcha.
✔️ Simplificación del flujo DevOps.

💬 **Interesante:** Terraform y Salt se integran naturalmente en pipelines de **CI/CD**, donde cada *commit* de código puede regenerar infraestructura y aplicar configuraciones sin intervención manual.

9. 📚Webgrafía

- **Terraform Documentation (Official)**
  - 🔗 https://developer.hashicorp.com/terraform/docs
  - Documentación oficial de Terraform: referencia de providers, recursos, variables y módulos.
- **Terraform Language Reference**
  - 🔗 https://developer.hashicorp.com/terraform/language
  - Referencia completa del lenguaje HCL, incluyendo sintaxis, bucles, funciones y estructuras for_each.
- **Salt Project Documentation (Latest)**
  - 🔗 https://docs.saltproject.io/en/latest/
  - Documentación oficial y actualizada del proyecto Salt: instalación, arquitectura, módulos, targeting y ejemplos prácticos.
- **Salt Installation Guide – Ubuntu 24.04 (Broadcom Repository)**
  - 🔗 https://github.com/saltstack/salt-install-guide
  - Guía oficial de instalación de Salt para Ubuntu 24.04 utilizando los repositorios firmados de Broadcom.
- **SaltStack Orchestration Overview**
  - 🔗 https://docs.saltproject.io/en/latest/topics/orchestrate/orchestrate.html

  - Documentación técnica sobre orquestación multinodo, runners y flujos de coordinación entre múltiples minions.
- **Salt States – Declarative Configuration**
  - 🔗 https://docs.saltproject.io/en/latest/topics/tutorials/starting_states.html
  - Tutorial oficial sobre la estructura y aplicación de archivos `.sls`, principios de configuración declarativa e idempotencia.
- **Salt Pillars and Grains**
  - 🔗 https://docs.saltproject.io/en/latest/topics/pillar/index.html
  - 🔗 https://docs.saltproject.io/en/latest/topics/grains/
  - Referencias sobre el uso de Pillars (datos seguros) y Grains (metadatos del sistema) en Salt Project.
- **HashiCorp Community – Terraform + Salt Integration Patterns**
  - 🔗 https://discuss.hashicorp.com/c/terraform/automation/
  - Casos de uso reales y ejemplos de integración entre Terraform y herramientas de configuración como Salt, Ansible o Puppet.
- **Salt Project Blog – Combining Terraform and SaltStack**
  - 🔗 https://saltproject.io/blog/terraform-and-saltstack-integration/
  - Artículo oficial sobre los flujos de trabajo combinados Terraform + Salt y su papel dentro de pipelines DevOps.
- **GitHub: Terraform + SaltStack Examples**
  - 🔗 https://github.com/saltstack/salt-examples/tree/master/integration/terraform
  - Repositorio con ejemplos prácticos de integración entre Terraform y Salt Project en entornos Docker, bare-metal y cloud.