Salt Assignment

1. Setting up the Salt Master in Multipass

Step 1.1 : *mkdir -p /etc/apt/keyrings*

What it does: It creates a new folder named `keyrings` inside the `/etc/apt/` directory.
Why: This directory is the standard location for storing cryptographic keys (`keyrings`) that the APT package manager uses to verify software packages. The `-p` ensures it doesn't complain if the folder already exists

Step 1.2: *curl -fsSL https://packages.broadcom.com/artifactory/api/security/keypair/SaltProjectKey/public | sudo tee /etc/apt/keyrings/salt-archive-keyring.pgp*

What it does: It downloads the public cryptographic key for the Salt Project.
Why: When we download software from the internet, we need to be sure it hasn't been tampered with. The official Salt Project uses this key to digitally sign their software packages. When our computer downloads a Salt package later, it will use this public key to check the signature. If the signatures match, our computer knows the software is genuine and hasn't been corrupted or replaced by a malicious version.

Step 1.3: *curl -fsSL https://github.com/saltstack/salt-install-guide/releases/latest/download/salt.sources | sudo tee /etc/apt/sources.list.d/salt.sources*

What it does: It downloads a small configuration file named `salt.sources`. This file tells APT the exact web address (repository URL) where the Salt Project stores all its installation files.
Why: our computer usually only knows about the main, official software sources (like Canonical or Debian). By adding this new file, we are pointing APT to the Salt Project's private software library so that when we type a command like `sudo apt install salt-master`, APT knows exactly where on the internet to go and get the necessary files

Step 1.4: *sudo apt update*

Step 1.5: *echo 'Package: salt-*
*Pin: version 3006.**
Pin-Priority: 1001' | sudo tee /etc/apt/preferences.d/salt-pin-1001

What it does: This command creates a configuration file that tells our system's package manager (**APT**) to **force the installation of a specific major version of SaltStack**, which is version **3006**.
The Details:

- `Package: salt-*`: This applies the rule to all packages that start with `salt-` (like `salt-master`, `salt-minion`, etc.).
- `Pin: version 3006.*`: This targets any version that starts with `3006.` (e.g., 3006.0, 3006.1, etc.).
- `Pin-Priority: 1001`: In APT, any priority **above 1000** means "install this version **even if it's considered a downgrade**" or "always prefer this version, even over newer ones." This is a way to ensure you get a specific, stable version.

Step 1.6: *sudo apt-get install salt-master*

What it does: This is the core installation command. It instructs the package manager to download and install the main Salt **Master** software package.

Step 1.7: *sudo systemctl enable salt-master && sudo systemctl start salt-master*

Why: we want the Salt Master running 24/7, and we don't want to manually start it after every maintenance or restart.

Step 1.8: *sudo systemctl status salt-master*

What it Does*:* It checks the current state of the `salt-master` service.
Why it's Used: This command is essential for **troubleshooting** and **verification**.

Step 1.9: *hostname -I*

What it Does*:* This command is to get the ip address of the salt master.

```
shambhavi.intern@PP-R7MV533L02 ~ % multipass shell salt-master
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-88-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Nov 28 10:17:10 IST 2025

  System load:             0.0
  Usage of /:              57.7% of 3.80GB
  Memory usage:            21%
  Swap usage:              0%
  Processes:               140
  Users logged in:         0
  IPv4 address for enp0s1: 192.168.64.11
  IPv6 address for enp0s1: fdd1:c079:7aa8:366d:5054:ff:fe27:164a

  => There is 1 zombie process.


Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Fri Nov 28 10:16:04 2025 from 192.168.64.1
ubuntu@salt-master:~$ sudo systemctl enable salt-master && sudo systemctl start salt-master
ubuntu@salt-master:~$ sudo systemctl status salt-master
● salt-master.service - The Salt Master Server
     Loaded: loaded (/usr/lib/systemd/system/salt-master.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-11-28 10:17:08 IST; 39s ago
       Docs: man:salt-master(1)
             file:///usr/share/doc/salt/html/contents.html
             https://docs.saltproject.io/en/latest/contents.html
   Main PID: 647 (/opt/saltstack/)
      Tasks: 31 (limit: 2261)
     Memory: 252.0M (peak: 254.7M)
        CPU: 5.006s
     CGroup: /system.slice/salt-master.service
             ├─647 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master MainProcess"
             ├─885 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master PubServerChannel._publish_daemon"
             ├─886 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master EventPublisher"
             ├─889 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master Maintenance"
             ├─890 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer ReqServer_ProcessManager"
             ├─891 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorkerQueue"
             ├─892 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorker-0"
             ├─893 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorker-1"
             ├─894 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master FileServerUpdate"
             ├─895 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorker-2"
             ├─896 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorker-3"
             └─898 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-master ReqServer MWorker-4"

Nov 28 10:17:07 salt-master systemd[1]: Starting salt-master.service - The Salt Master Server...
Nov 28 10:17:08 salt-master systemd[1]: Started salt-master.service - The Salt Master Server.
ubuntu@salt-master:~$
```

2. Setting up the Salt Minion in Multipass

Follow the exact same steps as described in setting up the Salt minion till step 1.4, and then follow the steps below:

Step 2.1: *sudo apt-get install salt-minion*

What it does: This is the core installation command. It instructs the package manager to download and install the main Salt **Minion software** package.

Step 2.2: *sudo systemctl enable salt-minion && sudo systemctl start salt-minion*

Step 2.3: *sudo systemctl status salt-minion*

```
------
shambhavi.intern@PP-R7MV533L02 ~ % multipass shell minion-01
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.8.0-88-generic aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

 System information as of Fri Nov 28 11:06:36 IST 2025

  System load:             0.12
  Usage of /:              57.6% of 3.80GB
  Memory usage:            39%
  Swap usage:              0%
  Processes:               102
  Users logged in:         0
  IPv4 address for enp0s1: 192.168.64.12
  IPv6 address for enp0s1: fdd1:c079:7aa8:366d:5054:ff:fe1c:2869

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Tue Nov 25 19:49:46 2025 from 192.168.64.1
ubuntu@minion-01:~$ sudo systemctl enable salt-minion && sudo systemctl start salt-minion
ubuntu@minion-01:~$ sudo systemctl status salt-minion
● salt-minion.service - The Salt Minion
     Loaded: loaded (/usr/lib/systemd/system/salt-minion.service; enabled; preset: enabled)
     Active: active (running) since Fri 2025-11-28 11:04:35 IST; 2min 57s ago
       Docs: man:salt-minion(1)
             file:///usr/share/doc/salt/html/contents.html
             https://docs.saltproject.io/en/latest/contents.html
   Main PID: 651 (python3.10)
      Tasks: 7 (limit: 1057)
     Memory: 102.4M (peak: 112.2M)
        CPU: 1.491s
     CGroup: /system.slice/salt-minion.service
             ├─651 /opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion
             └─840 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion MultiMinionProcessManager MinionProcessManager"
```

Step 2.4: *sudo vim /etc/salt/minion*

What it does: This command opens the main configuration file for the Salt Minion using the vim text editor. The configuration file is edited to add the **IP address** or **hostname** of the Salt Master to the master directive (found in step 1.8)

Why: By default, the Minion tries to connect to a Master named Salt. If our Master has a different name or, more commonly, if we are using IP addresses, we must explicitly tell the Minion **where its Master is located** so it can initiate communication.

Step 2.5: *sudo systemctl restart salt-minion*

Step 2.6: *sudo salt-key -L*

What it does: Exit from the salt minion and run this command on the **Salt Master**. The `-L` flag lists all keys currently known to the Master, categorizing them into:

- **Accepted Keys:** Minions that are currently managed.
- **Denied Keys:** Keys that were explicitly rejected.
- **Unaccepted Keys:** New keys sent by Minions awaiting approval.

Why: After the Minion is restarted, its new key appears in the **Unaccepted Keys** section. We run this command to verify that the Minion successfully contacted the Master and is ready for the next step.

Step 2.7: *sudo salt-key -a minion-01*

What it does: This command is run on the **Salt Master**. The `-a` flag tells the Master to **accept** the key from the Minion ID specified (here, `minion-01`)./etc/salt/pki/master/ - this is where all the keys are stored by the master of the minions

Why: For security, Salt uses a "key handshake" model. The Master must explicitly accept a Minion's key before it will communicate with it. Accepting the key establishes a trusted, encrypted channel between the Master and Minion, completing the initial setup.

Step 2.8 : *sudo salt 'minion-01' test.ping*

What it does: This is the first operational command sent from the Master to the Minion.

- `test.ping`: A simple module function that tells the Minion to respond with `True` if it is alive and communicating with the Master.

Why: This command verifies **end-to-end connectivity** and successful key acceptance. A response of `True` confirms that the Minion is running, listening, and correctly executing commands issued by the Master, signifying a successful setup.

```
Last login: Fri Nov 28 10:17:10 2025 from 192.168.64.1
ubuntu@salt-master:~$ sudo salt-key -L
Accepted Keys:
minion-01
Denied Keys:
Unaccepted Keys:
Rejected Keys:
ubuntu@salt-master:~$ sudo salt 'minion-01' test.ping
minion-01:
    True
ubuntu@salt-master:~$
```

The Salt Master and Minion are successfully set up and interacting, as verified by the successful `test.ping` response.

3. Salt State and Pillar Setup for Nginx

Step 3.1 : ***sudo mkdir -p /etc/salt/pillar***

What it does: Creates a new directory named `pillar` inside `/etc/salt/`. The `-p` flag ensures that if `/etc/salt/` doesn't exist, it's created too (though it usually already exists).

Why: The `/etc/salt/pillar` directory is the **standard place** on the **Salt Master** where we store **Pillar data files** (like private configuration info or specific settings for certain minions).

Step 3.2 : ***sudo vim /etc/salt/pillar/top.sls***

What it does: Opens the `top.sls` file inside the new `pillar` directory using the `vim` text editor so you can edit it.

Why: The `top.sls` file for Pillar is the index or map. It tells the Salt Master which Pillar data files (like `minion_data.sls`) should be given to which Minions (`minion-01`).

```
None
---
base:
    'minion-01':
        - minion_data
```

Step 3.3 : ***sudo cat /etc/salt/pillar/top.sls***

Step 3.4 : ***sudo vim /etc/salt/pillar/minion_data.sls***

What it does: Opens a new file named `minion_data.sls` in the `pillar` directory using `vim` so you can add content to it.
Why: This file contains the actual configuration data (the Pillar). In this case, it holds a custom message that the Nginx Salt State will likely use later. This data is specific to the minions mapped to it in `top.sls`.

```
None
---
nginx:
  hostname_message: 'Welcome to Minion-01, managed by SaltStack!'
```

Step 3.5 : ***sudo cat /etc/salt/pillar/minion_data.sls***

Step 3.6 : ***sudo vim /srv/salt/top.sls***

What it does: Opens the main `top.sls` file inside `/srv/salt/` (your Salt State environment) using `vim` for editing
Why: The `top.sls` file for States is the master map for configurations. It tells the Salt Master which Salt State file (like `nginx.sls`, which will contain the actual installation instructions) should be applied to which Minions (`minion-01`). This links the Minion to the Nginx state.

```
None
---
base:
  'minion-01':
    - nginx
```

Step 3.7 : ***sudo cat /srv/salt/top.sls***

Step 3.8 : ***sudo vim /srv/salt/nginx.sls***

What it does: Opens and creates the core Salt State file named **`nginx.sls`** inside the `/srv/salt/` directory using `vim`.

Why: This file is the recipe for setting up Nginx. It contains all the instructions to: install Nginx, manage its config files, ensure the default site is removed, create the custom index page, and ensure the Nginx service is running and enabled.

```
None
---
nginx_pkg:
  pkg.installed:
    - name: nginx
nginx_conf:
  file.managed:
    - name: /etc/nginx/conf.d/default.conf
    - source: salt://nginx/nginx_default.conf  # Custom Nginx config
    - user: root
    - group: root
    - mode: '644'
    - require:
        # Requisite argument list item (8 spaces for indentation)
        - pkg: nginx_pkg
    - watch_in:
        # Execution controller argument list item (8 spaces for indentation)
        - service: nginx_service
nginx_default_site_absent:
  file.absent:
    - name: /etc/nginx/sites-enabled/default
    - require:
      - pkg: nginx_pkg
    - watch_in:
      - service: nginx_service
nginx_index:
  file.managed:
    - name: /usr/share/nginx/html/index.html
    - source: salt://nginx/index.html.jinja  # Jinja template for index.html
    - template: jinja
    - require:
        # Requisite argument list item (8 spaces for indentation)
        - pkg: nginx_pkg
    - watch_in:
        # Execution controller argument list item (8 spaces for indentation)
```

```
        - service: nginx_service
nginx_service:
  service.running:
    - name: nginx
    - enable: true
```

Step 3.9 : ***sudo cat /srv/salt/nginx.sls***

Step 3.10 : ***sudo vim /srv/salt/nginx/nginx_default.conf***
What it does: Creates a new directory named `nginx` inside `/srv/salt/` and then **opens/creates** the Nginx configuration file named **`nginx_default.conf`** within it.
Why: This file is the custom Nginx configuration that will be copied to the minion. The `nginx.sls` state (Step 3.8, item `nginx_conf`) specifies that this file should be copied to `/etc/nginx/conf.d/default.conf` on the minion.

```
None
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html index.htm;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

Step 3.11 : ***sudo cat /srv/salt/nginx/nginx_default.conf***

Step 3.12 : ***sudo vim /srv/salt/nginx/index.html.jinja***

What it does: Opens/creates the **Jinja template** file named **`index.html.jinja`** in the `nginx` directory.
Why: This file is the custom web page that will be copied to the minion's web root (`/usr/share/nginx/html/index.html`). The `.jinja` extension indicates that it's a template, meaning it can use variables, specifically the Pillar data (`{{ pillar['nginx']['hostname_message'] }}`) you defined earlier.

```
None
<!DOCTYPE html>
<html>
<head>
    <title>Salt Master Failover Check</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; margin-top: 50px;
}
        .master-info { padding: 20px; border-radius: 8px; display: inline-block; }
        .primary { background-color: #d4edda; border: 1px solid #c3e6cb; color:
#155724; }
        .secondary { background-color: #f8d7da; border: 1px solid #f5c6cb; color:
#721c24; }
        h2 { margin-top: 0; }
    </style>
</head>
<body>

    <h1>Minion ID: {{ salt['grains.get']('id') }}</h1>

    {% set current_master_address = salt['config.get']('master_ip') %}

    {% if current_master_address == '192.168.64.11' %}
        {% set master_class = 'primary' %}
        {% set master_role = 'PRIMARY MASTER' %}
    {% else %}
        {% set master_class = 'secondary' %}
        {% set master_role = 'SECONDARY (FAILOVER) MASTER' %}
    {% endif %}

    <div class="master-info {{ master_class }}">
        <h2>MANAGEMENT STATUS</h2>
        <p>This Minion is currently managed by the:</p>
```

```
        <h3>{{ master_role }}</h3>

        <p>Master IP Address: {{ current_master_address }}</p>
    </div>

</body>
</html>
```

Step 3.13 : ***sudo cat /srv/salt/nginx/index.html.jinja***

Step 3.14 : ***sudo vim /etc/salt/master***

What it does: Opens the main Salt Master configuration file and adds the **`pillar_roots`** section, directing the Master to look for Pillar data files inside the `/etc/salt/pillar` directory.
Why: This is critical because you chose to store your Pillar files in `/etc/salt/pillar` instead of the default `/srv/pillar`. This setting explicitly tells the Master where to find the Pillar files needed for the Minions.

You must explicitly add the pillar_roots section to the /etc/salt/master file and restart the service.

None

```
pillar_roots:
  base:
    - /etc/salt/pillar
```

Step 3.15 : *sudo systemctl restart salt-master*

What it does: Stops and then starts the Salt Master service.
Why: The Master service must be restarted to read and apply any changes made to its main configuration file (`/etc/salt/master`). Without this, the previous step's configuration would not take effect.

This is the non-negotiable step to load the new configuration.

Step 3.16 **:** ***sudo salt 'minion-01' saltutil.refresh_pillar***
What it does: Tells the specific minion (`minion-01`) to discard its current Pillar data and fetch the updated Pillar data from the Master.
Why: While the Master restart is necessary, Minions often cache their Pillar data. This command explicitly forces the minion to refresh its cache and load the new data you just configured, ensuring it has the required `nginx:hostname_message` value.

Step 3.17 : *sudo salt 'minion-01' pillar.get nginx:hostname_message*

What it does: Asks the minion (`minion-01`) to look up and return the value associated with the Pillar key `nginx:hostname_message`.
Why: This is the final verification step. It confirms that the Minion successfully contacted the Master, found the Pillar configuration, refreshed its cache, and can now access the specific data that the Nginx Jinja template will need.

```
[ubuntu@salt-master:~$ sudo salt 'minion-01' saltutil.refresh_pillar
 minion-01:
     True
[ubuntu@salt-master:~$ sudo salt 'minion-01' pillar.get nginx:hostname_message
 minion-01:
     Welcome to Minion-01, managed by SaltStack!
```

Step 3.18 : *sudo salt 'minion-01' state.highstate test=True*

What it does: Runs the entire configuration sequence (`state.highstate`) defined in all your `top.sls` files, but in Test Mode (`test=True`) for the minion `minion-01`.
Why: This is a dry run that shows you exactly what changes Salt would make on the minion without actually executing them. It checks for syntax errors, missing files, and confirms that Salt is ready to install Nginx, apply the configuration, and start the service.

Step 3.19 : *sudo salt 'minion-01' state.highstate// true execution*

What it does: Runs the entire configuration sequence (`state.highstate`) defined in all your **`top.sls`** files on the minion `minion-01`, executing all changes.
Why: This is the true execution (the deployment). Salt will now perform all the actions defined in the `nginx.sls` file: install the package, manage the configuration files and the custom index page (inserting the Pillar message), and ensure the Nginx service is running. This finalizes the deployment.

```
,
ubuntu@salt-master:~$ sudo vim /srv/salt/nginx/nginx_default.conf
ubuntu@salt-master:~$ sudo salt 'minion-01' state.highstate
minion-01:
----------
          ID: nginx_pkg
    Function: pkg.installed
        Name: nginx
      Result: True
     Comment: All specified packages are already installed
     Started: 17:02:05.990860
    Duration: 12.717 ms
     Changes:
----------
          ID: nginx_conf
    Function: file.managed
        Name: /etc/nginx/conf.d/default.conf
      Result: True
     Comment: File /etc/nginx/conf.d/default.conf updated
     Started: 17:02:06.004319
    Duration: 13.158 ms
     Changes:
              ----------
              diff:
                  ---
                  +++
                  @@ -1,16 +1,10 @@
                  -server_name localhost;
                  -
                  -root /usr/share/nginx/html;
                  -
                  -index index.html index.htm;
                  -
                  -location / {
                  -    try_files $uri $uri/ =404;
                  -}
                   server {
                        listen 80;
                  -     server_name example.com;  # <-- The server_name directive belongs HERE
                  +     server_name localhost;
                        root /usr/share/nginx/html;
                  +     index index.html index.htm;

                  -     # other location blocks...
                  +     location / {
                  +         try_files $uri $uri/ =404;
                  +     }
                   }
----------
          ID: nginx_index
    Function: file.managed
        Name: /usr/share/nginx/html/index.html
      Result: True
     Comment: File /usr/share/nginx/html/index.html is in the correct state
     Started: 17:02:06.017550
    Duration: 7.582 ms
     Changes:
----------
          ID: nginx_service
    Function: service.running
        Name: nginx
      Result: True
     Comment: Service nginx is already enabled, and is running
     Started: 17:02:06.025626
    Duration: 63.634 ms
     Changes:
              ----------
              nginx:
                  True

Summary for minion-01
------------
Succeeded: 4 (changed=2)
Failed:    0
------------
Total states run:     4
Total run time:  97.091 ms
ubuntu@salt-master:~$ sudo salt 'minion-01' service.status nginx
minion-01:
    True
```

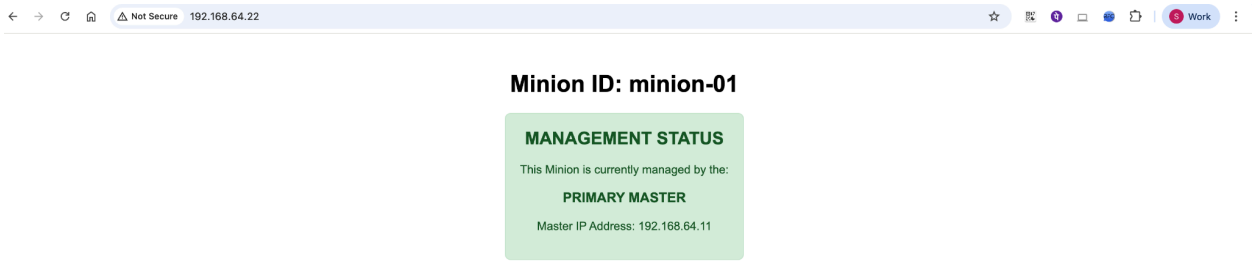
Step 3.20 : *sudo salt 'minion-01' grains.get ipv4*

What it does: Runs the `grains.get` function on `minion-01` to fetch the list of its configured IPv4 addresses.
Why: This is the **fastest, most reliable way** to get the minion's network address directly from SaltStack, which you need for your browser or `curl` command. **Note:** The output will be a list, so you'll choose the correct external address.

Once you have the IP address (e.g., `192.168.1.10`), use the below method. Since Nginx was configured to listen on port 80, no port number is needed.

Step 3.21 : **Open your browser and navigate to: `http://<minion-01-IP-address>`**

You will see the rendered HTML page. This confirms: 1) Nginx is running, 2) The custom `index.html.jinja` file was copied, and 3) The Jinja template successfully read and injected the Pillar data.



Step 3.22 : **sudo salt 'minion-01' service.status nginx**

What it does: Asks the minion to use its local system tools to check the current operational status of the **`nginx` service**.
Why: This confirms that the final line of your `nginx.sls` file (`service.running`) was successful and the web server is actively running on the Minion.

```
ubuntu@salt-master:~$   sudo salt 'minion-01' service.status nginx
minion-01:
    True
ubuntu@salt-master:~$
```

4. Setup third VM and configure as second salt-master use it as failover while accessing from minion

Step 4.1 : ***sudo cat /etc/salt/pki/master/master.pem***

What it does: Reads and outputs the entire content of the **Primary Master's Private Key** to the terminal screen.
Why: You need to copy this sensitive, non-public key to the clipboard so it can be manually transferred to the Secondary Master.

Step 4.2 : ***sudo vim /etc/salt/pki/master/master.pem***

What it does: Opens the corresponding file on the **Secondary Master** with root privileges using the `vim` text editor.
Why: This allows you to paste the Primary Master's private key content, **overwriting** the unique private key the Secondary Master generated during its installation.

Step 4.3 : ***sudo cat /etc/salt/pki/master/master.pub***

What it does: Reads and outputs the entire content of the Primary Master's Public Key to the terminal screen.
Why: You need to copy this public key so it can be transferred to the Secondary Master.

Step 4.4 : ***sudo vim /etc/salt/pki/master/master.pub***

What it does: Opens the public key file on the **Secondary Master** with root privileges.
Why: This allows you to paste the Primary Master's public key content, ensuring the public identities match.

By copying the keys (both Private and Public) from Primary to Secondary, you are essentially giving the Secondary Master a perfect duplicate of the Primary Master's ID card. When the Minion fails over, it sees the same ID card and happily continues working.

Step 4.5 : ***sudo chown root:root /etc/salt/pki/master/master.pem***

What it does: Changes the **owner** of the private key file (`master.pem`) to the `root` user and the `root` group..
Why: Key files must be owned by `root` for Salt to recognize them as secure and use them to start the master process.

Step 4.6 : ***sudo chown root:root /etc/salt/pki/master/master.pub***

What it does: Changes the **owner** of the public key file (`master.pub`) to the `root` user and the `root` group.
Why: Ensures the public key has the correct system ownership.

Step 4.7 : ***sudo chmod 600 /etc/salt/pki/master/master.pem***

What it does: Changes the **permissions** of the private key file to `600` (read/write only by the owner, `root`).
Why: This is a crucial security step. The private key must not be readable by any other user or group on the system.

Step 4.8 : ***sudo systemctl restart salt-master***

What it does: Stops and then starts the `salt-master` service on the Secondary Master..
Why: The Master must be restarted to abandon the old keys and **load the new, synchronized keys** from the file system.

Step 4.9 : ***sudo systemctl status salt-master***

What it does: Displays the current operational status of the `salt-master` service.
Why: Confirms that the service successfully started (`Active: active (running)`) using the new keys. If it fails, there is an issue with the keys or permissions.

Step 4.10 : ***Copy all the configuration of primary master to secondary master***

What it does: Copy all files and folders from `/srv/salt/` (SLS states) and `/srv/pillar/` (Pillar data) from Primary to Secondary.
Why: Ensures the Minion receives the **exact same configuration data and states** regardless of which Master it connects to. This makes the failover transparent to the Minion.

Step 4.11 : ***Configure the minion for failover sudo vim /etc/salt/minion***

What it does: Opens the Minion's main configuration file for editing and defines the list of masters, the connection behavior, and monitoring intervals.

Why: This is the core configuration that enables **redundancy**. By listing masters in order, the Minion knows where to connect first and where to jump if the connection fails (`master_type: failover`). `master_alive_interval` sets the crucial timeout for failure detection.

```
None
# /etc/salt/minion

# List masters in preferred order (Primary first)
master:
  - <Primary_Master_IP_or_FQDN>  # e.g., salt-master or its IP
  - <Secondary_Master_IP_or_FQDN> # e.g., secondary-master or its IP

# Enable the minion to failover to the next master in the list if the current one is
down
master_type: failover

# How often the minion checks if the current master is alive (in seconds)
master_alive_interval: 60

# Optional: To switch back to the primary master once it's back online
master_failback: True
```

Step 4.12 : ***sudo salt-key -L then sudo salt-key -A***

What it does: Accepts the Minion's public key that was sent to the Primary Master, verifying the Minion's identity and authorizing it to receive commands.
Why: The Minion cannot receive commands or execute states until its key is **accepted** by at least one Master. This ensures the initial communication channel is open.

Step 4.13 : ***sudo salt 'minion-01' test.ping***

What it does: Sends a simple, lightweight command to the Minion and requests a `True` response
Why: This is the **baseline connectivity test**. It confirms the Minion is correctly configured, has accepted the key, and is actively listening for and receiving commands from the Primary Master.

Step 4.14 : ***sudo systemctl stop salt-master***

What it does: Sends a signal to immediately stop the `salt-master` service on the Primary Master.
Why: This **simulates a failure** (e.g., hardware crash, network outage) to test the Minion's failover mechanism in a controlled environment.

Step 4.15 : ***sudo systemctl status salt-master***

What it does: Displays the current state of the `salt-master` service on the Primary Master.
Why: Confirms that the service is actually stopped (`Active: inactive (dead)`), verifying the simulated failure is in effect before proceeding to the failover test.

Step 4.16 : ***sudo salt-key -L***

What it does: Lists all minion keys known to the Secondary Master, categorized as Accepted, Unaccepted, Denied, or Rejected.
Why: Verifies that the Minion has attempted to connect to the Secondary Master and that its key is now visible (it should appear, often under `Accepted Keys` or `Unaccepted Keys`, due to the shared Master identity).

Step 4.17 : ***sudo salt-key -A***

What it does: Accepts any keys currently listed under Unaccepted Keys on the Secondary Master.
Why: Ensures that the Minion is authorized to communicate with the Secondary Master. Since the Masters share keys, this step may be redundant but is a necessary safety check to ensure full authentication.

Step 4.18 : ***sudo salt 'minion-01' test.ping***

What it does: Sends a ping command to the Minion, this time originating from the Secondary Master**.**
Why: This is the **definitive confirmation of failover**. If it returns `True`, it means the Minion successfully disconnected from the Primary Master and established an active command connection with the Secondary Master.

```
ubuntu@secondary-master:~$ sudo salt-key -L
Accepted Keys:
Denied Keys:
Unaccepted Keys:
minion-01
Rejected Keys:
ubuntu@secondary-master:~$ sudo salt-key -A
The following keys are going to be accepted:
Unaccepted Keys:
minion-01
Proceed? [n/Y] Y
Key for minion minion-01 accepted.
ubuntu@secondary-master:~$ sudo salt 'minion-01' test.ping
minion-01:
    True
```

Step 4.20 : ***sudo tail -f /var/log/salt/minion***

What it does: Streams the last lines of the Minion's log file to the screen in real-time.
Why: Provides **proof of the failover event**. The log will show entries like "Master `<Primary IP>` is unresponsive" followed by "Attempting to authenticate with the Salt Master at `<Secondary IP>`," confirming the master switch.

```
root@minion-01:/etc/salt# systemctl status salt-minion
● salt-minion.service - The Salt Minion
     Loaded: loaded (/lib/systemd/system/salt-minion.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2025-12-01 17:43:55 IST; 18min ago
       Docs: man:salt-minion(1)
             file:///usr/share/doc/salt/html/contents.html
             https://docs.saltproject.io/en/latest/contents.html
   Main PID: 4680 (python3.10)
      Tasks: 7 (limit: 4605)
     Memory: 132.7M
        CPU: 22.500s
     CGroup: /system.slice/salt-minion.service
             ├─4680 /opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion
             └─4689 "/opt/saltstack/salt/bin/python3.10 /usr/bin/salt-minion MultiMinionProcessManager MinionProcessManager"

Dec 01 17:48:26 minion-01 salt-minion[4689]: [WARNING ] Master ip address changed from 192.168.64.11 to 192.168.64.21
Dec 01 17:48:26 minion-01 salt-minion[4689]: [ERROR   ] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
Dec 01 17:48:37 minion-01 salt-minion[4689]: [ERROR   ] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
Dec 01 17:48:47 minion-01 salt-minion[4689]: [ERROR   ] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
Dec 01 17:57:03 minion-01 salt-minion[5163]: /opt/saltstack/salt/lib/python3.10/site-packages/salt/utils/odict.py:21: DeprecationWarning: This module is deprecated. Use the standard library's collections
Dec 01 17:57:03 minion-01 salt-minion[5163]:   warn_until(
Dec 01 17:57:03 minion-01 salt-minion[5163]: /opt/saltstack/salt/lib/python3.10/site-packages/salt/modules/pip.py:87: UserWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en
Dec 01 17:57:03 minion-01 salt-minion[5163]:   import pkg_resources  # pylint: disable=3rd-party-module-not-gated
Dec 01 17:57:17 minion-01 salt-minion[5163]: /opt/saltstack/salt/lib/python3.10/site-packages/salt/utils/odict.py:21: DeprecationWarning: This module is deprecated. Use the standard library's collections
Dec 01 17:57:17 minion-01 salt-minion[5163]:   warn_until(
log file: udo tail -f /var/log/salt/minion
^C
root@minion-01:/etc/salt# sudo tail -f /var/log/salt/minion
2025-12-01 16:54:00,757 [salt.crypt       :984 ][ERROR   ][3909] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
2025-12-01 16:54:10,822 [salt.crypt       :984 ][ERROR   ][3909] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
2025-12-01 16:54:20,943 [salt.crypt       :984 ][ERROR   ][3909] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
2025-12-01 17:28:12,678 [salt.utils.parsers:1062][WARNING ][3909] Minion received a SIGTERM. Exiting.
2025-12-01 17:43:51,432 [salt.utils.parsers:1062][WARNING ][4421] Minion received a SIGTERM. Exiting.
2025-12-01 17:48:26,976 [salt.minion      :188 ][WARNING ][4689] Master ip address changed from 192.168.64.11 to 192.168.64.21
2025-12-01 17:48:26,978 [salt.minion      :188 ][WARNING ][4689] Master ip address changed from 192.168.64.11 to 192.168.64.21
2025-12-01 17:48:26,999 [salt.crypt       :984 ][ERROR   ][4689] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
2025-12-01 17:48:37,049 [salt.crypt       :984 ][ERROR   ][4689] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
2025-12-01 17:48:47,177 [salt.crypt       :984 ][ERROR   ][4689] The Salt Master has cached the public key for this node, this salt minion will wait for 10 seconds before attempting to re-authenticate
```

Step 4.21 : ***sudo salt 'minion-01' state.highstate*** OR ***sudo salt 'minion-01' state.apply nginx***

What it does: Instructs the Minion to apply all configured states (`highstate`) or the specific `nginx` state using the files located on the Secondary Master.
Why: Tests operational functionality. It verifies not only the command channel but also that the Secondary Master can correctly serve the synchronized SLS configuration files to the Minion.

```
minion-01
[Proceed? [n/Y] Y
Key for minion minion-01 accepted.
[ubuntu@secondary-master:~$ sudo salt 'minion-01' test.ping
minion-01:
    True
[ubuntu@secondary-master:~$ sudo salt 'minion-01' state.highstate
minion-01:
----------
          ID: nginx_pkg
    Function: pkg.installed
        Name: nginx
      Result: True
     Comment: The following packages were installed/updated: nginx
     Started: 17:57:04.019564
    Duration: 12892.58 ms
     Changes:
              ----------
              fontconfig-config:
                  ----------
                  new:
                      2.13.1-4.2ubuntu5
                  old:
              fonts-dejavu-core:
                  ----------
                  new:
                      2.37-2build1
                  old:
              libdeflate0:
                  ----------
                  new:
                      1.10-2
                  old:
              libfontconfig1:
                  ----------
                  new:
                      2.13.1-4.2ubuntu5
                  old:
              libgd3:
                  ----------
                  new:
                      2.3.0-2ubuntu2.3
                  old:
              libjbig0:
                  ----------
                  new:
                      2.1-3.1ubuntu0.22.04.1
                  old:
              libjpeg-turbo8:
                  ----------
                  new:
                      2.1.2-0ubuntu1
                  old:
              libjpeg8:
                  ----------
                  new:
                      8c-2ubuntu10
                  old:
              libnginx-mod-http-geoip2:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libnginx-mod-http-image-filter:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libnginx-mod-http-xslt-filter:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libnginx-mod-mail:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libnginx-mod-stream:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libnginx-mod-stream-geoip2:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              libtiff5:
                  ----------
                  new:
                      4.3.0-6ubuntu0.12
                  old:
              libwebp7:
                  ----------
                  new:
                      1.2.2-2ubuntu0.22.04.2
                  old:
              libxpm4:
                  ----------
                  new:
                      1:3.5.12-1ubuntu0.22.04.2
                  old:
              nginx:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              nginx-common:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
              nginx-core:
                  ----------
                  new:
                      1.18.0-6ubuntu14.7
                  old:
----------
          ID: nginx_conf
    Function: file.managed
        Name: /etc/nginx/conf.d/default.conf
      Result: True
     Comment: File /etc/nginx/conf.d/default.conf updated
     Started: 17:57:16.913499
    Duration: 13.714 ms
```

Step 4.20 : **Open your browser and navigate to: `http://<minion-01-IP-address>`**

What it does: Accesses the web page served by the Minion's Nginx instance (which you configured using a dynamic Jinja template).

Why: This is the **final visual confirmation**. The webpage should display text/color confirming it is now managed by the **Secondary Master's IP**, demonstrating the entire redundancy and file synchronization process worked.

Additional Task -

Problem Statemnet : "Safety Loop"

1. **Master** pushes config to **Minion 1**.
2. **Minion 1** runs `nginx -t`. If it sees `80xxx`, it returns **Fail**.
3. **Master** receives **Fail**, triggers **Rollback 1** and **Email 1**.
4. **Master** sees `update_minion_2` requires `update_minion_1` (which failed) and marks it as **Blocked/Skipped**.
5. **Result:** Your production site on Minion 2 stays safe and untouched.

Implement "Salt Orchestration"

In a standard state run (`salt '*' state.apply`), every minion acts independently and simultaneously. If you have 10 web servers, they all go down for an update at the same time. Orchestration allows the **Master** to act as a conductor, ensuring **Minion A** finishes its update before **Minion B** starts.

1. The Key Difference: Runners vs. Execution Modules

- **Execution Modules (`salt`):** Run on the Minion.
- **Runners (`salt-run`):** Run on the Master.
- **Orchestration:** A specific runner (`state.orchestrate`) that reads an SLS file on the Master and uses it to send specific, sequenced commands to various Minions.

The Master must act as a conductor, updating **Minion A** first and verifying its success before moving to **Minion B**.

Step 1.1 : Create your Orchestration SLS

On your **Salt Master**, we create a new directory and file: `/srv/salt/orch/web_rolling_update.sls`.

```
None
# /srv/salt/orch/web_rolling_update.sls

# Step 1: Update the first batch (e.g., Minion 1)
update_web_server_1: #unique name for this stage of the job.
  salt.state:       #salt state module its a runner
    - tgt: 'minion1'
    - sls:
      - nginx        # This runs your existing nginx state

# Step 2: Health check for Minion 1
check_web_server_1:
  salt.function: # Function Module: Tells the Master to run a single
command/function.
    - name: cmd.run # The Function: Use the 'cmd.run' module on the minion.
    - tgt: 'minion1'
    - arg:
      - curl -f http://localhost
    - require:
      - salt: update_web_server_1
```

```
# Step 3: Only if Minion 1 is healthy, update Minion 2
update_web_server_2:
  salt.state:
    - tgt: 'minion2'
    - sls:
      - nginx
    - require:  #"Only start Minion 2 if the check on Minion 1 passed."
      - salt: check_web_server_1
```

Step B: Execute the Orchestration that starts a runner process

Step 1.2 : sudo salt-run state.orchestrate orch.web_rolling_update

```
                        +    <h1>Minion ID: minion-2</h1>
                        +
                        +

                        -<p><em>Thank you for using nginx.</em></p>
                        +
                        +
                        +
                        +
                        +
                        +    <div class="master-info primary">
                        +        <h2>MANAGEMENT STATUS</h2>
                        +        <p>This Minion is currently managed by the:</p>
                        +
                        +        <h3>PRIMARY MASTER</h3>
                        +
                        +        <p>Master IP Address: 192.168.64.11</p>
                        +    </div>
                        +
                         </body>
                         </html>
                ----------
                          ID: nginx_service
                    Function: service.running
                        Name: nginx
                      Result: True
                     Comment: Service restarted
                     Started: 15:11:18.167368
                    Duration: 632.17 ms
                     Changes:
                              ----------
                              nginx:
                                  True

            Summary for minion-2
            ------------
            Succeeded: 5 (changed=5)
            Failed:    0
            ------------
            Total states run:     5
            Total run time:  12.819 s

Summary for salt-master_master
------------
Succeeded: 3 (changed=3)
Failed:    0
------------
Total states run:     3
Total run time:  27.551 s
```

If `minion-1` had failed (e.g., a syntax error in the Nginx config), the Orchestration would have **stopped immediately**. `minion-2` would have stayed online with its old configuration, preventing a total site outage.

Now we will purposely add an error in the nginx config file and run the orchestration command to see how the update on minion1 failed and it does not pass the error to the next minion. Our minion-2 will continue to run on the older version avoiding error

In a manual world (or using a simpler script), an engineer might run a command across all servers at once, breaking the entire website.

Through this we achieve -

- **Zero-Downtime Reliability:** Even though your code was "broken," 50% of your infrastructure (Minion-2) stayed online and healthy.

- **Automated Validation:** The orchestration didn't just "try its best"; it validated the outcome before proceeding.

```
ubuntu@salt-master:/srv/salt/orch$ sudo vim  /srv/salt/nginx/nginx_default.conf
ubuntu@salt-master:/srv/salt/orch$ sudo cat  /srv/salt/nginx/nginx_default.conf
server {
    listen 80xxx;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }
}
ubuntu@salt-master:/srv/salt/orch$ sudo salt-run state.orchestrate orch.web_rolling_update
[ERROR   ] {'out': 'highstate', 'ret': {'minion-1': {'pkg_|-nginx_pkg_|-nginx_|-installed': {'name': 'nginx', 'changes': {}, 'result': True, 'comment': 'All specified packages are already installed', '__sls__': 'nginx', '__run_num__': 0, 'start_
time': '15:29:39.046495', 'duration': 17.539, '__id__': 'nginx_pkg'}, 'file_|-nginx_conf_|-/etc/nginx/conf.d/default.conf_|-managed': {'changes': {'diff': '--- \n+++ \n@@ -1,5 +1,5 @@\n server {\n-    listen 80;\n+    listen 80xxx;\n     server_
name localhost;\n     root /usr/share/nginx/html;\n     index index.html index.htm;\n', 'comment': 'File /etc/nginx/conf.d/default.conf updated', 'name': '/etc/nginx/conf.d/default.conf', 'result': True, '__sls__': 'nginx', '__run_num__': 1, 's
tart_time': '15:29:39.065390', 'duration': 17.861, '__id__': 'nginx_conf'}, 'file_|-nginx_default_site_absent_|-/etc/nginx/sites-enabled/default_|-absent': {'name': '/etc/nginx/sites-enabled/default', 'changes': {}, 'result': True, 'comment': 'F
ile /etc/nginx/sites-enabled/default is not present', '__sls__': 'nginx', '__run_num__': 2, 'start_time': '15:29:39.083343', 'duration': 0.225, '__id__': 'nginx_default_site_absent'}, 'file_|-nginx_index_|-/usr/share/nginx/html/index.html_|-mana
ged': {'changes': {}, 'comment': 'File /usr/share/nginx/html/index.html is in the correct state', 'name': '/usr/share/nginx/html/index.html', 'result': True, '__sls__': 'nginx', '__run_num__': 3, 'start_time': '15:29:39.083613', 'duration': 14.0
85, '__id__': 'nginx_index'}, 'service_|-nginx_service_|-nginx_|-running': {'name': 'nginx', 'changes': {}, 'result': False, 'comment': 'Running as unit: run-r767176882e7c4c3e8e404fe6565e4d7c7.scope; invocation ID: ede20e44aac643e68d9b44add86b9e2
8\nJob for nginx.service failed because the control process exited with error code.\nSee "systemctl status nginx.service" and "journalctl -xeu nginx.service" for details.', '__sls__': 'nginx', '__run_num__': 5, 'start_time': '15:29:39.134433', '
duration': 353.14, '__id__': 'nginx_service'}}}}
salt-master_master:
----------
          ID: update_web_server_1
    Function: salt.state
      Result: False
     Comment: Run failed on minions: minion-1
     Started: 15:29:37.854225
    Duration: 1609.977 ms
     Changes:
              minion-1:
              ----------
                        ID: nginx_pkg
                  Function: pkg.installed
                      Name: nginx
                    Result: True
                   Comment: All specified packages are already installed
                   Started: 15:29:39.046495
                  Duration: 17.539 ms
                   Changes:
              ----------
                        ID: nginx_conf
                  Function: file.managed
                      Name: /etc/nginx/conf.d/default.conf
                    Result: True
                   Comment: File /etc/nginx/conf.d/default.conf updated
                   Started: 15:29:39.065390
                  Duration: 17.861 ms
                   Changes:
                            ----------
                            diff:
                                ---
                                +++
                                @@ -1,5 +1,5 @@
                                 server {
                                -    listen 80;
                                +    listen 80xxx;
                                     server_name localhost;
                                     root /usr/share/nginx/html;
                                     index index.html index.htm;
              ----------
                        ID: nginx_default_site_absent
                  Function: file.absent
                      Name: /etc/nginx/sites-enabled/default
```

```
                    diff:
                        ---
                        +++
                        @@ -1,5 +1,5 @@
                         server {
                        -    listen 80;
                        +    listen 80xxx;
                             server_navigation localhost;
                             root /usr/share/nginx/html;
                             index index.html index.htm;
              ----------
                      ID: nginx_default_site_absent
                Function: file.absent
                    Name: /etc/nginx/sites-enabled/default
                  Result: True
                 Comment: File /etc/nginx/sites-enabled/default is not present
                 Started: 15:29:39.083343
                Duration: 0.225 ms
                 Changes:
              ----------
                      ID: nginx_index
                Function: file.managed
                    Name: /usr/share/nginx/html/index.html
                  Result: True
                 Comment: File /usr/share/nginx/html/index.html is in the correct state
                 Started: 15:29:39.083613
                Duration: 14.085 ms
                 Changes:
              ----------
                      ID: nginx_service
                Function: service.running
                    Name: nginx
                  Result: False
                 Comment: Running as unit: run-r767176082e7c4c3e8e404fe565e4d7c7.scope; invocation ID: edc20e44aac643e68d9b44add86b9e28
                          Job for nginx.service failed because the control process exited with error code.
                          See "systemctl status nginx.service" and "journalctl -xeu nginx.service" for details.
                 Started: 15:29:39.134433
                Duration: 353.14 ms
                 Changes:

              Summary for minion-1
              ------------
              Succeeded: 4 (changed=1)
              Failed:    1
              ------------
              Total states run:     5
              Total run time: 402.850 ms
----------
          ID: check_web_server_1
    Function: salt.function
        Name: cmd.run
      Result: False
     Comment: One or more requisite failed: orch.web_rolling_update.update_web_server_1
     Started: 15:29:39.464589
    Duration: 0.003 ms
     Changes:
----------
          ID: update_web_server_2
    Function: salt.state
      Result: False
     Comment: One or more requisite failed: orch.web_rolling_update.check_web_server_1
     Started: 15:29:39.464697
    Duration: 0.002 ms
     Changes:

Summary for salt-master_master
------------
Succeeded: 0 (changed=1)
Failed:    3
------------
Total states run:     3
Total run time:   1.610 s
ubuntu@salt-master:/srv/salt/orch$
```

Rollback and Email Notifications

```
/srv/salt/
└── nginx/
    ├── init.sls            # Your standard Nginx state
    ├── previous_stable.sls  # The rollback state we are creating
    └── files/
        ├── default.conf.j2  # The template we are editing (and might break)
        └── stable.conf      # A hard-coded, known-working config file
```

Enhance the Orchestration logic with an `onfail` trigger. If the Nginx update fails the health check, the Master must automatically send an email alert to team.

ubuntu@salt-master:**/srv/salt/orch**$ sudo mkdir -p /srv/salt/nginx/files
ubuntu@salt-master:**/srv/salt/orch**$ sudo mv /srv/salt/nginx/nginx_default.conf /srv/salt/nginx/files/default.conf.j2

```
None
server {
    listen 80xxx;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html index.htm;
    location / {
        try_files $uri $uri/ =404;
    }
}
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo vim /srv/salt/nginx/files/stable.conf

```
None
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;
}
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo vim /srv/salt/nginx/init.sls

```
None
nginx_packages:
  pkg.installed:
    - pkgs:
      - nginx
      - apache2-utils
      - ca-certificates
# Manage the whole directory, but add the shield here
nginx_config_dir:
  file.recurse:
    - name: /etc/nginx
    - source: salt://nginx/etc/nginx
    - include_empty: True
    - template: jinja
```

```
    - dir_mode: '0755'
    - file_mode: '0644'
    # THE SHIELD: This stops the recurse from happening if the source files are
broken
    - check_cmd: /usr/sbin/nginx -t -c /etc/nginx/nginx.conf -g "include %s;"
    - require:
      - pkg: nginx_packages
# This fixes the '80xxx' issue by ensuring the resolver is valid
nginx_dns_resolver:
  cmd.run:
    - name: (echo "resolver "; cat /etc/resolv.conf |awk '{print $2}' ; echo ";") |
tr '\n' ' ' > /etc/nginx/resolver.conf
    - unless: test -f /etc/nginx/resolver.conf && [ "$(stat -c %Y /etc/resolv.conf)"
-lt "$(stat -c %Y /etc/nginx/resolver.conf)" ]
    - require:
      - file: nginx_config_dir
nginx_service:
  service.running:
    - name: nginx
    - enable: True
    - reload: True
    - watch:
      - file: nginx_config_dir
      - cmd: nginx_dns_resolver
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo vim /srv/salt/nginx/previous_stable.sls

```
None
# 1. Kill the file that is actually causing the '80xxx' crash
remove_poison_config:
  file.absent:
    - name: /etc/nginx/conf.d/default.conf
# 2. Ensure the site config is back to stable
rollback_nginx_config:
  file.managed:
    - name: /etc/nginx/sites-available/default
    - source: salt://nginx/files/stable.conf
    - user: root
    - group: root
    - mode: 644
# 3. Now Nginx WILL start because the poison file is gone
restart_nginx_after_rollback:
  service.running:
    - name: nginx
```

```
    - enable: True
    - force_restart: True
    - watch:
      - file: rollback_nginx_config
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo vim /srv/salt/orch/web_rolling_update.sls

```
None
update_minion_1:
  salt.state:
    - tgt: 'minion-1'
    - sls:
      - nginx.init
rollback_minion_1:
  salt.state:
    - tgt: 'minion-1'
    - sls:
      - nginx.previous_stable
    - onfail:
      - salt: update_minion_1
alert_admin:
  salt.function:
    - name: smtp.send_msg
    - tgt: 'minion-1'
    - kwarg:
        recipient: 'shambhavi.intern@phonepe.com'
        message: 'Nginx deployment failed on minion-1. Rollback has been initiated.'
        subject: 'SaltStack Alert: Deployment Failed'
        sender: 'shambhavi.intern@phonepe.com' # Add this
        server: 'smtp.gmail.com'
        username: 'shambhavi.intern@phonepe.com' # 'user' becomes 'username'
        password: 'iwgtazsqbzayhhxz'
    - onfail:
      - salt: update_minion_1
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo vim /etc/salt/master.d/smtp.conf

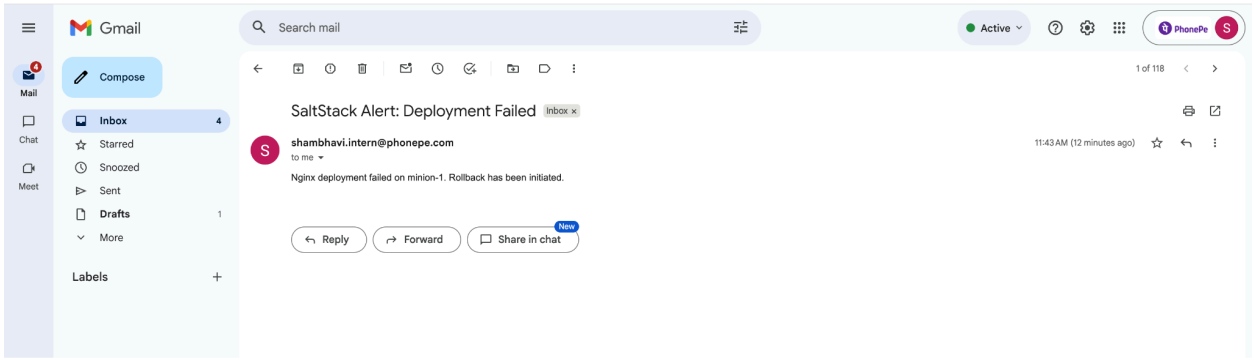
```
None
smtp.host: 'smtp.gmail.com'
smtp.port: 587
```

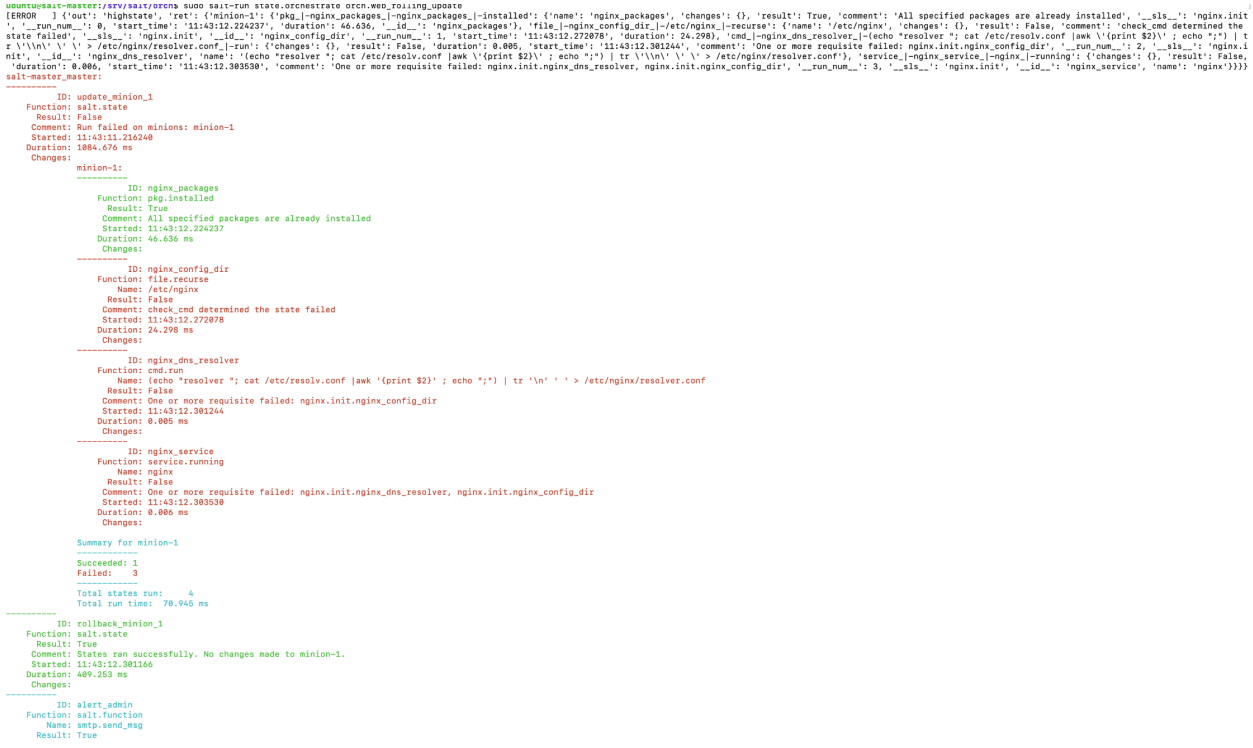
```
smtp.user: 'shambhavi.intern@phonepe.com'          # Your actual Gmail address
smtp.password: 'iwgtazsqbzayhhxz'       # The 16-character App Password (no spaces)
smtp.tls: True
smtp.from: 'shambhavi.intern@phonepe.com'          # Must match your Gmail address
```

ubuntu@salt-master:**/srv/salt/orch**$ sudo systemctl restart salt-master

ubuntu@salt-master:**/srv/salt/orch**$ sudo salt-run smtp.send \
  subject='SaltStack Alert System Test' \
  message='If you see this, your Salt Master is ready to send alerts!' \
  recipient='shambhavi.intern@phonepe.com'
'smtp.send' is not available.
ubuntu@salt-master:**/srv/salt/orch**$ sudo apt-get update
ubuntu@salt-master:**/srv/salt/orch**$ sudo apt-get install python3-pip -y
Salt's SMTP module is written in Python. It requires specific libraries to "speak" to the internet. You are installing the underlying tools that Salt needs to make an outgoing connection to Google.
ubuntu@salt-master:**/srv/salt/orch**$ sudo apt install python3-requests
ubuntu@salt-master:**/srv/salt/orch**$ sudo systemctl restart salt-master
ubuntu@salt-master:**/srv/salt/orch**$ sudo salt-run saltutil.sync_runners
ubuntu@salt-master:**/srv/salt/orch**$ sudo salt-run sys.list_runners | grep smtp

ubuntu@salt-master:**/srv/salt/orch**$ sudo salt-run state.orchestrate orch.web_rolling_update





Salt Beacons

A Beacon is a small process that runs on the **Minion**. It constantly watches a specific "event." When that event occurs (e.g., a file is deleted, or a service stops), the Beacon sends an **Event** to the Salt Master's **Event Bus**.

Right now, our orchestration runs only when **you** execute the command. If we add a **Beacon** and a **Reactor**:

1. If someone manually logs into Minion-1 and breaks the Nginx config.
2. The **Beacon** detects the change.
3. The **Reactor** automatically runs your `web_rolling_update` orchestration to fix it.

Create a Master-side listener that "hears" the crash and automatically triggers the Orchestration pipeline to repair the service and roll back to a `stable.conf` if necessary.

On **minion-1**, we need to tell Salt what to watch. You can do this by creating a file in `/etc/salt/minion.d/`

Step 1.1 : sudo vim /etc/salt/minion.d/beacons.conf

```
None
beacons:
  service:
    - services:
        nginx:
          onchangeonly: True  # Only fire an event when the status CHANGES
    - interval: 10            # Check every 10 seconds
```

Step 1.2 : sudo systemctl restart salt-minion

Step 1.3 : sudo salt-run state.event pretty=True
This is to check the logs of the nginx which we will run on the master

Step 1.4 : sudo systemctl stop nginx
Here we are stopping the nginx

```
ubuntu@salt-master:/srv/salt/orch$ sudo salt-run state.event pretty=True
salt/beacon/minion-1/service/nginx      {
    "_stamp": "2026-01-12T09:05:49.402198",
    "id": "minion-1",
    "nginx": {
        "running": true
    },
    "service_name": "nginx"
}
salt/beacon/minion-1/service/nginx      {
    "_stamp": "2026-01-12T09:06:09.366015",
    "id": "minion-1",
    "nginx": {
        "running": false
    },
    "service_name": "nginx"
}
```

Setting up Reactor

The Reactor is **Event-based**. It happens 24/7, even when you are sleeping and no one is touching the Salt Master.

- **Scenario:** At 3:00 AM, a random disk error causes Nginx to crash, or a junior admin manually logs into the server and accidentally stops the service.
- **Action:** The **Beacon** "sees" the crash and tells the Master. The **Reactor** hears it and fixes it.
- **Goal:** To maintain "Up-time" against unplanned accidents or crashes.

we must tell the Salt Master to listen for the specific "shout" coming from the Minion's service beacon.

Step 1.1 : sudo vim /etc/salt/master.d/reactor.conf

```
None
reactor:
  # This matches the tag sent by the service beacon when Nginx status changes
  - 'salt/beacon/*/service/nginx':
    - /srv/salt/reactor/remediate_nginx.sls
```

Step 1.1 : sudo mkdir -p /srv/salt/reactor
Step 1.1 : sudo vim /srv/salt/reactor/remediate_nginx.sls

```
None
# Only trigger if the beacon reports that Nginx is NOT running
{% if data.get('nginx', {}).get('running') == False %}
trigger_safe_orchestration:
  runner.state.orchestrate:
    - args:
      - mods: orch.web_rolling_update
{% endif %}
```

Step 1.1 : sudo systemctl restart salt-master

Step 1.1 : sudo salt-run state.event pretty=True

Step 1.1 : sudo systemctl stop nginx

```
salt/auth       {
    "_stamp": "2026-01-12T09:21:54.961503",
    "act": "accept",
    "id": "minion-01",
    "pub": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmG42cRiQaMZA/0tz6eIl\nZbQDWbqifmj97kKvqnC9h5/snQ8gS2WpSbj/zpeqAjPakksVTp2b4zBKMnUp1sAW\nU5GpnaP7ycz/L3xl1o5dgCZrPR
pYWD0rfWgshMXe6qRcSrD05UnV6+QentWor1070/JVltpEC/jakcY1luM\nA6Mlq0XqA0o26t8omAjjV5FOpiyLgFIUTLqB0Ar1PdgiunDnkwRMFgo8ztJ6RL0t\n6/aTSdzN65szHrKX/WWwAgflt1niEmGOKLXXnm/1nmFtZZPZN4L8Blk8xFOqP65m\ntwIDAQA
    "result": true
}
minion_start    {
    "_stamp": "2026-01-12T09:21:55.517746",
    "cmd": "_minion_event",
    "data": "Minion minion-01 started at Mon Jan 12 14:51:55 2026",
    "id": "minion-01",
    "pretag": null,
    "tag": "minion_start",
    "ts": 1768209715
}
salt/minion/minion-01/start     {
    "_stamp": "2026-01-12T09:21:55.555834",
    "cmd": "_minion_event",
    "data": "Minion minion-01 started at Mon Jan 12 14:51:55 2026",
    "id": "minion-01",
    "pretag": null,
    "tag": "salt/minion/minion-01/start",
    "ts": 1768209715
}
salt/beacon/minion-1/service/nginx      {
    "_stamp": "2026-01-12T09:22:19.404407",
    "id": "minion-1",
    "nginx": {
        "running": false
    },
    "service_name": "nginx"
}
salt/run/20260112092219706415/new       {
    "_stamp": "2026-01-12T09:22:19.708187",
    "fun": "runner.state.orchestrate",
    "fun_args": [
        {
            "mods": "orch.web_rolling_update"
        }
    ],
    "jid": "20260112092219706415",
    "user": "Reactor"
}
salt/run/20260112092219832977/args      {
    "_stamp": "2026-01-12T09:22:19.898823",
    "args": {
        "arg": [
            "nginx.init"
        ],
        "expect_minions": true,
        "kwarg": {
            "concurrent": false,
            "queue": false
        },
        "ret": "",
        "ssh": false,
```

```
                "changes": {
                    "out": "highstate",
                    "ret": {
                        "minion-1": {
                            "cmd_|-nginx_dns_resolver_|-(echo \"resolver \"; cat /etc/resolv.conf |awk '{print $2}' ; echo \";\") | tr '\\n' ' ' > /etc/nginx/re
                                "__id__": "nginx_dns_resolver",
                                "__run_num__": 2,
                                "__sls__": "nginx.init",
                                "changes": {},
                                "comment": "One or more requisite failed: nginx.init.nginx_config_dir",
                                "duration": 0.005,
                                "name": "(echo \"resolver \"; cat /etc/resolv.conf |awk '{print $2}' ; echo \";\") | tr '\\n' ' ' > /etc/nginx/resolver.conf",
                                "result": false,
                                "start_time": "14:52:21.602540"
                            },
                            "file_|-nginx_config_dir_|-/etc/nginx_|-recurse": {
                                "__id__": "nginx_config_dir",
                                "__run_num__": 1,
                                "__sls__": "nginx.init",
                                "changes": {},
                                "comment": "check_cmd determined the state failed",
                                "duration": 12.443,
                                "name": "/etc/nginx",
                                "result": false,
                                "start_time": "14:52:21.587904"
                            },
                            "pkg_|-nginx_packages_|-nginx_packages_|-installed": {
                                "__id__": "nginx_packages",
                                "__run_num__": 0,
                                "__sls__": "nginx.init",
                                "changes": {},
                                "comment": "All specified packages are already installed",
                                "duration": 40.827,
                                "name": "nginx_packages",
                                "result": true,
                                "start_time": "14:52:21.542101"
                            },
                            "service_|-nginx_service_|-nginx_|-running": {
                                "__id__": "nginx_service",
                                "__run_num__": 3,
                                "__sls__": "nginx.init",
                                "changes": {},
                                "comment": "One or more requisite failed: nginx.init.nginx_dns_resolver, nginx.init.nginx_config_dir",
                                "duration": 0.005,
                                "name": "nginx",
                                "result": false,
                                "start_time": "14:52:21.605620"
                            }
                        }
                    }
                },
                "comment": "Run failed on minions: minion-1",
                "duration": 1702.581,
                "name": "update_minion_1",
                "result": false,
                "start_time": "14:52:19.896920"
            }
        }
    },
    "outputter": "highstate",
    "retcode": 1
},
"success": false,
"user": "Reactor"
}
salt/beacon/minion-1/service/nginx      {
    "_stamp": "2026-01-12T09:22:29.327817",
    "id": "minion-1",
    "nginx": {
        "running": true
    },
    "service_name": "nginx"
}
```


1. The Detection (The Beacon)

At timestamp `09:22:19`, the Beacon on **minion-1** sent a message to the Master's event bus. **Tag:** `salt/beacon/minion-1/service/nginx` **Data:** `"running": false` This is the "shout" for help. Nginx was stopped, and the Beacon caught it instantly.

2. The Decision (The Reactor)

Milliseconds later, the **Reactor** saw that specific tag and consulted your mapping.**User:** `Reactor` **Function:** `runner.state.orchestrate` **Args:** `mods: orch.web_rolling_update` The Reactor didn't just try to restart Nginx; it triggered your professional **Orchestration** pipeline to ensure a safe, validated recovery.

3. The Attempted Update (The Shield)

The Orchestration started the `update_minion_1` job. However, because you still had the `80xxx` port error in your config, the **Shield** blocked it: **ID:** `nginx_config_dir` **Comment:** `check_cmd determined the state failed` **Result:** `false` This is exactly what we want. The system refused to apply the "poison" config even during an automated recovery.

4. The Self-Healing Loop (Rollback & Alert)

Because the update failed, the Orchestration triggered the `onfail` steps:

- **Step A: Rollback**
  ID: `rollback_minion_1` triggers `nginx.previous_stable` **Action:** It successfully restarted Nginx using the stable configuration.
- **Step B: Alert**
  Function: `smtp.send_msg` **Result:** `true` The Master successfully sent the email alert to your inbox at `09:22:26`.

5. The Resolution (Success)

Finally, at timestamp `09:22:29`, the Beacon fired one last time:

> **nginx:** `{"running": true}` **The loop is complete.** The system detected a failure, tried to update, hit a config error, rolled back to a safe version, alerted you, and confirmed the service is back online.