Saltstack - Week 4

Assignment Report

Siddharth Kumar

August 26, 2025

Contents

Chapter 1

Setup Salt Master and Minion

1.1 Master - Minion setup

```
1  # On Master VM
2  sudo apt update
3  sudo apt install salt-master -y
4
5  # On Minion VM
6  sudo apt update
7  sudo apt install salt-minion -y
8  sudo vim /etc/salt/minion
9       master: 192.168.64.X   #replace with salt-master-vm IP
10 sudo systemctl restart salt-minion
11
12 #On Master VM
13 sudo salt-key -L
14 sudo salt-key -A
```

Listing 1.1: Installing Salt Master and Minion

Explanation
- Lines 2–3: Update package lists and install the Salt master on the master VM.
- Lines 6–7: Update package lists and install the Salt minion on the minion VM.
- Lines 8–9: Edit the minion configuration file to point it to the master's IP address.
- Line 10: Restart the Salt minion service so it connects to the master.
- Lines 13–14: On the master, list pending minion keys and accept them so the master–minion trust is established.

1.2 Screenshots

Figure 1.1: Master Accepted our minions keys

Chapter 2

Salt States and Pillars

2.1 Salt State for Config File

2.1.1 State File - /srv/salt/top.sls

```
1  base:
2  'nginx-vm':
3    - nginx
```

Listing 2.1: Top.sls

Explanation
- Line 1: Defines the ‘base‘ environment in the Salt top file.

- Line 2: Targets the minion with ID ‘nginx-vm‘.

- Line 3: Applies the ‘nginx‘ state to that minion, ensuring the defined configuration is enforced.

2.1.2 State File -/srv/salt/nginx/init.sls

```
1  nginx:
2    pkg.installed:
3      - name: nginx
4
5  service.running:
6      - name: nginx
7      - enable: True
8      - reload: True
9
10 /etc/nginx/sites-available/bookstore:
11   file.managed:
12     - source: salt://nginx/bookstore.jinja
13     - template: jinja
14     - context:
15 backend_servers: {{ salt['mine.get']('api*', 'network.
       ip_addrs') }}
16 frontend_server: {{ pillar['nginx']['frontend_server'] }}
17 server_name: {{ pillar['nginx']['server_name'] }}
18     - require:
19 - pkg: nginx
20
21
22 /etc/nginx/sites-enabled/bookstore:
23   file.symlink:
24     - target: /etc/nginx/sites-available/bookstore
25     - require:
26 - file: /etc/nginx/sites-available/bookstore
```

Listing 2.2: init.sls

Explanation
- Lines 1–3: The ‘nginx‘ package is installed using the ‘pkg.installed‘ state. This ensures that the nginx software is present on the minion before attempting to start the service or manage configuration files.

- Lines 5–8:
The ‘service.running‘ state ensures that the nginx service is:
- Running (‘service.running‘).
- Enabled to start automatically on boot (‘enable: True‘).
- Reloaded whenever there are configuration changes (‘reload: True‘).

- Lines 10–19:
The ‘/etc/nginx/sites-available/bookstore‘ file is managed using ‘file.managed‘ with a Jinja template:

- ‘source: salt://nginx/bookstore.jinja‘ points to the template in the Salt file server.
- ‘template: jinja‘ tells Salt to render it as a Jinja template.
- ‘context:‘ defines variables used in the template:
- ‘backend_servers‘ is dynamically fetched from minions whose IDs match ‘api*‘ using the Salt Mine.
- ‘frontend_server‘ and ‘server_name‘ are pulled from pillar data for flexible configuration.
- ‘require:‘ ensures that the nginx package is installed before attempting to manage the config file.

- Lines 22–26:
The ‘/etc/nginx/sites-enabled/bookstore‘ symlink is created with ‘file.symlink‘:
- ‘target‘ points to the managed config file in ‘sites-available‘.
- ‘require:‘ ensures that the source config file exists before creating the symlink.
- This is a standard nginx pattern to enable sites while keeping the original configuration in ‘sites-available‘.

- Overall: This state file ensures a fully configured nginx server: package installed, service running, site configuration deployed with dynamic values, and site enabled via symlink.

2.1.3 State File -/srv/salt/nginx/bookstore.jinja

```
1  # Upstream backend APIs
2  upstream backend_api {
3      {% for backend, ips in backend_servers.items() %}
4          {% for ip in ips %}
5      server {{ ip }}:5000;
6          {% endfor %}
7      {% endfor %}
8  }
9
10 server {
11     listen 80;
12     server_name {{ server_name }};
13
14     # Frontend React Dev Server Proxy
15     location / {
16         proxy_pass http://{{ frontend_server }};
17         proxy_set_header Host $host;
18     }
19
20     # API Proxy
21     location /api/ {
22         proxy_pass http://backend_api/;
23     }
24 }
```

Listing 2.3: bookstore.jinja

Explanation
This Jinja template generates the nginx configuration for the bookstore application. It combines backend API servers and a frontend server into one configuration.
- Lines 1-8:
Define the upstream block ‘backend_api‘. - It loops through all backends and their IPs from ‘backend_servers‘.
- Each IP is added as a server on port 5000.
- This way, nginx can load balance between multiple backend API servers automatically.

- Lines 10-12:
Start the main ‘server‘ block.
- The server listens on port 80 (HTTP).
- The ‘server_name‘ is dynamically injected from pillar data, allowing flexible use for different domains.

- Lines 14-18:
Configure the frontend React server proxy.
- The ‘location /‘ block proxies all root requests to the ‘frontend_server‘ defined in pillar.

- Lines 20-24:
Configure the API proxy.
- The ‘location /api/‘ block forwards requests to the ‘backend_api‘ upstream group defined earlier.
- This ensures all API calls are distributed among the backend servers.

Overall, this template makes nginx act as a reverse proxy: sending frontend requests to the React server and API requests to a pool of backend services.

2.1.4 Screenshots

Figure 2.1: State applied from Minion

2.2 Pillar File and Using Values

2.2.1 Pillar - /srv/pillar/top.sls

```
1 base:
2   'nginx-vm':   # only apply to nginx server
3     - nginx
```

Listing 2.4: Top.sls

Explanation
- Line 1: Defines the ‘base‘ environment in the Salt top file.
- Line 2: Targets the minion with ID ‘nginx-vm‘.
- Line 3: Applies the ‘nginx‘ pillar file to that minion, ensuring the defined configuration is enforced.

2.2.2 Pillar - /srv/pillar/nginx.sls

```
1 nginx:
2   backend_servers:
3     - 192.168.64.30:5000
4     - 192.168.64.31:5000
5   frontend_server: 192.168.64.32:3000
6   server_name: bookstore.local
```

Listing 2.5: nginx.sls

Explanation
This pillar file defines reusable configuration values for nginx. Using a pillar allows central management and easy updates without modifying state files.
- Line 2:
Defines the root key ‘nginx‘, grouping all related settings.
- Lines 3-5:
‘backend_servers‘ lists the backend API servers by their IP and port (here two servers on port 5000).
- Line 6:
‘frontend_server‘ specifies the React frontend server running on port 3000.
- Line 7:
‘server_name‘ sets the domain (here `bookstore.local`) that nginx should respond to.

In short, this pillar makes it easy to inject environment-specific values (IPs, ports, domains) into the Jinja template, keeping states clean and flexible.

2.2.3 Screenshots

Figure 2.2: Pillar values applied

Chapter 3

Running Salts without Master

3.1 Explanation

1. Setup multi-master for minions. if one down then it can connect to different master.
2. Don't keep files like states and pillars on master. use gitfs fileserver backend to connect to git and use state and files from there.
3. If somehow all masters are down. Then, we can pull states and pillar on mininons from git and run Salt states locally on the minion using `--local` mode, without requiring a master connection.

3.2 Minion config - /etc/salt/minon

```
1 master:
2   - 192.168.64.34    # salt-master-vm
3   - 192.168.64.35    # salt-master-vm1
4 master_type: failover
5 master_alive_interval: 30
```

Listing 3.1: Multi-master setup on nginx-vm(Minion)

Explanation
This configuration defines how a Salt minion connects to multiple masters and manages failover.
- Line 1 :
'master': Lists the available master servers by their IPs. In this case, `192.168.64.34` and `192.168.64.35`.
- Line 4 :

‘master_type: failover‘: Ensures the minion connects to the first master in the list and only switches to the next if the current master becomes unavailable.
- Line 5 :
‘master_alive_interval: 30‘: Configures the interval (in seconds) at which the minion checks if the master is alive.

This setup provides **high availability**: if one master fails, the minion can continue functioning by connecting to the backup master.

3.3 Master config - /etc/salt/master

```
1  fileserver_backend:
2  - gitfs
3
4  gitfs_remotes:
5  - https://github.com/siddharth5intern-collab/salt.git:
6  - base: main
7  - root: states      # states/ folder inside repo
8
9  ext_pillar:
10 - git:
11 - main https://github.com/siddharth5intern-collab/salt.git:
12 - root: pillar    # use pillar/ folder
```

Listing 3.2: Connecting file server to github

Explanation
This configuration integrates Salt with a Git repository for both states and pillar data. It allows managing infrastructure as code directly from version control.

- Line 1:
‘fileserver_backend‘ specifies which backend Salt should use for serving state files. Here, it is set to ‘gitfs‘.
- Line 4: ‘gitfs_remotes‘ lists Git repositories that contain Salt states.
- Lines 5-7: The repository `https://github.com/siddharth5intern-collab/salt.git` is used.
- ‘base: main‘ tells Salt to use the `main` branch as the **base environment**.
- ‘root: states‘ restricts Salt to use only the `states/` directory inside the repo.
- Line 9: ‘ext_pillar‘ defines external pillar sources.
- Lines 10-12: Pillar data is also pulled from the same Git repo.

- ‘main‘ refers to the branch used.
- ‘root: pillar‘ restricts Salt to load pillar files only from the `pillar/` directory.

With this setup, both **states** and **pillars** are version-controlled in Git, enabling collaborative development and consistent configuration management.

3.4 Salt call locally

```
# Running state locally on Minion
sudo salt-call --local state.apply nginx
```

Listing 3.3: Running Salt Locally

3.5 Screenshots

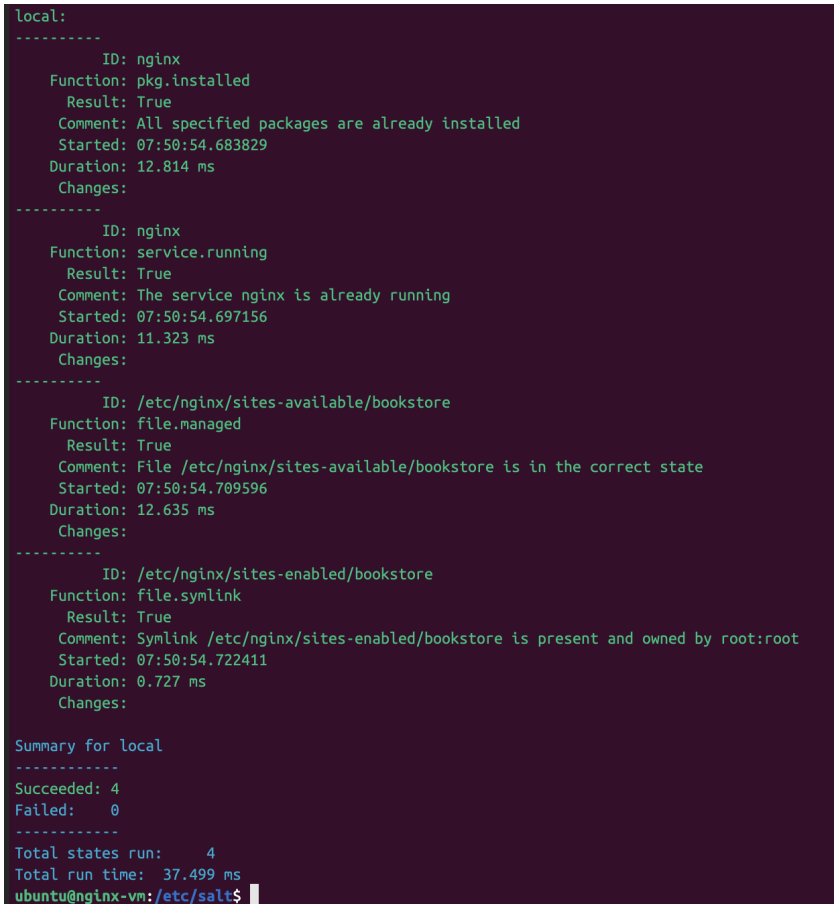

Figure 3.1: Salt state applied without Master

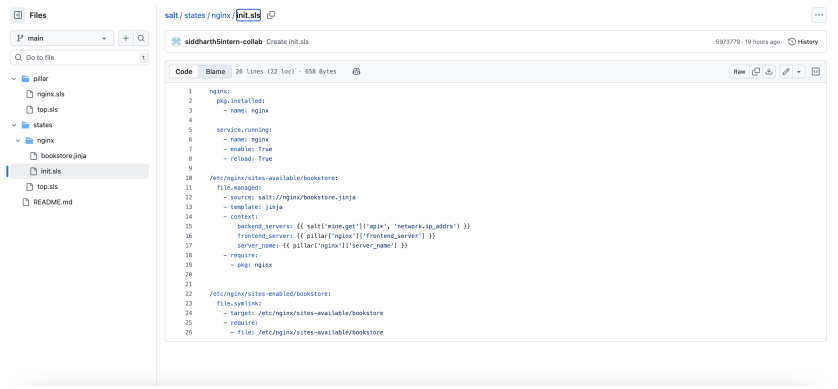

Figure 3.2: Salt and pillar files on github

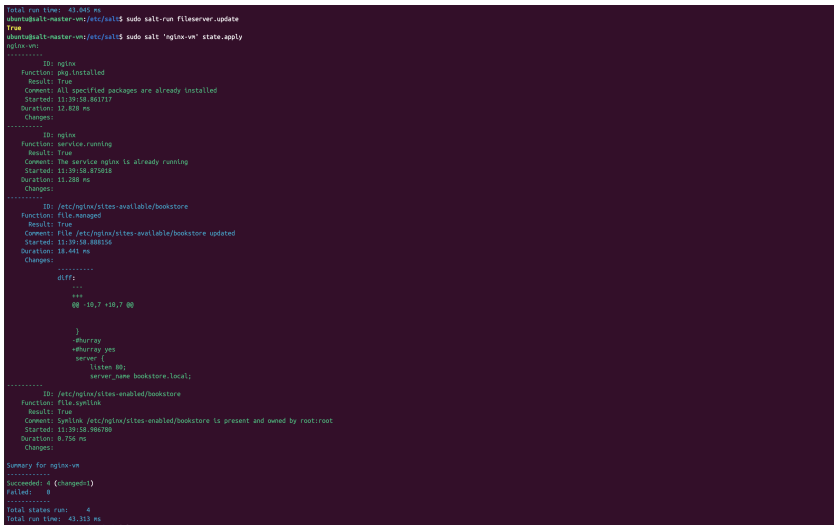

Figure 3.3: Running salt and pillar from git

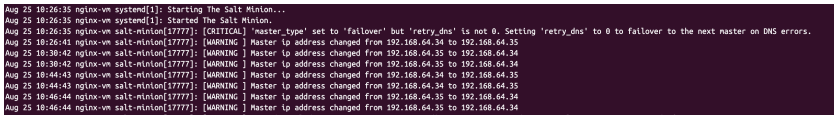

Figure 3.4: Minion connecting to another available master