Infrastructure Automation

Defending your systems at scale

Defensive Cyber Security Interest Group (DCIG)
Spring 2023

Why automate?

- A new CVE has been discovered and you must apply a mitigation to several hundred systems immediately
- Reduces chance of error, ensures all systems are running the same configuration
- Easily audit and keep record of all changes made to systems
- Patching can be a 5-minute code change and deploy rather than an all-day exercise

What's on the market?

- Lots of options!
- Each have their own pros/cons
- Master vs masterless?
- Agent vs agentless?
- Enterprise vs community driven?
- **Try a few options until you find the tool you like the most**

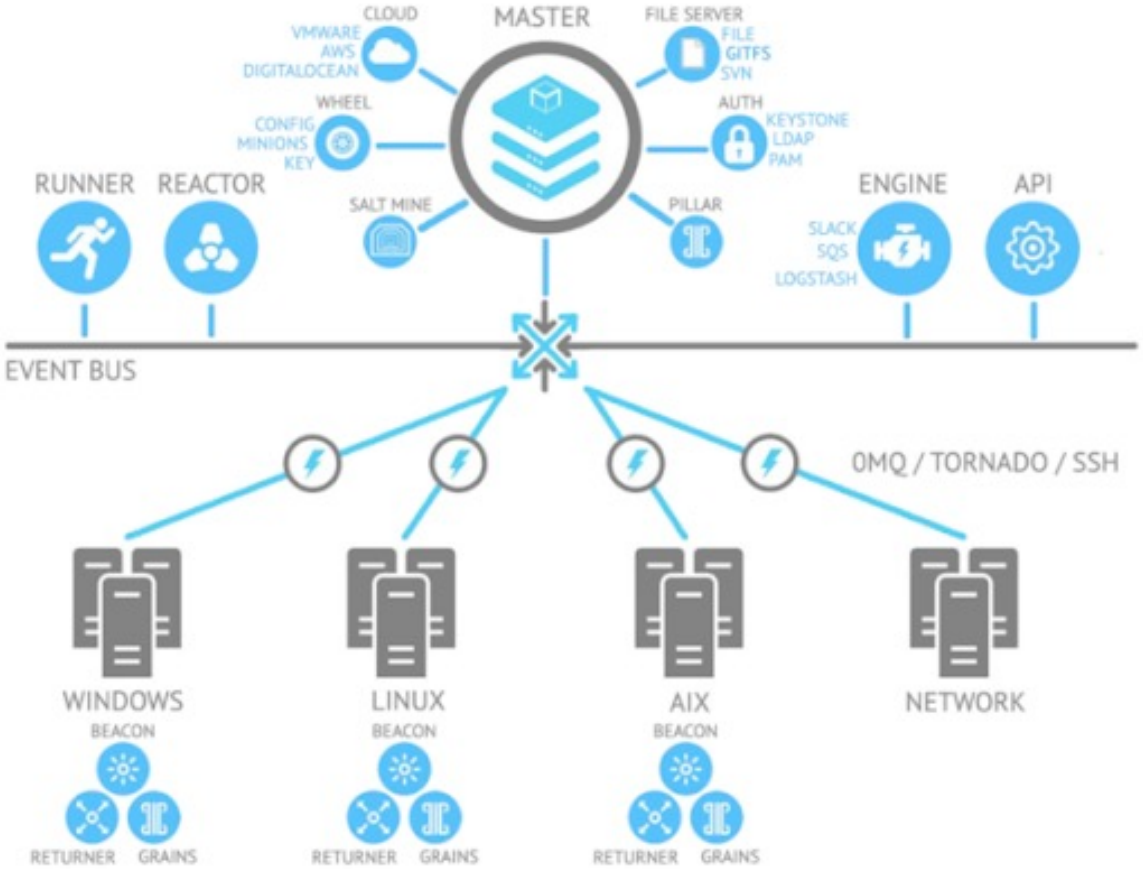
Why Salt Stack?

- Supports configuration management, automation, provisioning, and orchestration
- Allows for both SSH and agent-based deployments (Minions)
- Advanced templating through Jinja
- Active open-source community
- Enterprise support through VMware Aria Automation (SaltStack Config)

Salt Stack – How does it work?

- One system running the ***Salt Master***, all other systems running a ***Salt Minion***
- Master pushes a job to be executed, minions subscribe to those jobs
- Indicate which minions should execute through ***targets***, which can include minion ***grains***
- Example job:

salt –G 'os:Windows' pkg.install firefox



https://docs.saltproject.io/en/3004/topics/salt_system_architecture.html

So many possibilities!

- Jobs can be run **ad-hoc** (such as the previous example of installing Firefox)
- **States** are YAML files that specify what state a target system should be in (such as a managed file, installed package, or configuration)
- **Pillars** can be used to distribute secrets to specific systems
- **Beacons** and **Reactors** can listen and respond to system events

Sample IP Tables State

```
iptables:
  pkg:
    - installed
  service:
    - running
    - watch:
      - pkg: iptables
      - file: iptables
  file:
    - managed
    - source: salt://iptables/iptables
    {% if grains['os'] == 'CentOS' or grains['os'] == 'Fedora' %}
    - name: /etc/sysconfig/iptables
    {% elif grains['os'] == 'Arch' %}
    - name: /etc/conf.d/iptables
    {% endif %}
```

Salt Master configuration

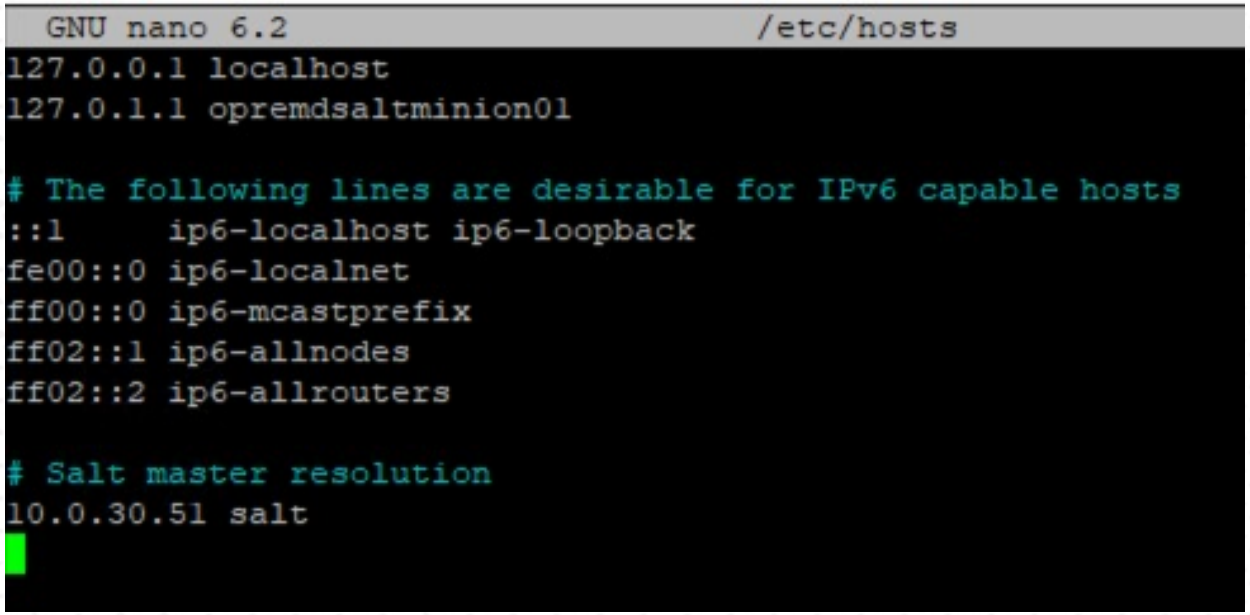
- Install curl (**sudo apt-get install curl**)
- Add salt onedir package repository
- Install salt modules (**sudo apt-get install salt-master salt-minion**)
- Enable salt modules (**sudo systemctl enable salt-master salt-minion**)
- https://docs.saltproject.io/salt/install-guide/en/latest/topics/install-by-operating-system/debian.html#install-onedir-packages-of-salt-on-debian-10-buster

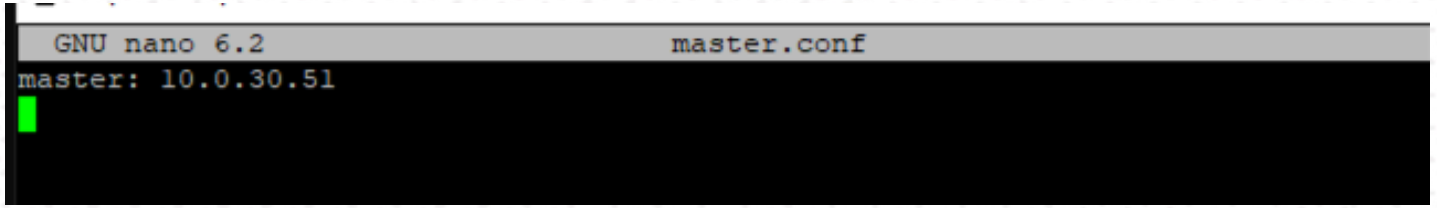
Salt Minion configuration

- This is the system that we want to manage
- Download same one-dir package in last slide, but only install and enable the **salt-minion**
- https://docs.saltproject.io/salt/install-guide/en/latest/topics/install-by-operating-system/debian.html#install-onedir-packages-of-salt-on-debian-10-buster

Salt Minion configuration (cont.)

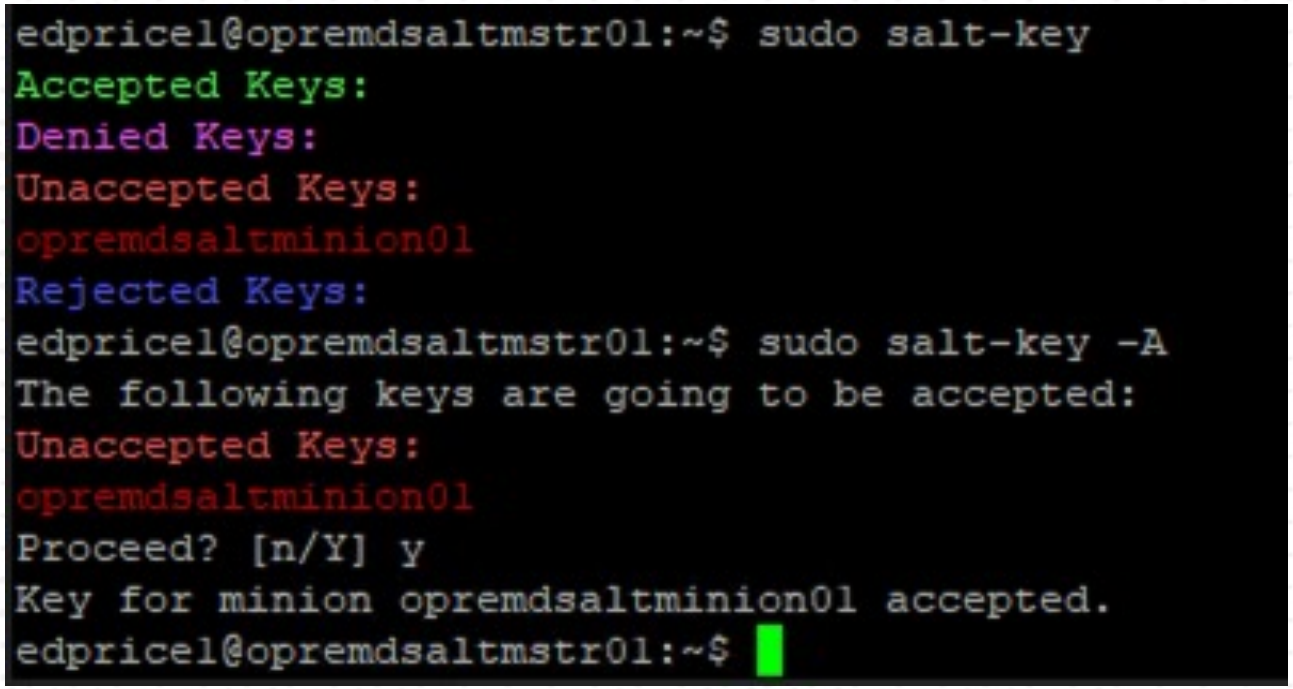
- We need to configure the minion to listen to our salt master.
- This can be done by setting a DNS record (**/etc/hosts**) for "**salt**" to the salt master
- Alternatively, create a custom minion configuration (**/etc/salt/minion.d/minion.conf** on Linux)
- Restart salt minion (**sudo systemctl restart salt-minion**)





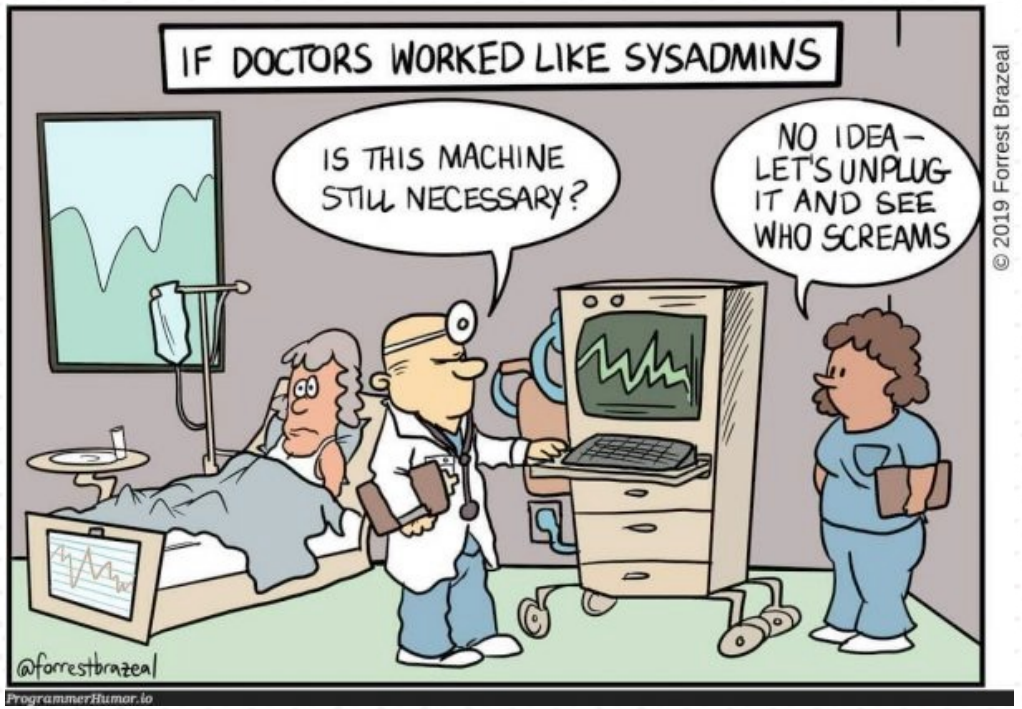
Salt Master – Accept minion keys

- At this point, the salt minion should reach out to salt master with its public key
- On the salt master, run **sudo salt-key** to see a list of minion keys
- Accept all incoming keys with **sudo salt-key –A**
- ***On production systems salt key fingerprints should be verified before being accepted!***



Salt Master – Test minion connection

- We now have a working connection between the salt master and minion
- To ping all connected minions, run **sudo salt '*' test.ping**
- Commands can be run **ad-hoc** without any further configuration **sudo salt '*' cmd.run 'echo hello world!'**




```
edpricel@opremdsaltmstr01:~$ sudo salt '*' cmd.run 'echo Hello World!'
opremdsaltminion01:
    Hello World!
```

Salt Master – Creating our first state

- The real power with Salt comes with applying **States** through **Infrastructure as Code**
- Minions can pull files from the salt fileserver located in **/srv/salt** on the salt master, and **salt://** in state files and commands
- This state will ensure that UFW is installed on all systems

```
sudo mkdir /srv/salt
sudo vi /srv/salt/ufw.sls

ufw:
  pkg.installed

sudo salt '*' state.apply ufw
```

Choose your journey

- Now that you're a salty sysadmin, choose your path on what to learn next!

Advanced targeting with grains and RegEx (Slide 17)

Pulling security, compliance, and operations data from systems (Slide 21)

Hardening my infrastructure through Salt States (Slide 26)

Sharing secret information with Salt Pillars (Slide 29)

Choose your journey

Advanced minion targeting

Salt Targeting Grains

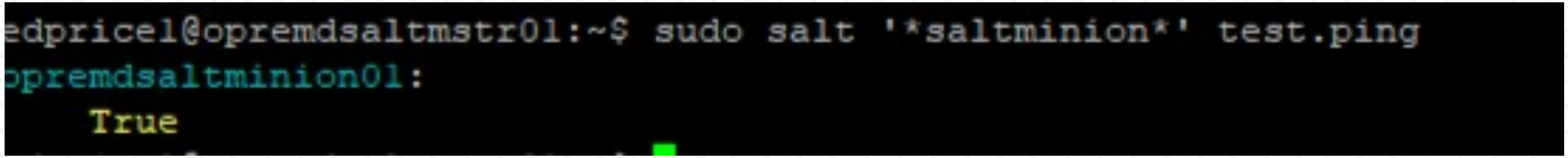
- https://docs.saltproject.io/en/latest/topics/targeting/index.html
- https://docs.saltproject.io/en/latest/topics/grains/index.html
- Salt Grains are system properties defined by the minion such as OS, IP address, etc.
- Custom grains can also be added
- Target with –**G** flag
  sudo salt –G 'os_family:Debian' test.ping

sudo salt '*' grains.items

```
os:
    Ubuntu
os_family:
    Debian
osarch:
    amd64
oscodename:
    jammy
osfinger:
    Ubuntu-22.04
osfullname:
    Ubuntu
osmajorrelease:
    22
osrelease:
    22.04
osrelease_info:
    - 22
    - 4
path:
    /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin
pid:
    25004
productname:
    VMware20,1
ps:
    ps -efHww
pythonexecutable:
    /usr/bin/python3
pythonpath:
    - /usr/bin
    - /usr/lib/python310.zip
    - /usr/lib/python3.10
    - /usr/lib/python3.10/lib-dynload
    - /usr/local/lib/python3.10/dist-packages
    - /usr/lib/python3/dist-packages
```

Salt Targeting Globbing

- https://docs.saltproject.io/en/latest/topics/targeting/globbing.html
- Each minion has a unique identifier, which is the FQDN by default.
- Minion IDs can be targeted through **globbing**
- Example in my home lab:
  sudo salt '*saltminion*' test.ping



Salt Targeting RegEx

- https://docs.saltproject.io/en/latest/topics/targeting/globbing.html#regular-expressions
- Regular Expressions (Globbing on steroids) can also be used to target minions with the **–E** flag
- On my homelab:
  sudo salt –E '[a-z]*[0-9]{2}.*' test.ping

`.`	any character except newline
`\w \d \s`	word, digit, whitespace
`\W \D \S`	not word, digit, whitespace
`[abc]`	any of a, b, or c
`[^abc]`	not a, b, or c
`[a-g]`	character between a & g
Anchors	
`^abc$`	start / end of the string
`\b \B`	word, not-word boundary
Escaped characters	
`\. \* \\`	escaped special characters
`\t \n \r`	tab, linefeed, carriage return
Groups & Lookaround	
`(abc)`	capture group
`\1`	backreference to group #1
`(?:abc)`	non-capturing group
`(?=abc)`	positive lookahead
`(?!abc)`	negative lookahead
Quantifiers & Alternation	
`a* a+ a?`	0 or more, 1 or more, 0 or 1
`a{5} a{2,}`	exactly five, two or more
`a{1,3}`	between one & three
`a+? a{2,}?`	match as few as possible
`ab|cd`	match ab or cd

```
edpricel@opremdsaltmstr01:~$ sudo salt -E '[a-z]*[0-9]{2}.*' test.ping
[sudo] password for edpricel:
opremdsaltminion01:
    True
```

Choose your journey

Pulling compliance, security, and operational data from systems with Salt

Pulling data from minions

- Salt can be used to pull information from systems for many different purposes
- Security (password complexity, CVE exploitability)
- Compliance (open ports, allowed users, running backup services)
- Operations (System usage, running services, networking configurations)
- Data can be placed in central location on salt master to be exported for processing

Compliance Data

- https://github.com/evynprice/dcig-salt/tree/main/states/comp
- Example states in states/comp
- The linux_comp state will check netstat, apt list, /etc/passwd, lsb_release, and /etc/rsyslog.conf and pipe them to a temporary directory.
- That directory is then uploaded to the salt master for centralized storage
- The orchestration file is intended to be run from your salt master. It takes the files from minion cache and puts them in a more permanent directory.
- **The master configuration file_recv: True must be enabled to allow minions to push data to master.**
- **salt 'master*' state.apply comp.orch**

Operations Data

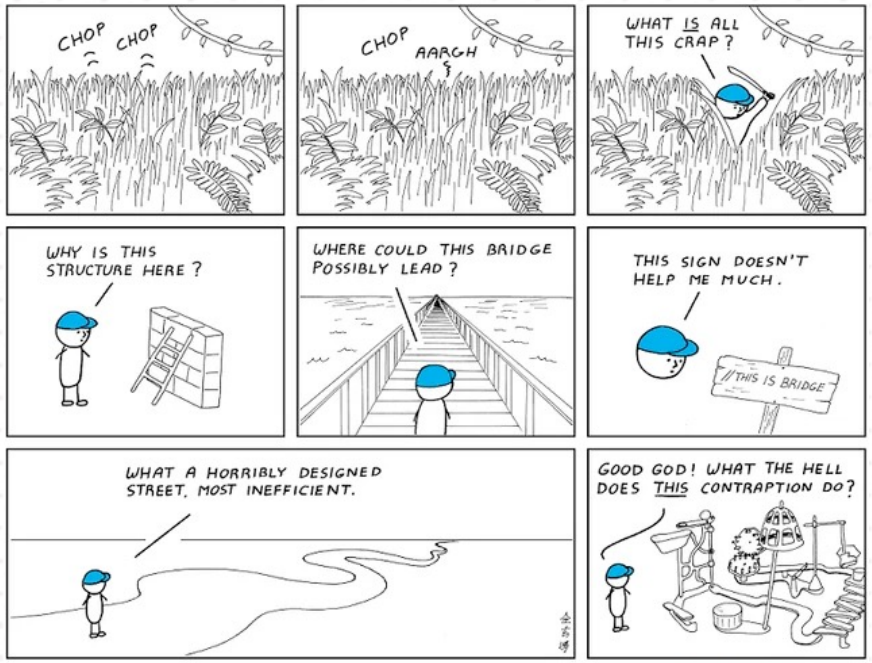
- https://github.com/evynprice/dcig-salt/tree/main/states/ops
- Example states in states/ops
- Similar to compliance checks but includes checks that may only be interesting to operators
- Includes network information, logged in users, and system resource consumption.
- **Salt 'master*' state.apply ops.orch**

Choose your journey

Hardening your infrastructure at scale with Salt

Hardening infrastructure with Salt

- Salt can be used to apply hardening to infrastructure
- Install packages, manage configurations, add and remove users, even modify the registry on Windows
- Can simply run existing Powershell scripts or create salt **States** to apply hardening on a routine basis



I hate reading other people's code.

Salt installing packages

- Package management with Salt is different with Linux and Windows based packages. We will use the native pkg state module. Windows users can use the winrepo-ng package manager.
- https://github.com/evynprice/dcig-salt/blob/main/states/hardening/linux_packages.sls

16 lines (14 sloc) | 151 Bytes

```
ntp:
  pkg:
    - latest
  service:
    - running
    - enable: True

fail2ban:
  pkg:
    - latest
  service:
    - running

telnet:
  pkg:
  - purged
```

Image Prompt

boopednose

This is the installation wizard

Salt manage configurations

- Along with installing software, salt can be used to manage the configurations for services.
- The easiest way to do this is to use the **file.managed** state. This will ensure that a local file contents stay the same as a configuration stored on the Salt Master.
- https://github.com/evynprice/dcig-salt/blob/main/states/hardening/ssh_config.sls

```
1  sshd_conf:
2    file:
3      - managed
4      - name: /etc/ssh/sshd_config
5      - source: salt://states/hardening/files/sshd_config
6      - user: root
7      - group: root
8      - mode: 644
```

Choose your journey

Sharing secret information with Salt Pillars

Why do we need pillars anyway?

- https://docs.saltproject.io/en/3004/topics/salt_system_architecture.html#open-event-system-event-bus
- The Salt architecture uses an **Open Event Bus.** This means that traffic from the salt master to the minions is **open**, but the response from the minion to the master is **private.**
- This allows for efficiency, but secrets cannot be shared through normal jobs
- **Pillars** are the solution to this, as minions can only see the data they are explicitly permitted.

Creating pillars

- Follow this tutorial to create a simple pillar on the salt master: https://docs.saltproject.io/en/latest/topics/tutorials/pillar.html
- https://github.com/evynprice/dcig-salt/tree/main/pillar
- Pillar files should be placed in /srv/pillar, state files in /srv/salt
- Test the pillar with **salt '*' pillar.items**

To start setting up the pillar, the /srv/pillar directory needs to be present:

```bash
mkdir /srv/pillar
```

Now create a simple top file, following the same format as the top file used for states:

`/srv/pillar/top.sls`:

```yaml
base:
  '*':
    - data
```

This top file associates the data.sls file to all minions. Now the `/srv/pillar/data.sls` file needs to be populated:

`/srv/pillar/data.sls`:

```yaml
info: some data
```

To ensure that the minions have the new pillar data, issue a command to them asking that they fetch their pillars from the master:

```bash
salt '*' saltutil.refresh_pillar
```

Now that the minions have the new pillar, it can be retrieved:

```bash
salt '*' pillar.items
```

The key `info` should now appear in the returned pillar data.

Using pillars in States

- Pillars are most used in Salt **States** to send targeted data only to specific minions
- https://github.com/evynprice/dcig-salt/blob/main/states/pillar_test.sls
- This example state will echo the pillar data created in the previous slide

```
Summary for opremdsaltmstr01
------------
Succeeded: 1 (changed=1)
Failed:    0
------------
Total states run:     1
Total run time:   2.778 ms
opremdsaltminion01:
----------
          ID: pillar_test
    Function: cmd.run
        Name: echo DCIG is awesome!
      Result: True
     Comment: Command "echo DCIG is awesome! " run
     Started: 03:46:16.918759
    Duration: 18.65 ms
     Changes:
              ----------
              pid:
                  47535
              retcode:
                  0
              stderr:
              stdout:
                  DCIG is awesome!
```

Resources

- https://docs.saltproject.io/en/latest/contents.html
- https://docs.saltproject.io/en/latest/topics/index.html
- https://docs.saltproject.io/en/latest/topics/salt_system_architecture.html
- https://docs.saltproject.io/en/latest/ref/configuration/master.html
- https://docs.saltproject.io/en/latest/ref/configuration/minion.html
- https://docs.saltproject.io/en/latest/topics/jobs/index.html
- https://docs.saltproject.io/en/latest/ref/file_server/index.html
- https://docs.saltproject.io/en/latest/topics/grains/index.html
- https://docs.saltproject.io/en/latest/topics/states/index.html
- https://www.vmware.com/support/acquisitions/saltstack.html