Software-defined Datacenter Maintenance
No more sleepless nights and long weekends when doing maintenance
CAS1172

Raine Curtis
N America Core Services Team Lead, Consultant
rcurtis@suse.com
SUSE

Stephen Nemeth
Senior Architect
Stephen.nemeth@suse.com
SUSE
1 Agenda

This session will present some best practices for datacenter management and automation.

The technologies used in this discussion are:

• SUSE Manager
• Salt
1.1 SUSE Manager

- Patching lifecycle
- Configuration management
- API
1.2 Salt

- SUSE Manager/Salt Features
- Salt Command-line
- Salt REST API
2 SUSE Manager
2.1 Using SUSE Manager for Maintenance

- Patching lifecycle management
- Configuration management
- API
2.2 Patching Lifecycle Management

It is a best practice to have managed systems subscribe from a controlled channel set.

- Typically this a point-in-time snapshot of the distribution channels
- There are two main best practices:
  - Consolidated organizational base channel set
  - Distributed organizational base channel set
Fig. 2.1: Consolidated Organizational Base Channel Set
Fig. 2.2: Distributed Organizational Base Channel Set

Copyright 2019 SUSE
2.3 Configuration Management

• Use state channels
• Learn forms and formulas
• Leverage configuration scope
2.3.1 State Channels

Use Salt... your life will be easier, and you’ll have lots of friends!

State channels are Salt states managed in SUSE Manager

- Allows for UI for state management and editing
- Supports configuration file management
- Provides version control of states
Fig. 2.3: State channel
2.3.2 Forms and Formulas

*Pillar management was never easier!

- Create your own forms from simple yaml
- Pillar data is automatically created
- Forms may be assigned to different systems with different values (pillar)
Fig. 2.4: SUSE Manager Forms and Formulas
2.4 Configuration Scope

**States** - may be assigned at the following levels:

- Organization (all systems)
- System groups
- Individual system

**Forms and Formulas** - may be assigned at:

- System groups
- Individual system
2.5 Using the API (commandline)

There is a great deal of functionality outside the SUSE Manager UI. Most APIs may be called with the `spacecmd` utility.

Manage all types of SUSE Manager objects:

- Software Channels
- Systems
- Activation keys
- Users
- System Set Manager
- …
2.6 API Example

The following example shows how to easily manage multiple systems at once at the commandline using the System Set Manager(ssm) object.

Add systems beginning with **dev** to ssm:

```
spacecmd ssm_add dev*
```
View the systems now in the ssm:

```bash
spacecmd ssm_list
dev01.example.com
dev02.example.com
dev03.example.com
```

Add a new `apps` channel to all systems in the ssm:

```bash
spacecmd system_addchildchannels ssm apps
```

Clean up the ssm:

```bash
spacecmd ssm_remove dev*
```
3 Salt
3.1 Using Salt for Maintenance

Working with Salt directly provides additional functionality and automation.

You can interact with Salt directly by:

- Working at the command-line
- Using the Salt API
3.2 Salt Commands

The following are common Salt commands:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>salt-key</td>
<td>Manage Salt keys</td>
</tr>
<tr>
<td>salt-call</td>
<td>Call Salt functions from execution modules on minions</td>
</tr>
<tr>
<td>salt</td>
<td>The main remote execution command. Only runs on master</td>
</tr>
<tr>
<td>salt-run</td>
<td>Sends jobs to the master. e.g. orchestration states</td>
</tr>
</tbody>
</table>

Fig. 3.1: Salt commands
3.3 Location of Commands

Fig. 3.2: Salt commands
3.4 Salt Key Management

`salt-key` executes simple management of Salt server public keys used for authentication.

View keys:

```
salt-key
# or..
salt-key -L
```

**Note:** In salt **uppercase** arguments mean more than one minion.
3.4.1 Accepting Multiple Keys at Once

Keys may be accepted in the following ways:

- in **SUSE Manager**, under **Salt > Keys**
- the **salt-key** command
- using a Salt reactor
- using the API
- setting the Salt master option `auto_accept: True` (not recommended)

When migrating multiple hosts, accepting keys in bulk form is valuable:

```
# one system
salt-key -a minion01

# multiple systems at once
salt-key -y -A
```
3.5 Sending Commands to Minions

Instead of logging to each system, a salt module may be run on many systems simultaneously from the Salt master.

Remote execution in salt is performed with the `salt` command.

The syntax is:

```
salt [options] '<target>' <module.function> [arguments|keyword arguments]
```

An example of targeting minions:

```
salt '*' cmd.run 'cat /etc/resolv.conf'
```
3.5.1 Using Salt Grains

View system grains:

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

Target by system a grain:

```
salt -G 'oscodename:SUSE Linux Enterprise Server 12 SP3' grains.item num_cpus
```

```
server1.example.com:
    ----------
    num_cpus:
        1
build.example.com:
    ----------
    num_cpus:
        1
...
```
3.6 Salt Rest API

In this section you will gain an understanding of the Salt RESTful API. The topics covered will be:

• Turning on the API
• Using the LocalClient through REST
• Using the RunnerClient through REST
3.7 Overview

The Salt API project is a modular interface on top of Salt that can provide a variety of entry points into a running Salt system.

It can start and manage multiple interfaces allowing a REST API to co-exist with XMLRPC or even a Websocket API.

By default the Salt API was developed to act as the communication layer for the Salt web UI, but it can also be used as a means to access Salt from other remote apis as well.

Salt API is closely tied to the external authentication system and uses TLS for all connections.
3.8 SUSE Manager and Salt API

SUSE Manager uses the Salt API.

The REST configuration shipped is:

```conf
# file: /etc/salt/master.d/susemanager.conf (fragment)
# Setup cherrypy
rest_cherrypy:
  port: 9080
  host: 127.0.0.1
  collect_stats: false
  disable_ssl: true
  ssl_crt: /etc/pki/tls/certs/spacewalk.crt
  ssl_key: /etc/pki/tls/private/spacewalk.key
```

Copyright 2019 SUSE
3.8.1 External Authentication

The REST interface requires the external authentication system to be defined.

The External Authentication configuration shipped is:

```yaml
# Setup API authentication
external.auth:
  auto:
    admin:
      - .*
      - '@wheel'
      - '@runner'
      - '@jobs'
```
3.8.2 Managing Salt API

If ever needed, the Salt API may be restarted by executing:

```
systemctl salt-api restart
```

**Warning:** Be very careful changing any of these settings so that SUSE Manager integration is not broken.
3.9 RESTful LocalClient Call

Both the `curl` command line and native programming languages can be used to access this web service API.

Supports token passing or cookie storage for session management.

Logging into the API will return a token that must be referenced for all subsequent calls.
Perform a API RESTful login.

```
curl -sSk http://localhost:9080/login \
-H 'Accept: application/x-yaml' \
-d username=saltdev \
-d password=saltdev \
-d eauth=auto
```

return:
- eauth: auto
  expire: 1437104145.631494
  perms:
    - '@runner'
    - '@wheel'
    - '*'
  start: 1437060945.6314919
  token: 13279c452c51fbbf00c9f32d1e62c79be8db262c
  user: saltdev
So to run a `test.ping` to all hosts using the token you would make the call as:

curl -sSk http://localhost:9080 \
-H 'X-Auth-Token:4d3f6b45da944946a92b7d1efee5a382cd0160f1' \
-d client=local \
-d tgt='*' \
-d fun=test.ping
Cookies can be used instead of passing the token each time:

```
curl -sSk http://localhost:9080/login \\n-H 'Accept: application/x-yaml' \\
-d username=saltdev \\
-d password=saltdev \\
-d eauth=auto \\
-c cookies.txt
```
Now all subsequent calls can use cookies such as in this call to get the grain information from

```
curl -sSk http://localhost:9080 \
  -b cookies.txt \
  -H 'Accept: application/x-yaml' \
  -d client=local \
  -d tgt=websvr1 \
  -d fun=grains.items
```
3.9.1 State Runs through Salt API

A call can be made to run a state dry-run with `state.sls` for the `apache` state:

```
curl -sSk http://localhost:9080 -b cookies.txt
-H 'Accept: application/x-yaml' -d client=local
-d tgt='minion01' -d fun=state.sls -d arg=httpd
-d arg='test=True'
```

A call can be made to run the state with `state.sls`:

```
curl -sSk http://localhost:9080 -b cookies.txt
-H 'Accept: application/x-yaml' -d client=local
-d tgt='minion01' -d fun=state.sls -d arg=httpd
```
3.10 Using API Registered URLs

The Salt API has registered URLs, or hooks to shortcut access to information.

The following shows examples of using registered URLs.
3.10.1 Listing Minions

Using the `/minions` hook:

```bash
# Show all minions and grains
curl -sSk http://localhost:9080/minions -b cookies.txt
   -H 'Accept: application/x-yaml'

# Show a specific minion
curl -sSk http://localhost:9080/minions/minion01 -b cookies.txt
   -H 'Accept: application/x-yaml'
```
3.10.2 Listing Keys

Using the /keys hook:

```bash
# List all keys
curl -sSk http://localhost:9080/keys -b cookies.txt
   -H 'Accept: application/x-yaml'

# List a specific minion's keys
curl -sSk http://localhost:9080/keys/minion01 -b cookies.txt
   -H 'Accept: application/x-yaml'
```
3.10.3 List Jobs

Using the **jobs** hook:

```bash
# List all jobs
curl -sSk http://localhost:9080/jobs -b cookies.txt
   -H 'Accept: application/x-yaml'
```
3.10.4 Logging Out

Logging out:

curl -sSk http://localhost:9080/logout -H 'Accept: application/x-yaml'
   -b cookies.txt
3.11 Registered URLs in the API

There are special registered URIs for accessing common data:

<table>
<thead>
<tr>
<th>URL Hook</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>The primary entry point to Salt's REST API</td>
</tr>
<tr>
<td>/login</td>
<td>Authenticate against Salt's eauth system</td>
</tr>
<tr>
<td>/logout</td>
<td>Destroy the currently active session and expire the session cookie</td>
</tr>
<tr>
<td>/minions/(min)</td>
<td>Convenience URLs for working with minions</td>
</tr>
<tr>
<td>/jobs</td>
<td>Convenience URL for getting lists of previously run jobs or job</td>
</tr>
<tr>
<td>/run</td>
<td>Class to run commands without normal session handling</td>
</tr>
<tr>
<td>/events</td>
<td>Expose the Salt event bus</td>
</tr>
<tr>
<td>/hook</td>
<td>A generic web hook entry point that fires an event on Salt's event bus</td>
</tr>
<tr>
<td>/keys/(key)</td>
<td>List all RSA keys or show a specific key</td>
</tr>
<tr>
<td>/ws</td>
<td>Open a WebSocket connection to Salt's event bus</td>
</tr>
<tr>
<td>/stats</td>
<td>Expose statistics on the running CherryPy server</td>
</tr>
</tbody>
</table>

Fig. 3.3: Salt API Registered Hooks
4 Demo

- Demo of using SUSE Manager API
- Demo of using Salt API
5 Thanks