Simplifying Pillar Management with Forms and Formulas in SUSE Manager

TUT1053

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

Patrick Swartz
Sales Engineer
patrick.swartz@suse.com
SUSE

1 Agenda

- The Salt Pillar system
- SUSE Manager pillar integration
- SUE Manager Forms and Formulas

Copyright 2017 SUSE

Note: This session assumes that you have a basic knowledge of Salt and SUSE Manager.

Copyright 2017 SUSE

2 Pillars of Salt

Copyright 2017 SUSE

2.1 Objectives

In this section we will discuss:

- Storing Static Data in the Pillar
- Declaring Master Pillar Data
- Pillar Dictionary Merging
- Including Other Pillars
- In-Memory Pillar Data vs. On-Demand Pillar Data
- Viewing Pillar Data
- Viewing Pillar Errors

Copyright 2017 SUSE

2.2 Storing Static Data in the Pillar

Pillar is an interface for Salt designed to offer global values that can be distributed to minions.

- The most common type of information stored in pillar are configuration items.
- Pillar data is managed in a similar way as the Salt State Tree.
- Pillar was added to Salt in version 0.9.8

Copyright 2017 SUSE

2.3 Key Concepts of Pillar

Pillar data is compiled on the master.

- Pillar data for a given minion is only accessible by the minion for which it is targeted in the pillar configuration.
- This makes pillar useful for storing sensitive data specific to a particular minion.
- A session (AES) key is used to encrypt pillar data when a minion requests a refresh.

Copyright 2017 SUSE

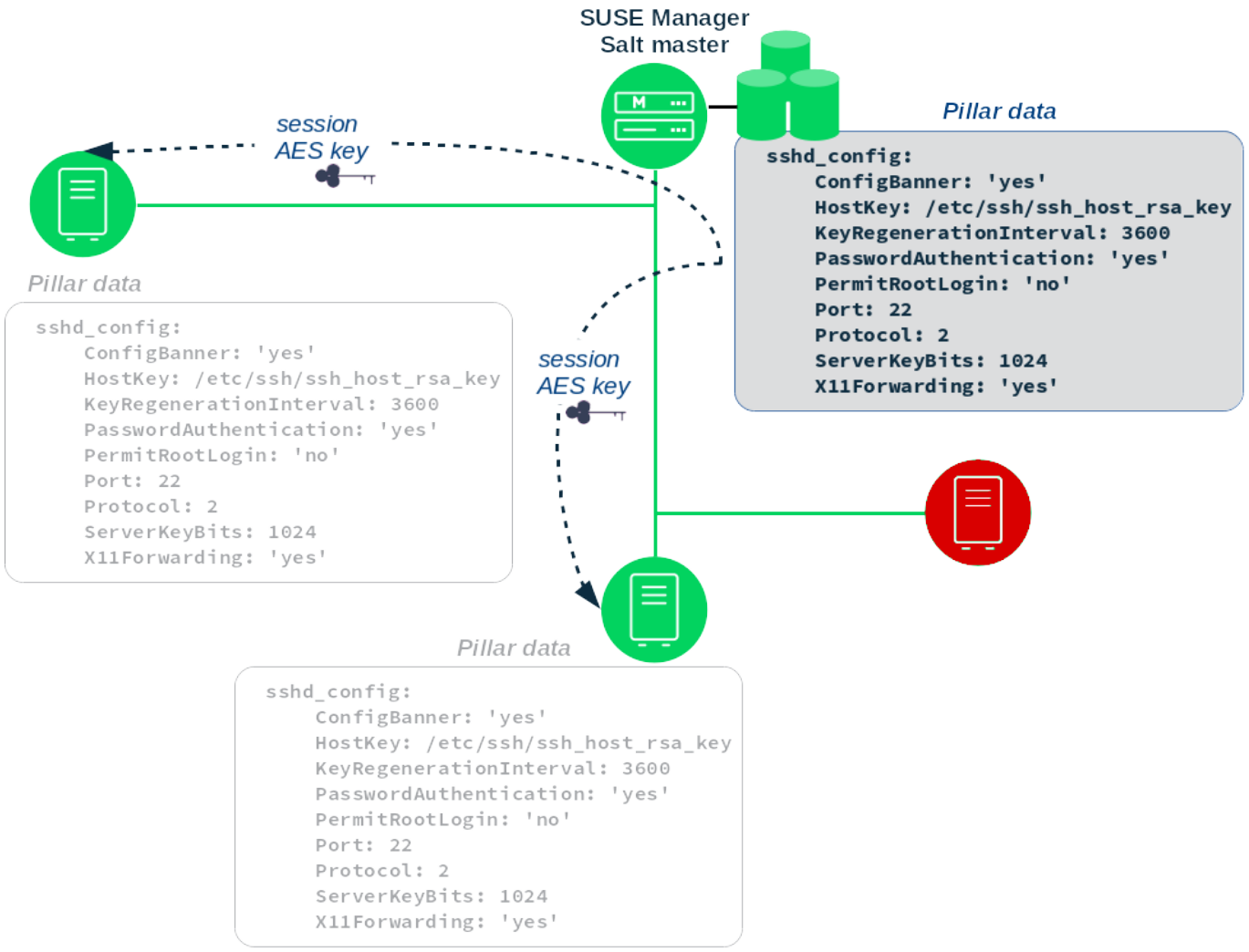

Fig. 2.1: Salt Pillar Session Keys

Copyright 2017 SUSE

2.4 Declaring Master Pillar Data

The Salt Master server maintains a `pillar_roots` setup that matches the structure of the `file_roots` used in the Salt file server.

- Like `file_roots`, the `pillar_roots` option maps environments to directories.
- The pillar data is then mapped to minions based on matchers in a top file which is laid out in the same way as the state top file.
- Salt pillars can use the same matcher types as the standard top file.
- `pillar_roots` is configured just like `file_roots`.

Copyright 2017 SUSE

2.5 Pillar Roots

For example here is a pillar top file:

```
pillar_roots:
  base:
    - /srv/pillar
```

This example configuration declares that the base environment will be located in the `/srv/pillar` directory.

Important: It must not be in a subdirectory of the state tree.

Copyright 2017 SUSE

2.6 Example Pillar

The following example pillar contains configuration information for NTP:

`/srv/pillar/ntp_us.sls`

```
ntp:
  server: us.pool.ntp.org
  options: iburst
```

Copyright 2017 SUSE

The following example pillar contains configuration information for NTP:

`/srv/pillar/ntp_eu.sls`

```
ntp:
  server: europe.pool.ntp.org
  options: iburst
```

Copyright 2017 SUSE

2.7 Pillar Top File

The to file used matches the name of the top file used for States, and has the same structure:

`/srv/pillar/top.sls`

```
base:
  'location:us':
    -match: grain
    - ntp_us
  'location:eu':
    -match: grain
    - ntp_eu
```

Copyright 2017 SUSE

Note: Pillar files are applied in the order they are listed in the top file.

Therefore conflicting keys will be overwritten in a ‘last one wins’ manner!

For example, in the above scenario conflicting key values in `services` will overwrite those in `packages` because it’s at the bottom of the list.

Copyright 2017 SUSE

2.8 Viewing Pillar Data

To view pillar data, use the `pillar` execution module.

This module includes several functions, each of them with their own use. These functions include:

- `pillar.items` - Compiles a fresh pillar dictionary and returns it, leaving the in-memory pillar data untouched.

  If pillar keys are passed to this function however, this function acts like `pillar.item` and returns their values from the in-memory pillar data.

Copyright 2017 SUSE

2.8.1 Viewing Raw Pillar Data

- `pillar.item` - Retrieves the key/value of one or more keys from the in-memory pillar data.
- `pillar.get` - Gets the value of pillar. Returns only the value, and not the `key:value` pair like `pillar.item`.
- `pillar.raw` - Like `pillar.items`, it returns the entire pillar dictionary, but from the in-memory pillar data instead of compiling fresh pillar data.

Copyright 2017 SUSE

2.9 Refreshing Pillar Data

The in-memory pillar data is generated on minion start, and can be refreshed using the `saltutil.refresh_pillar` function:

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

This function triggers the minion to asynchronously refresh the in-memory pillar data and will always return `None`.

Copyright 2017 SUSE

2.10 Triggered Pillar Refresh

In contrast to in-memory pillar data, certain actions trigger pillar data to be compiled to ensure that the most up-to-date pillar data is available.

These actions include:

- Running states
- Running `pillar.items`

So, if pillar data is modified, and then states are run, the states will see the updated pillar data.

Copyright 2017 SUSE

2.11 In-memory Pillar

If a state has not run since the last pillar change then the following functions will not see this data unless refreshed using `saltutil.refresh_pillar`.

- `pillar.item`
- `pillar.get`
- `pillar.raw`

Copyright 2017 SUSE

2.12 Master Provided Pillar Error

By default if there is an error rendering a pillar, the detailed error is hidden and replaced with:

```
Rendering SLS 'my.sls' failed. Please see master log for␣
↪details.
```

The error is protected because it's possible to contain templating data which would give that minion information it shouldn't know, like a password!

To have the master provide the detailed error that could potentially carry protected data set `pillar_safe_render_error` to `False`:

```
pillar_safe_render_error: False
```

Copyright 2017 SUSE

2.13 External Pillars

Salt provides a mechanism for generating pillar data by calling external pillar interfaces.

Just as with traditional pillars, external pillars must be refreshed in order for minions to see any fresh data.

SUSE Manager presents pillar data to minions using an external pillar module.

Copyright 2017 SUSE

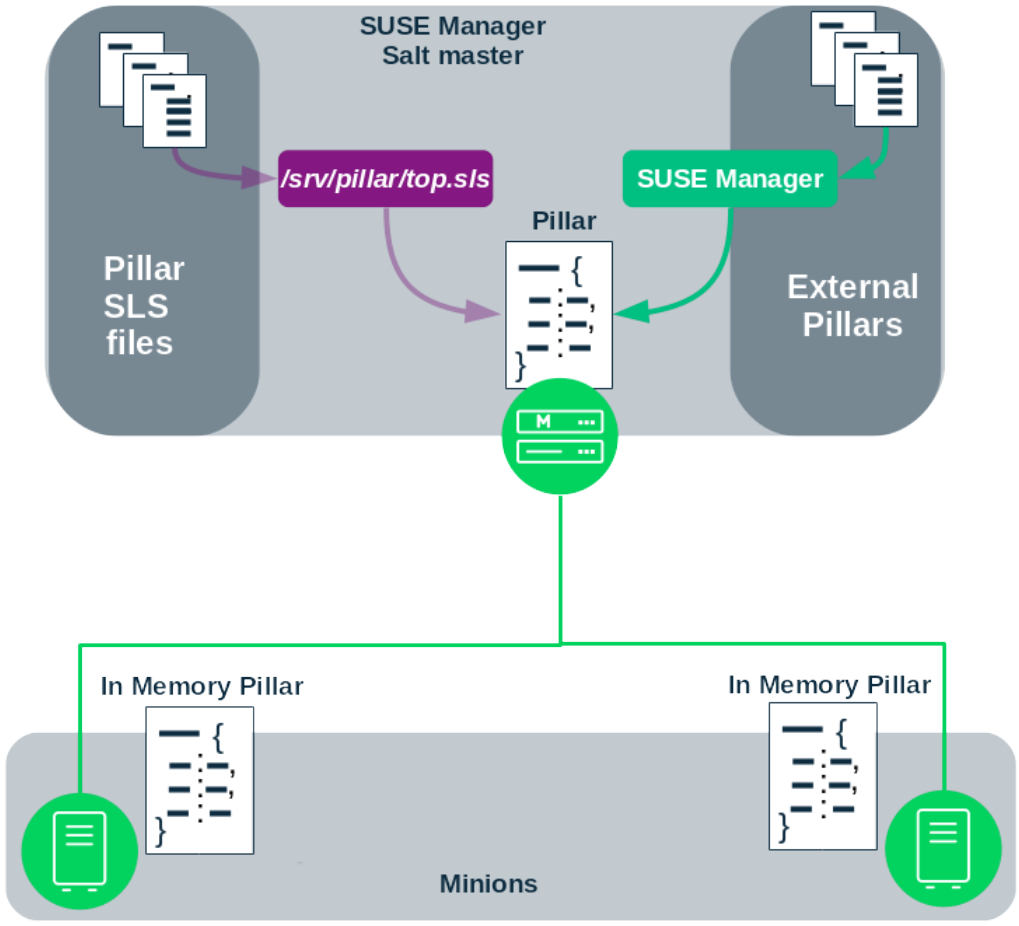

Fig. 2.2: SUSE Manager External Pillar

Copyright 2017 SUSE

2.14 Default SUSE Manager Provided Pillar Data

The

`/usr/share/susemanager/modules/pillar/suma_minion.py`

SUSE Manager external pillar module generates pillar data for managed systems for:

- beacons
- channels
- gpgkeys
- formulas
- system groups
- organization

Copyright 2017 SUSE

A sample of this pillar structure looks like:

```
server2.example.com:
  addon_group_types:
  - salt_entitled
  beacons:
    pkgset:
      cookie: /var/cache/salt/minion/rpmdb.cookie
      interval: 5
  channels:
    sle-manager-tools12-pool-x86_64-sp3:
    ...
  group_ids:
  - 8
  mgr_server: suma.example.com
  org_id: 1
```

Copyright 2017 SUSE

2.15 Summary

In this section we discussed:

- Storing Static Data in the Pillar
- Declaring Master Pillar Data
- Pillar Dictionary Merging
- Including Other Pillars
- In-Memory Pillar Data vs. On-Demand Pillar Data
- Viewing Pillar Data
- Viewing Pillar Errors

Copyright 2017 SUSE

3 Creating SUMA Forms and Formulas

Copyright 2017 SUSE

3.1 Objectives

In this section we will discuss:

- Creating SUSE Manager Forms for states
- Assigning formulas

Copyright 2017 SUSE

3.2 Managing States in SUSE Manager

1. SUSE Manager 3.1.4 now manages states in the web UI under **>Configuration>Configuration Channels**.
2. Here you can select **Create State Channel**.
3. In version 3.0-3.1 states were managed in a state catalog.

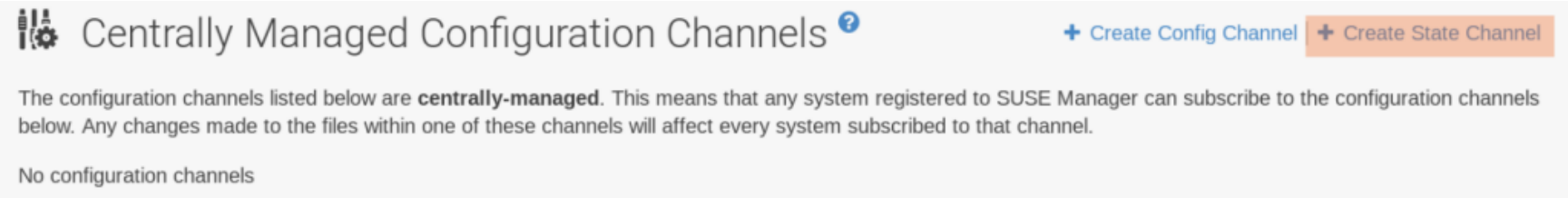

Fig. 3.1: Salt Configuration Channel

Copyright 2017 SUSE

3.2.1 Creating State Configuration Channel

- Creates a state `init.sls` file in a folder with the state name.

  For example, if the state was named `corp-ssh` then the contents would be placed as:

  `/srv/susemanager/salt/manager_org_1/corp-ssh/init.sls`

- This state can be referenced at the command-line as:

```
salt \* state.sls manager_org_1.corp-ssh
```

Copyright 2017 SUSE

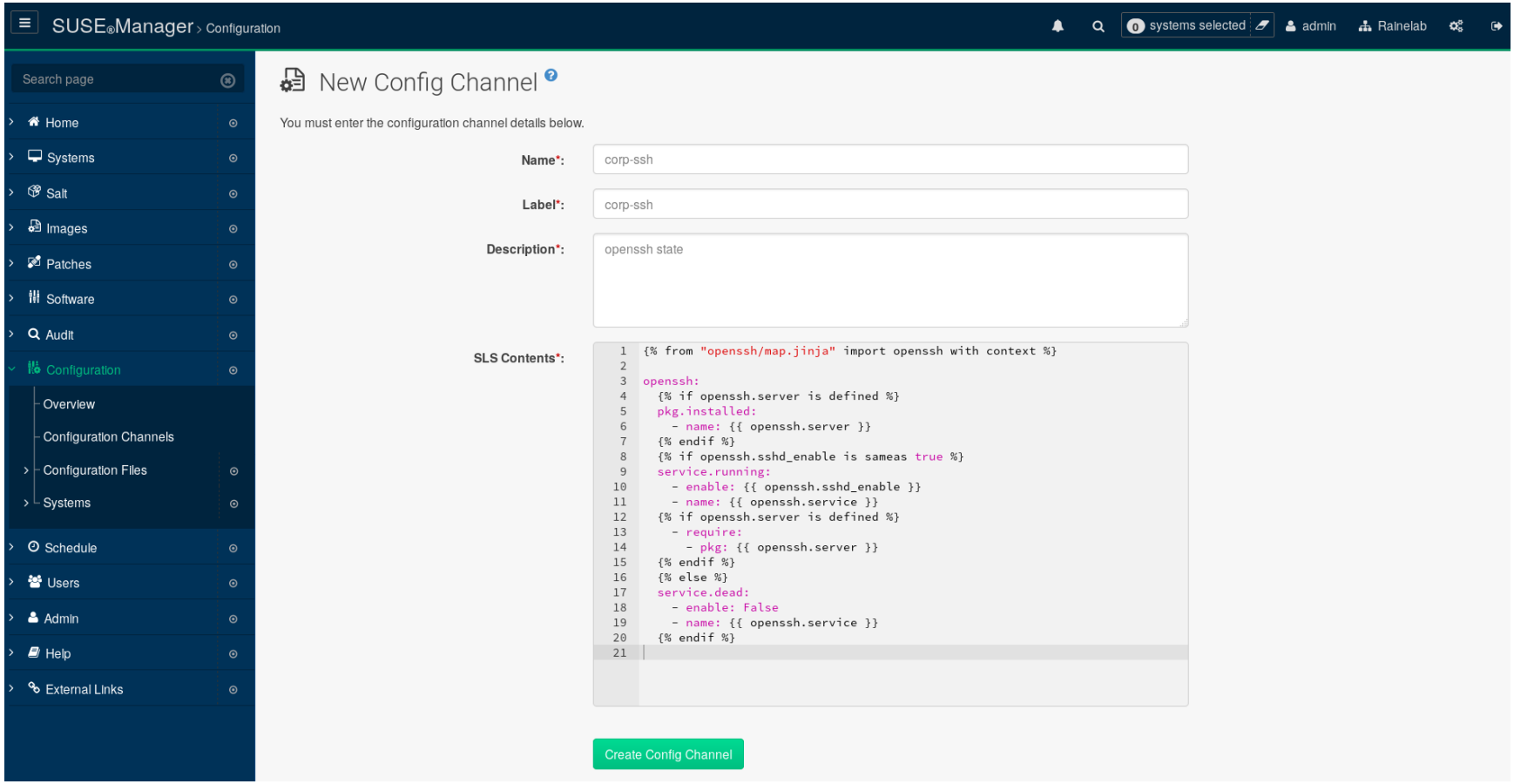

Fig. 3.2: State Creation in SUSE Manager

Copyright 2017 SUSE

3.2.2 Adding Additional Files to State Channel

Additional files may be added to the state channel.

- Configuration files and state files may be added.
- For example, if you added an `/etc/ntp.conf`
- When this file is uploaded it will be stored in

  `/srv/susemanager/salt/manager_org_1/ntp/etc/ntp.conf`

Copyright 2017 SUSE

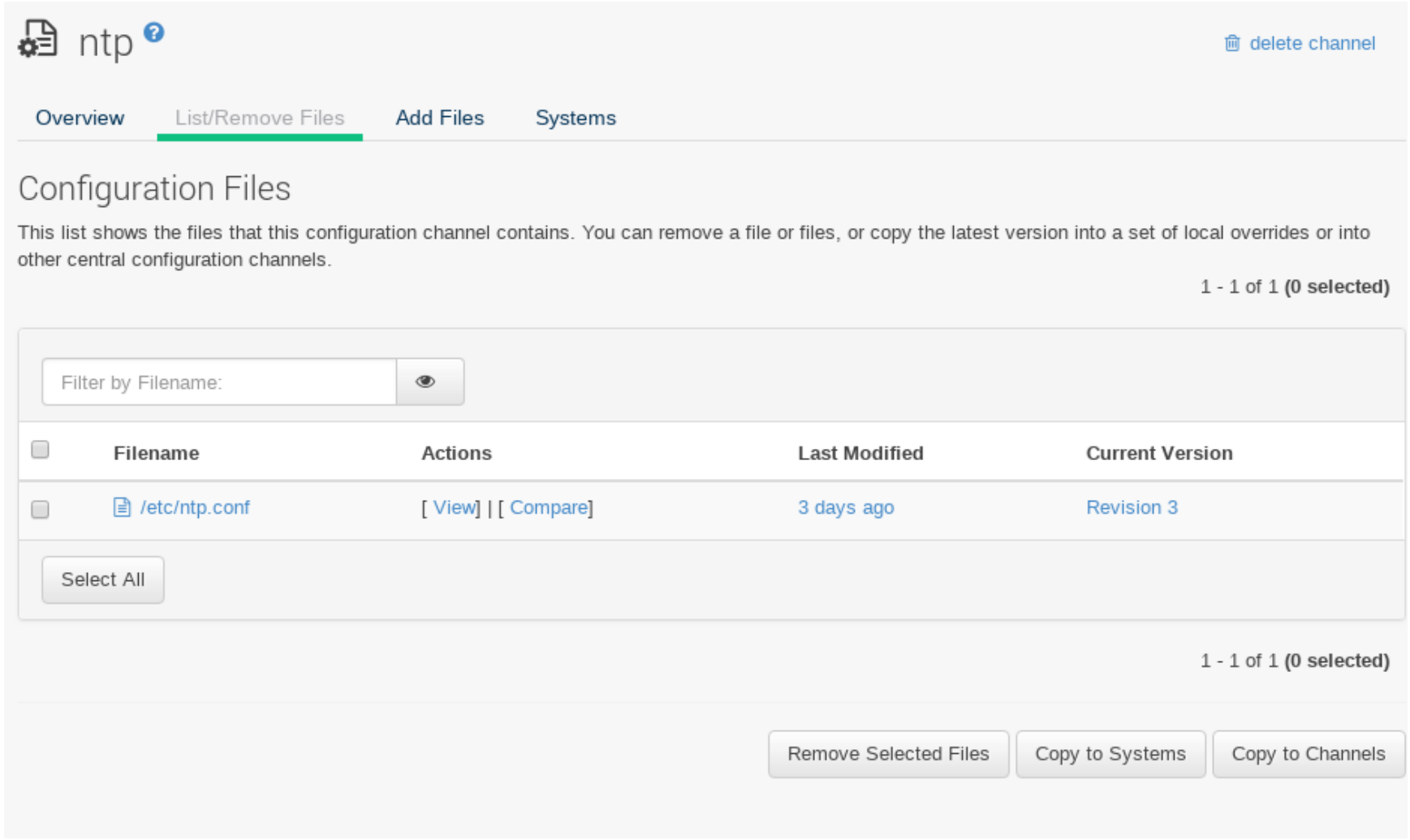

Fig. 3.3: Adding additional files to a state channel

Copyright 2017 SUSE

3.3 Salt Formulas

Formulas are collections of Salt States that have been pre-written by other Salt users and contain generic parameter fields.

Formulas allow for reliable reproduction of a specific configuration again and again.

Formulas can be installed from RPM packages, external git repository, or created locally on the disk.

Copyright 2017 SUSE

3.4 SUSE Manager Formula

A SUSE Manager formula differs form a regular Salt formula because it includes a form in the Web UI in addition to the set of states.

Type	Path
File-based	
states	/srv/salt
forms/metadata	/srv/formula_metadata
RPM-based	
states	/usr/share/susemanager/formulas/states
forms/metadata	/usr/share/susemanager/formulas/metadata

Fig. 3.4: Formula paths for SUSE Manager

Copyright 2017 SUSE

3.5 Developing a Formula

The following example shows how to develop a SUSE Manager Formula.

1. Develop the SUSE Manager form.

   Builds pillar data.

2. Develop the Salt State.

   Have the state use the pillar data.

Copyright 2017 SUSE

/srv/formula_metadata/ntp/form.yml

```
ntp:
  $name: NTP Client Settings
  $type: group
  server:
    $type: select
    $values: ['us.pool.ntp.org',
              'time.nist.gov',
              'europe.pool.ntp.org'
    ]
    $default : us.pool.ntp.org
  options:
    $type: text
    $default : iburst
```

Copyright 2017 SUSE

The example defined:

- creates pillar data as the form is defined and based on the form definition.
- a list of servers is possible for the `ntp:server` pillar.
- a text field is defined for the `ntp:options` pillar.

The form above will create a pillar similar to:

```
ntp:
  server: us.pool.ntp.org
  options: iburst
```

Copyright 2017 SUSE

3.6 Create the State

The following example state will consist of:

- the `init.sls` as the main state in `ntp`.
- create a `ntp.conf` template

Copyright 2017 SUSE

/srv/ntp/init.sls

```
# vim:syntax=yaml:ft=yaml
# -----------------------------------------------
# SUSE Professional Consulting Services
# description: ntp formula
# author: Raine Curtis (rcurtis@suse.com)
# -----------------------------------------------

{% ntp_server = salt['pillar.get']('ntp:server', 'us.pool.ntp.
↪org') %}
{% ntp_options = salt['pillar.get']('ntp:options', 'iburst') %}

install_ntp_pkg:
  pkg.installed:
   - name: ntp

push_ntp_conf:
  file.managed:
   - name: /etc/ntp.conf
```

(continues on next page)

Copyright 2017 SUSE

(continued from previous page)

```
    - user: root
    - group: ntp
    - mode: 640
    - require:
       - pkg: install_ntp_pkg
    - contents:  |
           # ntp configuration - managed by salt
           server {{ntp_server}} {{ntp_options}}

start_ntp_srv:
  service.running:
    - name: ntpd
    - enable: True
    - watch:
       - file: push_ntp_conf
```

Copyright 2017 SUSE

The above example works, but could be formalized by:

- create a `metadata.yml` file.
- create a `pillar.example` to show what pillar should create

Copyright 2017 SUSE

3.7 Using the Formula

A SUSE Manager formula will show in the **Salt > Formula Catalog** in the web UI.

It can be assigned to any place a state can be assigned such as:

- organization
- system group
- individual system

Copyright 2017 SUSE

3.8 Assigning the Formula

For example, to assign the formula to a system group select **Systems > System Groups** and then select the group you want.

Select the group, and then select the **Formulas** tab.

Check the box next to the formula you want to assign to the group.

A new sub-tab will be be displayed with the name of the formula.

Copyright 2017 SUSE

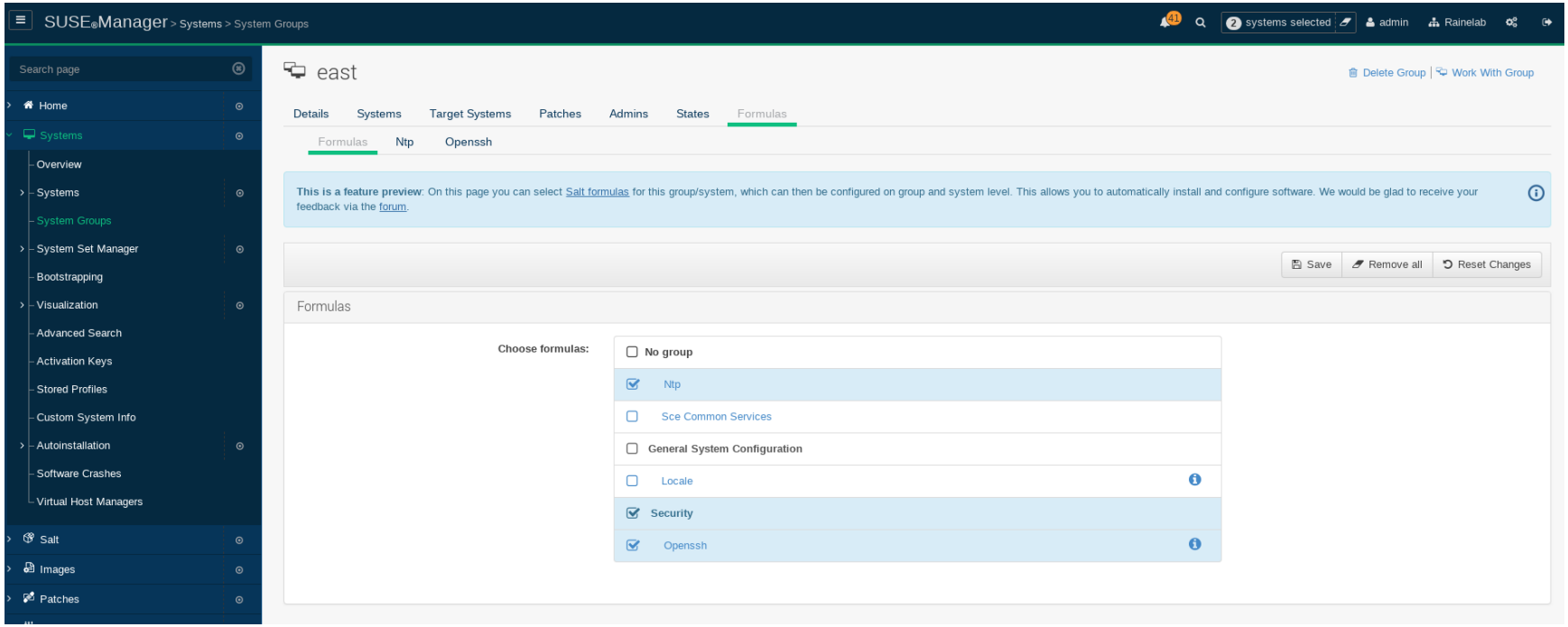

Fig. 3.5: Formula Assignment

Copyright 2017 SUSE

3.9 Setting Pillar Values

Pillar values may be assigned using the form.

Select the *sub-tab formula* page.

Copyright 2017 SUSE

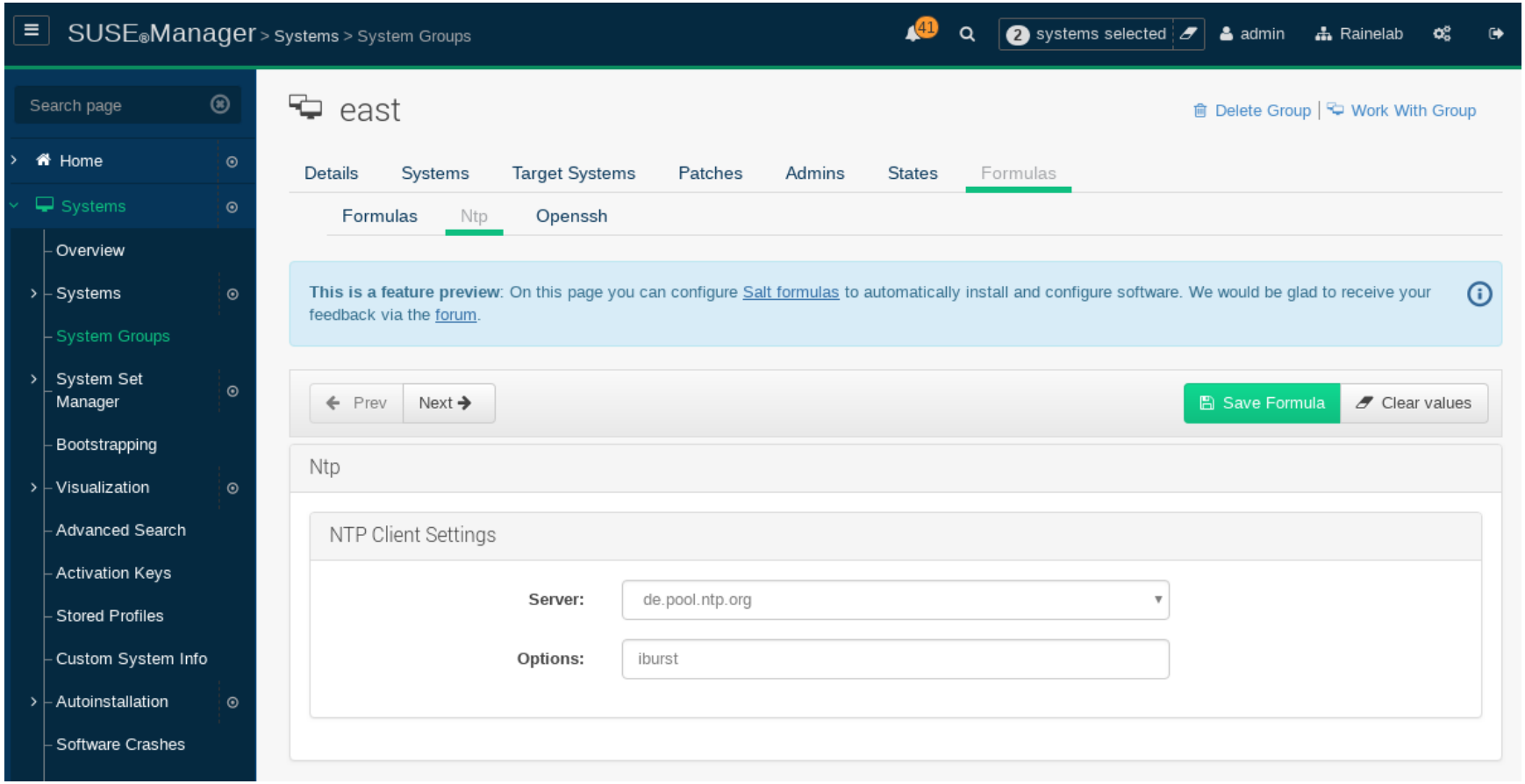

Fig. 3.6: Example Formula Form Page

Copyright 2017 SUSE

3.10 Summary

In this section we discussed:

- Creating SUSE Manager Forms for states
- Assigning formulas

Copyright 2017 SUSE

4 Demo

Demo of using SUSE Manager forms and formulas

Copyright 2017 SUSE

5 Thanks

Copyright 2017 SUSE