## Table of Contents

**Specialized Guides Overview**

1. Salt Guide Overview ......................................................... 2
   1.1. Terminology .................................................................... 2
   1.2. Salt Command ............................................................... 4
      1.2.1. Salt Targets ........................................................... 4
      1.2.2. Salt Execution Modules .......................................... 5
      1.2.3. Salt Function Arguments ....................................... 5
   1.3. Often Used Salt Commands ............................................ 6
   1.4. Salt States and Pillars .................................................. 7
      1.4.1. Group States .......................................................... 8
      1.4.2. Salt Pillars ............................................................. 8
      1.4.3. Download Endpoint ................................................. 9
   1.5. GPG Encrypted Pillars .................................................. 10
      1.5.1. Generate GPG keyring for Salt Master ....................... 10
      1.5.2. Use GPG for encrypting Pillar secrets ....................... 11
      1.5.3. Export the GPG key ................................................. 12
   1.6. Custom Salt States ........................................................ 12
      1.6.1. Create a New Custom Salt Channel ......................... 12
      1.6.2. Example Custom State Files ................................. 14
      1.6.3. Custom State to Trust a GPG Key ......................... 15
      1.6.4. Apply a custom state at highstate ....................... 16
   1.7. Salt File Locations and Structure .................................... 16
   1.8. The gitfs Fileserver Backend ......................................... 18
   1.9. Install Using Yomi ...................................................... 19
      1.9.1. Install the Yomi Formula ....................................... 20
      1.9.2. Install the PXE Image ........................................... 21
      1.9.3. Register Yomi in Cobbler ..................................... 21
      1.9.4. Example Salt Pillar Preparation ............................ 22
      1.9.5. Monitor the Installation ....................................... 25
   1.10. Configuration Modules ................................................ 25
      1.10.1. Install Configuration Modules ............................ 26
   1.11. Salt Formulas ............................................................ 26
      1.11.1. Formulas Provided by Uyuni ................................ 26
      1.11.2. Bind Formula ....................................................... 27
      1.11.3. Branch Network Formula .................................... 29
      1.11.4. DHCPd Formula ................................................... 31
      1.11.5. Image Synchronization Formula .......................... 33
      1.11.6. Liberate Formula ............................................... 33
      1.11.7. Monitoring Formula ............................................. 39
      1.11.8. PXE Formula ....................................................... 43
      1.11.9. Saltboot Formula ................................................. 44
      1.11.10. TFTPd Formula ................................................ 48
      1.11.11. Virtualization Guest Formula ............................ 48
      1.11.12. Virtualization Host Formula ............................... 48
      1.11.13. VsFTPd Formula ................................................ 49
      1.11.14. Yomi Formula .................................................... 49
11.1.5. Custom Salt Formulas ................................................. 53
11.2. Salt SSH ................................................................. 67
  11.2.1. SSH Connection Methods ........................................ 67
  11.2.2. Salt SSH Integration ............................................... 68
  11.2.3. Authentication ...................................................... 68
  11.2.4. User Account ........................................................ 68
  11.2.5. HTTP Redirection ................................................. 69
  11.2.6. Call Sequence ...................................................... 69
  11.2.7. Bootstrap Sequence ................................................ 70
  11.2.8. Proxy Support ........................................................ 72
  11.2.9. Users and SSH Key Management .................................. 75
  11.2.10. Repository Access with a Proxy ................................. 76
  11.2.11. Proxy Setup ........................................................ 77
  11.2.12. Rotate SSH keys ................................................... 78
11.3. Salt Rate Limiting ...................................................... 79
  11.3.1. Batching ............................................................ 79
  11.3.2. Disabling the Salt Mine .......................................... 80
11.4. Scaling Minions (Large Scale Deployments) .......................... 80
2. Large Deployments Guide Overview ...................................... 81
  2.1. Hardware Requirements ............................................... 81
  2.2. Using a Single Server to Manage Large Scale Deployments .......... 82
    2.2.1. Operation Recommendations .................................... 82
  2.3. Multiple Servers with Hub to Manage Large Scale Deployments ... 85
    2.3.1. Hub Requirements ................................................ 85
    2.3.2. Hub Installation .................................................. 86
    2.3.3. Using the Hub API ............................................... 87
    2.3.4. Hub XMLRPC API Namespaces .................................. 87
    2.3.5. Hub XMLRPC API Authentication Modes ......................... 88
    2.3.6. Hub Reporting .................................................... 92
  2.4. Managing Large Scale Deployments in a Retail Environment ........ 95
  2.5. Tuning Large Scale Deployments .................................... 95
    2.5.1. The Tuning Process .............................................. 96
    2.5.2. Environmental Variables ....................................... 97
    2.5.3. Parameters ........................................................ 98
  2.6. Monitoring Large Scale Deployments ................................ 115
3. Quick Start: SAP Overview ................................................. 116
  3.1. Prepare Server ......................................................... 116
  3.2. Preparing Clients ..................................................... 117
    3.2.1. Register Clients to the SUSE Customer Center ............... 117
    3.2.2. Configure the Clients for Clustering ......................... 117
    3.2.3. Create a Shared Storage Device ................................ 117
    3.2.4. Download the SAP Installation Software ....................... 118
    3.2.5. Configure Clients to Use Latest module.run ................ 118
    3.2.6. Install Additional Disks for HANA ............................ 118
    3.2.7. Register Clients to the Server ................................ 118
  3.3. Configure Clients ..................................................... 119
    3.3.1. Enable and Configure the HANA Formula ....................... 119
    3.3.2. Enable and Configure the Cluster Formula .................... 120
4. GNU Free Documentation License ......................................... 123
Specialized Guides Overview

Updated: 2024-04-29

This book contains a list of special topics and functionalities of Uyuni.

It is designed to introduce basic, routine or some advanced tasks, by explaining what you are achieving in each step, and the various options available to you along the way.

You can read specialized guides for:

- Specialized-guides › Salt
- Specialized-guides › Large-deployments
- Specialized-guides › Qs-sap
Chapter 1. Salt Guide Overview

Updated: 2024-04-29

Salt is a remote execution engine, configuration management and orchestration system used by Uyuni to manage clients.

In Uyuni, the Salt master runs on the Uyuni Server, allowing you to register and manage Salt clients.

This book is designed to be a primer for using Salt with Uyuni.

For more information about Salt, see the Salt documentation at https://docs.saltstack.com/en/latest/contents.html.

The current version of Salt in Uyuni is '3006.0'.

1.1. Terminology

Beacon

Beacons allow you to use the Salt event system to monitor non-Salt processes. Clients can use beacons to connect to various system processes for constant monitoring. When a monitored activity occurs, an event is sent on the Salt event bus that can then trigger a reactor.

To use beacons on SUSE Linux Enterprise Server Salt clients, install the python-pyinotify package. For Red Hat Enterprise Linux systems, install the python-inotify package.

For more information on beacons, see https://docs.saltstack.com/en/latest/topics/beacons/

Broker

The Salt broker allows clients to pass commands to each other. The broker acts like a switch, therefore peer communication will only work for clients on the same network, or connected to the same proxy.

For more information on Salt and peer communication, see https://docs.saltstack.com/en/latest/ref/peer.html.

Environment

Uyuni implements Salt with a single environment. Multiple Salt environments are not supported.

Formulas

Formulas are collections of Salt States that contain generic parameter fields. Formulas are used within Uyuni to assist with configuring Salt clients. Some formulas have extensive configuration options, and use forms to help organize them in the Uyuni Web UI.

For more information about formulas, see Specialized-guides › Salt.
**Grains**

Grains provide information about the hardware of a client. This includes the operating system, IP addresses, network interfaces, and memory. When you run a Salt command any modules and functions are run locally from the system being called.

For more information on grains, see https://docs.saltstack.com/en/latest/topics/grains/.

**Highstate**

This term is used when you apply all outstanding states to all targeted clients at the same time. The highstate must be applied when doing changes to systems, including enabling and disabling formulas.

**Key Fingerprints**

Key fingerprints are exchanged between the Uyuni Server and Salt clients to verify the identity of the server and the client. This prevents Salt clients from connecting to the wrong server. You can see the fingerprints of your Salt clients by navigating to Salt › Keys.

**Master**

The Salt master issues commands to its attached clients. In Uyuni, the Salt master must be the Uyuni Server.

**Minions**

Salt clients that are connected to and controlled by the Salt master on the Uyuni Server. In Uyuni, these are sometimes referred to as Salt clients. This is a difference in terminology only.

**Modules**

Functions within Salt are stored in modules. Salt modules are stored on clients and the Uyuni Server within the /usr/lib/python*/site-packages/salt/ directory. There are many types of Salt modules, including state and execution modules. You can write your own Salt modules using Python.

For a complete list of available Salt modules, see https://docs.saltstack.com/en/latest/ref/index.html.

**Pillars**

Pillars are created on the Uyuni Server. They contain information about a client or group of clients. Pillars allow you to send confidential information to a targeted client or group of clients. Pillars are useful for sensitive data, configuration of clients, variables, and any arbitrary data.

For more information on pillars, see https://docs.saltstack.com/en/latest/topics/tutorials/pillar.html.

**States**

States are configuration templates. They allow you to describe what each of your systems should look like, including the applications and services that are installed and running. States are applied to the target client. This automates the process of bringing a large number of systems into a known state, and then maintaining them.
Do not update the salt package using states. Update all other system packages using states. You can then update the salt package from the Uyuni Web UI as a separate step.

For more information on states, see https://docs.saltstack.com/en/latest/topics/tutorials/starting_states.html.

For more Salt terminology, see https://docs.saltstack.com/en/latest/glossary.html.

1.2. Salt Command

Salt commands have three main components: target, function, and arguments. The calls are constructed in this format:

\[
\text{salt 'target' <function> [arguments]}
\]

The target defines the client, or group of clients, on which to run the function.

The function is the particular task to be run.

Arguments provide any extra data required by the function.

1.2.1. Salt Targets

Salt command targets allow you to specify a client or group of clients. There are several different targets you can use.

**General Targeting**

List available grains on all clients:

\[
\text{salt '*' grains.ls}
\]

Target a specific client:

\[
\text{salt 'web1.example.com' test.ping}
\]

**Glob Targeting**

Target all clients using a particular domain:

\[
\text{salt '*example.com' test.ping}
\]

Target all clients using a particular label:
List Targeting

Specify a flat list of clients, using their IDs:

```bash
salt -L 'client_ID1, client_ID2, client_ID3' test.ping
```

Regular Expression Targeting

You can also define targets with PCRE-compliant regular expressions:

```bash
salt -E '(?!web)' test.ping
```

IP Address Targeting

List available client IP addresses:

```bash
salt '*' network.ip_addrs
```

Target a specific client IP address:

```bash
salt -S '172.31.60.74' test.ping
```

Target all clients on a subnet:

```bash
salt -S 172.31.0.0/16 test.ping
```

For more on targeting, see [https://docs.saltstack.com/en/latest/topics/targeting/](https://docs.saltstack.com/en/latest/topics/targeting/).

1.2.2. Salt Execution Modules

When you have specified a target, provide the module and function to execute on the target.

Find which modules can be executed on the target:

```bash
salt '*' sys.doc
```

For a full list of callable modules, see [https://docs.saltstack.com/en/latest/ref/modules/all/index.html](https://docs.saltstack.com/en/latest/ref/modules/all/index.html).

1.2.3. Salt Function Arguments

Functions accept arguments for any extra data.
For example, the `pkg.install` function requires an argument specifying which package to install:

```bash
salt '*' pkg.install yast2
```

You can provide more than one argument to a function, with spaces between them. For example:

```bash
salt '*' cmd.run 'echo "Hello: $FIRST_NAME"' env='{FIRST_NAME: "John"}"
```

### 1.3. Often Used Salt Commands

This section contains the most commonly used Salt commands. For a complete list of available Salt commands, see [https://docs.saltstack.com/en/latest/ref/cli/index.html](https://docs.saltstack.com/en/latest/ref/cli/index.html).

**salt-run**

Display all clients that are running:

```bash
salt-run manage.up
```

Display all clients that are not running:

```bash
salt-run manage.down
```

Display the current status of all Salt clients:

```bash
salt-run manage.status
```

Check the version of Salt running on the Uyuni Server and active clients:

```bash
salt-run manage.versions
```

**salt-cp**

Copy a file to a client or set of clients:

```bash
salt-cp '*' foo.conf /root
```

**salt-key -l**

List public keys:

```bash
salt-key -l all
```
salt-key -a my-minion
Accept pending key for a minion:

```
salt-key -a my-minion
```

salt-key -A
Accept all pending keys:

```
salt-key -A
```

salt grains
List all available grains:

```
salt '*' grains.ls
```

List collected grain system data:

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

1.4. Salt States and Pillars

States are configuration templates. They allow you to describe what each of your systems should look like, including the applications and services that are installed and running. Salt state files are referred to as SLS (Salt State) files.

States are applied to the target systems by matching relevant state data to clients. The state data comes from Uyuni in the form of package and custom states.

You can target clients at three specific levels of hierarchy and priority: individual clients, system groups, and organization. Individual clients have priority over groups, and groups have priority over the organization.

For example:

- The Organization requires that version 1 is installed. All clients are part of the same Organization.
- Group A requires that version 2 is installed. Client1, Client2, and Client3 are part of Group A.
- Group B requires any version installed. Client4 is part of Group B.

Leading to these possible scenarios:

- Client1 wants package removed, package is removed (Client Level)
- Client2 wants version 2, gets version 2 (Client Level)
• Client3 wants any version, gets version 2 (Group Level)
• Client4 wants any version, gets version 1 (Organization Level)

For more information on Salt states, see https://docs.saltproject.io/en/latest/topics/states/.

You can create custom Salt states with Uyuni. For more information, see Specialized-guides › Salt.

1.4.1. Group States

Pillar data can be used to perform bulk actions, like applying all assigned states to clients within the group. This section contains some examples of bulk actions that you can take using group states.

To perform these actions, you will need to determine the ID of the group that you want to manipulate. You can determine the Group ID by using the spacecmd command:

```
spacecmd group_details
```

These examples use an example Group ID of GID.

To apply all states assigned to the group:

```
salt -I 'group_ids:GID' state.apply custom.group_GID
```

To apply any state (whether or not it is assigned to the group):

```
salt -I 'group_ids:GID' state.apply `state`
```

To apply a custom state:

```
salt -I 'group_ids:2130' state.apply manager_org_1.'`customstate`'
```

Apply the highstate to all clients in the group:

```
salt -I 'group_ids:GID' state.apply
```

1.4.2. Salt Pillars

Uyuni exposes a small amount of internal data as pillars which can be used with custom states. Pillars are created on the Uyuni Server, and contain information about a client or group of clients. For custom information in pillars, see Client-configuration › Custom-info. Pillars are useful for sensitive data, configuration of clients, variables, and any arbitrary data.

Pillars are managed either automatically by Uyuni, or manually by the user.
If you change pillar data on the server (Salt master) the actual pillar data on the client (minion) is updated only after calling `saltutil.refresh_pillar` for the client.

Otherwise it could happen that `pillar.items` and `pillar.get` calls would produce different results with different values of pillars that were not refreshed.

To avoid hard-coding organization IDs within SUSE Linux Enterprise Server files, a pillar entry is added for each organization:

```
org-files-dir: relative_path_to_files
```

The specified file is available for all clients which belong to the organization.

This is an example of a pillar located at `/etc/motd`:

```
file.managed:
  - source: salt://{{ pillar['org-files-dir'] }}/motd
  - user: root
  - group: root
  - mode: 644
```

For more information on Salt pillars, see https://docs.saltproject.io/en/latest/topics/pillar/.

### 1.4.3. Download Endpoint

By default, Uyuni assumes that the download endpoint to use is the FQDN of the Uyuni Server or Proxy. However, there are some cases where you might like to use a different FQDN as the download endpoint. The most common example is if you need to use load balancing, caching proxies, or in environments with complicated networking requirements.

To change the package download endpoint, you can manually adjust three Salt pillars: * `pkg_download_point_protocol`, defaults to https. * `pkg_download_point_host`, defaults to the FQDN of the Uyuni Server (or Proxy, if in use). * `pkg_download_point_port`, defaults to 443.

If you do not adjust these pillars directly, Uyuni will fall back to the default values.

**Procedure: Changing the Package Download Endpoint Pillar**

1. Navigate to `/srv/pillar/` and create a file called `top.sls` with these contents:

```
base:
  '*':
    - pkg_download_points
```
This example directs Salt to look at the `pkg_download_points.sls` file to determine the base URL to use. You can adjust this file to target different clients or groups, depending on your environment.

2. Remain in `/srv/pillar/` and create a file called `pkg_download_points.sls` with the base URLs you want to use. For example:

   ```
   pkg_download_point_protocol: http
   pkg_download_point_host: example.com
   pkg_download_point_port: 444
   ```

3. OPTIONAL: If you want to use external pillars, for example Group IDs, open the master configuration file and set the `ext_pillar_first` parameter to `true`. You can then use Group IDs to set conditional values, for example:

   ```
   {% if pillar['group_ids'] is defined and 8 in pillar['group_ids'] %}
   pkg_download_point_protocol: http
   pkg_download_point_host: example.com
   pkg_download_point_port: 444
   {% else %}
   pkg_download_point_protocol: ftp
   pkg_download_point_host: example.com
   pkg_download_point_port: 445
   {%- endif %}
   ```

4. OPTIONAL: You can also use grains to set conditional values, for example:

   ```
   {% if grains['fqdn'] == 'client1.example.com' %}
   pkg_download_point: example1.com
   {% elif grains['fqdn'] == 'client2.example.com' %}
   pkg_download_point: example2.com
   {% else %}
   pkg_download_point: example.com
   {%- endif %}
   ```

### 1.5. GPG Encrypted Pillars

Salt has support to transparently decrypt GPG-encrypted Pillar data built-in. The decryption happens on the Salt Master.

#### 1.5.1. Generate GPG keyring for Salt Master

The GPG keyring can be specified in `/etc/salt/master` or in its own file under `/etc/salt/master.d/`, for example `/etc/salt/master.d/gpg-pillar.conf`.

Always create a separate keyring for the Salt Master.

**Procedure: Generating key pair**

1. On the Salt Master create GPG home directory and restrict its permissions:
mkdir /etc/salt/gpgkeys
chmod 700 /etc/salt/gpgkeys

2. Generate a key pair interactively.

   The password must be empty.

   gpg --gen-key --homedir /etc/salt/gpgkeys

3. Salt does not run with root permissions on SUSE Linux Enterprise and openSUSE distributions.

   chown -R salt:salt /etc/salt/gpgkeys

4. Configure Salt Master to use the new GPG home directory

   echo 'gpg_keydir: /etc/salt/gpgkeys' >/etc/salt/master.d/gpg-pillar.conf
   systemctl reload-or-restart salt-master

1.5.2. Use GPG for encrypting Pillar secrets

Salt GPG renderer decrypts GPG encrypted contents that are ASCI-armored. To use the GPG renderer in a Pillar YAML file, change

```yaml
#!yaml
```

to

```yaml|gpg
```

Encrypting pillar secrets can be done anywhere as long as the GPG and the public key generated in Procedure: Generating key pair are available.

In this example, "SUMA Salt Master" is the GPG key’s UID created earlier.

```bash
echo 't0ps3cr3t' | gpg --armor --batch --encrypt --recipient "SUMA Salt Master"
```

When the GPG encrypted contents are created and available as ASCII-armored output, this output can be used as a multi-line string in a pillar YAML file:
When the pillar is assigned to a system with `top.sls`, the GPG encrypted pillar data is available in a decrypted format.

```
When the pillar is assigned to a system with top.sls, the GPG encrypted pillar data is available in a decrypted format.

 suma-sles15sp1.tf.local:
----------
my-secret:
t0p s3cr3t!
```

1.5.3. Export the GPG key

To export the GPG key, use the command:

```
gpg --export 'SUMA Salt Master' --homedir /etc/salt/gpgkeys --output suma-salt-master.gpg```

Here 'SUMA Salt Master' is the name used during key generation.

The `suma-salt-master.gpg` public key can be freely shared.

1.6. Custom Salt States

You can create your own custom Salt states with Uyuni as centrally managed configuration channels. Custom states are stored as Salt state files on the Uyuni Server with a `.sls` extension.

1.6.1. Create a New Custom Salt Channel

You can use the Uyuni Web UI to create and edit custom Salt state files. You must create a state channel first, with an initial state named `init.sls`. The `init.sls` file is used to reference all other state files within the channel. The custom states that you create using the Web UI are stored on the Uyuni Server in
the /srv/susemanager/salt/<organization>/ directory.

After the channel is created with an init.sls file, you can write additional state files in the Web UI. Alternatively, you can upload existing state files to use within your state channel, or import them from other channels or clients.

**Procedure: Creating a Custom Salt Channel and Initial State**

1. In the Uyuni Web UI, navigate to **Configuration > Channels**.
2. Click **[Create State Channel]**.
3. In the **Name** field, type a name for your state.
4. In the **Label** field, type a label. Use alphanumeric characters, hyphens, and underscores. Do not use spaces.
5. In the **Description** field, type a short description of the configuration your state performs.
6. In the **SLS Contents** field, type the contents of your init.sls state. If you want to reference file templates in this configuration channel, ensure your file starts by specifying the source of the managed file, using this syntax:

   ```
   file.managed:
   - source: salt://<org_name>/<channel_name>/etc/<ID>/<filename>
   ```

   Example custom state files are given later in this section. Click **btn:[Update Channel]** to save your state.

**Procedure: Adding Additional Files to a Custom State Channel**

1. In the Uyuni Web UI, navigate to **Configuration > Channels**.
2. Click the name of the channel you want to add files to.
3. To create a new file, click **[Create configuration file]** and type the contents of the file.
4. To upload an existing file, click **[Upload Configuration Files]** and select the file to upload.
5. To copy an existing file, click **[Import a File from Another Channel or System]** and select the file to copy.

**Procedure: Editing a Custom Salt State**

1. In the Uyuni Web UI, navigate to **Configuration > Channels**.
2. Click **[View/Edit <filename>.sls File]**.
3. Make your changes to the file.
4. Click **[Update Configuration File]** to save your state.

You can also manage revisions, compare the state to others in your organization, and download the .sls file from this dialog.
Procedure: Assigning a Client to a Custom Salt State

1. In the Uyuni Web UI, navigate to **Configuration › Channels**.
2. Click the name of the state you want to assign a client to.
3. Navigate to the **Systems › Target Systems** tab.
4. Check the clients you want to assign.
5. Click **[Subscribe systems]**.

For more information about Salt state modules, see [https://docs.saltproject.io/en/latest/ref/states/all/index.html](https://docs.saltproject.io/en/latest/ref/states/all/index.html).

### 1.6.2. Example Custom State Files

This section contains some example custom state files. Use these as a basis for writing your own custom states.

#### Listing 1. Example: Manage a File

```yaml
my_config_change_id:
  file.managed:
    - name: /etc/my.conf
    - source: salt://example_org/example_channel/etc/my.conf
    - user: root
    - group: root
    - mode: 644
    - template: jinja
```

#### Listing 2. Example: Package Management

```yaml
my_pkg_id:
  pkg.installed:
    - refresh: True
    - pkgs:
      - glibc
      - kernel-default
      - hello: 1.0-42
```

#### Listing 3. Example: Remote Command

```yaml
ip_forward-on:
  cmd.run:
    - name: echo "1" > /proc/sys/net/ipv4/ip_forward
    - onlyif:
      - test `cat /proc/sys/net/ipv4/ip_forward` -eq 0
```

#### Listing 4. Example: Service Management

```yaml
time_service_id:
  service.running:
    - name: chronyd
    - enable: True
```
1.6.3. Custom State to Trust a GPG Key

By default, operating systems trust only their own GPG keys when they are installed, and do not trust keys provided by third party packages. The clients can be successfully bootstrapped without the GPG key being trusted. However, you cannot install new third party packages or update them until the keys are trusted.

Salt clients are set to trust SUSE tools channels GPG keys when they are bootstrapped. For all other clients and channels, you need to manually trust third party GPG keys.

If you are bootstrapping Salt clients from the Uyuni Web UI, you can use a custom Salt state to trust the GPG key.

Procedure: Trusting a GPG Key With a Custom Salt State

1. Locate the key that you need to trust. Ensure you have the correct key, and that you also have the fingerprint used to verify the key. This information is available from the vendor or, in some cases, from a key server.

2. Copy the key to a file location where the client can access it. We recommend saving it in the /srv/www/htdocs/pub/ directory, where all user public files must be saved.

3. In the Uyuni Web UI, navigate to Configuration > Channels.

4. Click [Create State Channel].

5. In the Name field, type a name for your state. For example, GPG Key Trusts.

6. In the Label field, type a label. For example, GPG_Key_Trusts.

7. In the Description field, type a short description of the configuration your state performs. For example, Trusts GPG Keys for CentOS.

8. In the SLS Contents field, create a state to retrieve the appropriate key from the Uyuni Server and trust it on the client. The exact contents of your state varies depending on your client operating system. For example:

```bash
rpm_trust_gpg_key:
   cmd.run:
     - name: rpm --import https://{{ salt['pillar.get']('mgr_server') }}/pub/<third-party-gpg>.key
     - unless: rpm -q gpg-pubkey-<key_id>

deb_trust_gpg_key:
   mgrcompat.module_run:
     - name: pkg.add_repo_key
     - path: https://{{ salt['pillar.get']('mgr_server') }}/pub/<third-party-gpg>.key
```
Alternatively, you can add GPG keys to a configuration channel, using a managed file to deploy them directly on the client. In this case, you would use a local path to the key, rather than a URL.

- Click btn:[Update Channel] to save your state.
- Navigate to menu:Configuration[Channels] and click the name of the state you want to assign a client to.
- Navigate to the menu:Systems[Target Systems] tab and check the clients you want to assign.
- Click btn:[Subscribe systems]. When the configuration file is next run on the client, the GPG key is trusted.

Alternatively, you can manage your GPG keys from your own repository hosted on an external file management system.

1.6.4. Apply a custom state at highstate

To apply a custom state at highstate create a mapping in /srv/salt/top.sls. This short example maps the test state to the system group 12:

```bash
# /srv/salt/top.sls
base:
  'group_ids:12':
    - match: pillar
    - test
```

1.7. Salt File Locations and Structure

There are several ways to set up the Salt file structure. This section describes how Salt is supported and set up as part of Uyuni Server. The main configuration file is /etc/salt/master.d/susemanager.conf.

Do not edit the /etc/salt/master.d/susemanager.conf configuration file. This file belongs to the spacewalk-setup package and is marked as %config. When SUSE updates the spacewalk-setup package, the susemanager.conf file is overwritten, and any customization is lost. Instead, add your own configuration file to the /etc/salt/master.d/ directory. This prevents the update process from deleting your settings from the main susemanager.conf configuration file.

Some settings from /etc/salt/master.d/susemanager.conf that can help with finding configuration options:
When you are working with `/etc/salt/master.d/susemanager.conf`, be aware that:

- Files listed are searched in the order they appear
- The first matching file found is called

The Uyuni Server reads Salt state data from five root directories:

**/usr/share/susemanager/salt**

This directory is shipped and updated with Uyuni and includes certificate setup and common state logic to be applied to packages and channels.

⚠️ Do not edit or add custom Salt data to this directory.

**/usr/share/salt-formulas/states**

**/usr/share/susemanager/formulas/states**

These directories are shipped and updated with Uyuni or additional extensions. They include states for Salt formulas.

⚠️ Do not edit or add custom Salt data to this directory.

**/srv/susemanager/salt**

This directory is generated by Uyuni, based on assigned channels and packages for clients, groups, and organizations. This directory will be overwritten and regenerated. It is the Salt equivalent of the Uyuni database.

⚠️ Do not edit or add custom Salt data to this directory.

Within this directory, each organization has a sub-directory.
Listing 5. Example: SLS File Directory Structure

```
manager_org_<org id>
  - files
    ... files needed by states (uploaded by users)...
    state.sls
    ... other SLS files (created by users)...
For example:
manager_org_TESTING
  - files
    - motd     # user created
    ... other files needed by states ...
    motd.sls    # user created
    ... other SLS files ...
```

/srv/salt

This directory is used for custom state data, modules, and related data. Uyuni does not operate or use this directory directly. The state data in this directory is used by the client highstate, and is merged with the total state result generated by Uyuni. Use this directory for custom Salt data.

The Uyuni Server reads Salt pillar data from two root directories:

/usr/share/susemanager/pillar

This directory is generated by Uyuni. It is shipped and updated together with Uyuni.

⚠️ Do not edit or add custom Salt data to this directory.

/srv/pillar

By default, Uyuni does not operate or use this directory directly. The custom pillar data in this directory is merged with the pillar result created by Uyuni. Use this directory for custom Salt pillar data.

ℹ️ You can use the gitfs fileserver backend to serve Salt data from git repositories. For more information, see Specialized-guides › Salt.

1.8. The gitfs Fileserver Backend

In Uyuni, pygit2 is the supported Python interface to git. When pygit2 is installed the gitfs fileserver backend is available and it is a supported feature.

Configuration options are set in the /etc/salt/master file, or in a separate configuration file in the /etc/salt/master.d/ directory. The basic settings are:

**fileserver_backend**

List of fileserver backends that the Salt master checks for files in the order they are defined. Options:

- **roots**: Files local on the Salt master (Uyuni Server). `roots` is required to keep the product running. You can only enable `gitfs` optionally. Additionally, SUSE strongly recommends to prefer `roots` (local files) over `gitfs`. The standard backend.
- **gitfs**: Files stored in one or more git repositories. The repositories are defined with `gitfs_remotes`.

  Example:

  ```
  filesserver_backend:
  - roots
  - git
  ```

**gitfs_remotes**

List of git repositories. `git://, https://, file://, or ssh://` URLs can be configured. For SSH remotes, a `scp`-like syntax is also supported; for example: `gitlab@gitlab.example.com:universe/setup.git`. Then you can also specify options for credentials, file locations, or branches such as `pubkey`, `privkey`, `root`, `base`.

Example:

```
gitfs_remotes:
- https://example.com/myformulas/formula.git
- gitlab@gitlab.example.com:universe/setup.git:
  - pubkey: /var/lib/salt/.ssh/id_rsa_gitlab.pub
  - privkey: /var/lib/salt/.ssh/id_rsa_gitlab
  - root: srv/salt
  - base: master
```

**ext_pillar**

List of external pillar interfaces. Salt can also serve pillar data from one or more git repositories. For syntax and options, also see the `gitfs_remotes` setting.

Example:

```
ext_pillar:
- git:
  - master gitlab@gitlab.example.com:universe/setup.git:
    - root: srv/pillar
    - pubkey: /var/lib/salt/.ssh/id_rsa_gitlab.pub
    - privkey: /var/lib/salt/.ssh/id_rsa_gitlab
```

For more information, see:


### 1.9. Install Using Yomi

Yomi (yet one more installer) is an installer for SUSE and openSUSE operating systems. Yomi is designed as a Salt state, and can be used for installing SUSE operating systems on new systems.
In Uyuni, Yomi can be used as part of provisioning new clients, as an alternative to AutoYaST.

Yomi consists of two components:

- The Yomi formula, which contains the Salt states and modules required to perform the installation.
- The operating system image, which includes the pre-configured `salt-minion` service.

Both components can be used independently of Uyuni, or integrated with it. This section describes how to use it with Uyuni.

- For more information about using Yomi independently, see [https://github.com/openSUSE/yomi](https://github.com/openSUSE/yomi).
- For build assets, see [https://build.opensuse.org/project/show/systemsmanagement:yomi](https://build.opensuse.org/project/show/systemsmanagement:yomi).

To use Yomi for installing a client operating system, follow this process:

- Install the `yomi-formula` package.
- Prepare the Salt pillar for the new installation.
- Boot the new client using the PXE boot image for Yomi.

To use Yomi with Uyuni, ensure you have enough available memory. To boot from USB or DVD image, you need at least 512 MB. To boot from a PXE server, you need at least 2 GB.

### 1.9.1. Install the Yomi Formula

Before you begin, you need to install the Yomi formula, which is available as a package in Uyuni.

The `yomi-formula` package contains the Salt states and modules that describe the Yomi state, and the formulas with forms to create the pillar. It also contains documentation about the different sections of the pillar, and some examples about how to parameterize installations based on openSUSE, MicroOS, or SLE.

The formula package performs these actions:

- Adds a new configuration file called `yomi-formula.conf` in the `/etc/salt/master.d` directory. This configuration file defines the Python module and Salt states required by Yomi.
- Installs the Yomi Salt states in the `/usr/share/salt-formulas/states/` directory.
- Provides some example configuration files in the `/usr/share/yomi/` directory.
- Installs the required forms and sub-forms in the `/usr/share/salt-formulas/metadata/` directory.
- Provides some pillar examples in the `/usr/share/yomi/pillar/` directory.

**Procedure: Installing the Yomi Formula**

1. On the Uyuni Server, at the command prompt, as root, install the `yomi-formula` package:
zypper in yomi-formula

2. Restart services:

systemctl restart salt-master.service

For more information about the Yomi formula, see Specialized-guides > Salt.

1.9.2. Install the PXE Image

To provision a new client, you need an operating system image to boot from. You can use any image that contains a salt-minion service enabled, together with a minimal set of tools that are required during the installation, for example parted or btrfs tools.

Yomi provides an already prepared image, based on openSUSE Tumbleweed, openSUSE Leap (for Uyuni), or SLE (for SUSE Manager). For Uyuni, the image is packaged as an RPM. This is done in a similar way to how pxe-default-image is distributed.

The package installs a standard PXE OEM image generated by Kiwi, the initial kernel and initrd in the /srv/pxe-yomi-image/ directory, and the second stage kernel, initrd and image in the /srv/pxe-yomi-image/image directory.

Procedure: Installing the PXE Image

1. On the Uyuni Server, at the command prompt, as root, install the pxe-yomi-image service:

zypper in pxe-yomi-image-opensuse15

When you have the package installed, you can register Yomi in Cobbler.

1.9.3. Register Yomi in Cobbler

Uyuni uses Cobbler to manage the PXE boot service, so you will need to register the image in Cobbler.

Procedure: Registering the Yomi Image in Cobbler

1. On the Uyuni Server, at the command prompt, as root, create a directory for the Yomi image:

mkdir /srv/tftpboot/pxe-yomi-image

2. Define a distribution in Cobbler, including the path to install the second stage kernel and initrd, the location of the full image, and any further kernel options. Adjust this command to include the correct version of the product, and the TFTP server address:
By default, the salt-minion service in pxe-yomi-image is configured to find the Salt master under the salt address. If the DNS server is not able to resolve this address, you need to adjust the kernel-options parameter from the Cobbler command that register the distribution, and add a new kernel command line of `ym.master=master_address`. This will override the default configuration for the salt-minion.

**Procedure: Registering the Yomi Profile in Cobbler**

1. On the Uyuni Server, at the command prompt, as root, define a profile in Cobbler based on the image.

```bash
cobbler profile add --name pxe-yomi-profile --distro=pxe-yomi-image
```

2. **OPTIONAL**: Create a system in Cobbler. If you know the MAC address for the new client to be provisioned, you can have it boot directly from the Yomi image.

```bash
cobbler system add --name=yomi --mac=00:11:22:33:44:55 --profile=pxe-yomi-profile
```

3. When the new node has been provisioned, remove the temporary Cobbler system:

```bash
cobbler system remove --name=yomi
```

**1.9.4. Example Salt Pillar Preparation**

The parameters of the new installation are defined with a Salt pillar. The pillar includes parameters that the Yomi state requires during the installation, including the partitions, file systems, repositories, packages installed, and services enabled.
The pillar is defined using the formulas with forms. In this example, we prepare the pillar for a minimal openSUSE Tumbleweed installation. You can find examples for MicroOS or SLES in the example directory `/usr/share/yomi/pillar/.

To begin, boot the client that you want to provision using the Yomi PXE boot image, using the Cobbler procedures described earlier in this section.

When the salt-minion service is running on the new client, accept the key by navigating to Salt › Keys. When the key is accepted, you can view and manage the client by navigating to Systems › Overview. Navigate to the Formulas tab, and add all the Yomi Installer formulas to the client. When you have added all the formulas, complete the forms and sub-forms. This section outlines each form and provides example settings for a minimal installation. For a detailed explanation of every option, see Specialized-guides › Salt.

Yomi
The Yomi form contains some general configuration options. For example, the keyboard language and layout, the locale information, and the option to perform a full reset of the system after provisioning.

For this example, set the Reboot parameter to yes.

Yomi Storage
This sub-form provides information about the devices, partitioning, file system (including the Btrfs subvolumes, for example), and LVM and RAID configuration.

For this example, we assume that the new client has a single device named `/dev/sda`, and that it belongs to a non-UEFI system. In this case, we have only three partitions: one for the boot loader, one for swap and one for the system. We also expect to have an ext4 file system for the root directory.

Device 1:

- Device: `/dev/sda`
- Label: GPT
- Initial Gap: 1 MB

Create three partitions:

- Partition 1:
  - Partition Number: 1
  - Partition Size: 1 MB
  - Partition Type: boot

- Partition 2:
  - Partition Number: 2
  - Partition Size: 1024 MB
Partition Type: swap

- Partition 3:
  - Partition Number: 3
  - Partition Size: rest
  - Partition Type: linux

Create two file systems:

- Filesystem 1:
  - Partition: /dev/sda2
  - Filesystem: swap

- Filesystem 2:
  - Partition: /dev/sda3
  - Filesystem: ext4
  - Mountpoint: /

**Yomi Bootloader**

This sub-form provides details required for GRUB.

Set these parameters:

- Device: /dev/sda
- Theme: selected

The *Kernel* parameter can be used for the GRUB *append* section.

**Yomi Software**

This form provides the different repositories and packages to install. You can also register the product in this form, using SUSEConnect, and install the different modules after registering.

For this example we are going to install a very minimal openSUSE Tumbleweed distribution, using publicly available repositories. For production deployments, you will need to provide a local repository.


Add these packages: * pattern:enhanced_base * glibc-locale * kernel-default

You can also add patterns and products, together with packages, by using the correct prefix.

**Yomi Services**

By default Yomi is installed with the *salt-minion* service, but you must enable it.
Add a new enabled service:

- Service 1:
  - Service: salt-minion

Yomi Users

This form sets out the system users. In this example, we have a single root user. To provide a password, you must use the hashed version of the password, not the plain text. This behavior is set to be changed in future versions of Yomi.

- User 1:
  - Username: root
  - Password Hash: $1$wYJUgpM5$RXMMeASDc035eXNbYWF10

1.9.5. Monitor the Installation

You can monitor the installation as it progresses, using the `monitor` tool from Yomi. You can continue monitoring as the highstate is applied to the new client. To use the tool, you will need to have enabled Events in the Yomi formula, and have the `salt-api` service activated.

For more information about the salt-api service, and how to use the monitor tool, see https://github.com/openSUSE/yomi.

1.10. Configuration Modules

Salt uses execution and state modules to define, apply, and orchestrate configuration of your devices. Uyuni provides a set of modules called Uyuni configuration modules, that you can use to configure both SUSE Manager and Uyuni Servers.

You can use the Uyuni configuration modules directly or using SLS files. They are especially useful if you have multiple Uyuni Servers, for example in Hub installations, but they are also useful for smaller installations.

For more information about using Hub, see Specialized-guides › Large-deployments.

You can use Uyuni configuration modules to configure:

- Organizations
- Users
- User permissions
- System groups
- Activation Keys

For more information about Salt execution modules, see https://docs.saltstack.com/en/latest/topics/tutorials/modules.html.
For more information about Salt state modules, see https://docs.saltstack.com/en/latest/topics/tutorials/starting_states.html.

1.10.1. Install Configuration Modules

The Uyuni configuration modules are available in the `uyuni-config-modules` package. On the Uyuni Server, at the command prompt, as root, use this command:

```
zypper in uyuni-config-modules
```

This package also installs detailed API descriptions, indications on pillar settings, and examples. When you have installed the package, navigate to `/usr/share/doc/packages/uyuni-config-modules/`.

1.11. Salt Formulas

Formulas are collections of Salt States that contain generic parameter fields. Formulas allow for reliable reproduction of a specific configuration. Some formulas are supplied by SUSE, or you can install formulas from RPM packages or an external git repository.

Formulas work best for large, non-trivial, configurations. For smaller tasks, use a state rather than a formula. Formulas and states both act as a kind of configuration documentation. When you have written and stored the configuration, they provide a snapshot of your infrastructure.

Formula data can be managed using the XMLRPC API.

You can use the Uyuni Web UI to apply Uyuni formulas. The most commonly used formulas are documented in this section.

Alternatively, you can use pre-written formulas as a starting point for your own custom formulas. Pre-written formulas are available from https://github.com/saltstack-formulas.

For more information on custom formulas, see Specialized-guides › Salt.

1.11.1. Formulas Provided by Uyuni

Some formulas are installed by default with Uyuni. Other official formulas can be installed as RPM packages. When the formula is installed, you can activate them using the Uyuni Web UI.

For information about writing custom formulas, see Specialized-guides › Salt.

1.11.1.1. Install Formulas with Zypper

Formulas are provided in the Uyuni pool software channel.
If a formula uses the same name as an existing Salt state, the two names will collide, and could result in the formula being used instead of the state. Always check states and formulas to avoid name clashes.

**Procedure: Installing Formulas with Zypper**

1. On the Uyuni Server, at the command prompt, search for available formulas:

   ```bash
   zypper se --type package formula
   ```

2. Get more information about a formula:

   ```bash
   zypper info <formula_name>
   ```

3. On the Uyuni Server, at the command prompt, as root, install the formula:

   ```bash
   zypper in <formula_name>
   ```

**1.11.1.2. Activate Formulas from the Web UI**

Formulas provided by Uyuni, or formulas that you have installed, can be activated using the Uyuni Web UI.

**Procedure: Activate Formulas from the Web UI**

1. In the Uyuni Web UI, navigate to **Systems** › **List**, select the client you want to activate the formula for.

2. Navigate to the **Systems** › **Formulas** tab, and check the formula you want to activate.

3. Click **[Save]**.

4. Navigate to the new subtab for the formula, and configure the formula as required.

5. Apply the highstate.

**1.11.2. Bind Formula**

The Bind formula is used to configure the Domain Name System (DNS) on the branch server. POS terminals will use the DNS on the branch server for name resolution of Saltboot specific hostnames.

When you are configuring the Bind formula for a branch server with a dedicated internal network, check that you are using the same fully qualified domain name (FQDN) on both the external and internal branch networks. If the FQDN does not match on both networks, the branch server will not be recognized as a proxy server.
The following procedure outlines a standard configuration with two zones. Adjust it to suit your own environment.

Zone 1 is a regular domain zone. Its main purpose is to resolve Saltboot hostnames such as TFTP, FTP, or Salt. It can also resolve the terminal names if configured.

Zone 2 is the reverse zone of Zone 1. Its main purpose is to resolve IP addresses back to hostnames. Zone 2 is primarily needed for the correct determination of the FQDNs of the branch.

Procedure: Configuring Bind with Two Zones

1. Check the Bind formula, click Save, and navigate to the **Formulas** → **Bind** tab.
2. In the **Config** section, select **Include Forwarders**.
3. In the **Configured Zones** section, use these parameters for Zone 1:
   - In the **Name** field, enter the domain name of your branch network (for example: `branch1.example.com`).
   - In the **Type** field, select `master`.
4. Click **Add item** to add a second zone, and set these parameters for Zone 2:
   - In the **Name** field, use the reverse zone for the configured IP range (for example: `com.example.branch1`).
   - In the **Type** field, select `master`.
5. In the **Available Zones** section, use these parameters for Zone 1:
   - In the **Name** field, enter the domain name of your branch network (for example: `branch1.example.org`).
   - In the **File** field, type the name of your configuration file.
6. In the **Start of Authority (SOA)** section, use these parameters for Zone 1:
   - In the **Nameserver (NS)** field, use the FQDN of the branch server (for example: `branchserver.branch1.example.org`).
   - In the **Contact** field, use the email address for the domain administrator.
   - Keep all other fields as their default values.
7. In the **Records** section, in subsection **A**, use these parameters to set up an A record for Zone 1:
   - In the **Hostname** field, use the hostname of the branch server (for example: `branchserver`).
   - In the **IP** field, use the IP address of the branch server (for example, `192.168.1.5`).
8. In the **Records** section, subsection **NS**, use these parameters to set up an NS record for Zone 1:
   - In the input box, use the hostname of the branch server (for example: `branchserver`).
9. In the **Records** section, subsection **CNAME**, use these parameters to set up CNAME records for Zone 1:
   - In the **Key** field, enter `tftp`, and in the **Value** field, type the hostname of the branch server.
server (for example: branchserver).

◦ Click Add Item. In the Key field, enter ftp, and in the Value field, type the hostname of the branch server.

◦ Click Add Item. In the Key field, enter dns, and in the Value field, type the hostname of the branch server.

◦ Click Add Item. In the Key field, enter dhcp, and in the Value field, type the hostname of the branch server.

◦ Click Add Item. In the Key field, enter salt, and in the Value field, type the FQDN of the branch server (for example: branchserver.branch1.example.org).

10. Set up Zone 2 using the same parameters as for Zone 1, but ensure you use the reverse details:

◦ The same SOA section as Zone 1.

◦ Empty A and CNAME records.

◦ Additionally, configure in Zone 2:
  • Generate Reverse field by the network IP address set in branch server network formula (for example, 192.168.1.5/24).
  • For Zones should specify the domain name of your branch network (for example, branch1.example.org).

11. Click [Save Formula] to save your configuration.

12. Apply the highstate.

Reverse name resolution on terminals might not work for networks that are inside one of these IPv4 private address ranges:

- 10.0.0.0/8
- 172.16.0.0/12
- 192.168.0.0/16

If you encounter this problem, go to the Options section of the Bind formula, and click [Add item]:

- In the Options field, enter empty-zones-enable.
- In the Value field, select No.

1.11.3. Branch Network Formula

The Branch Network formula is used to configure the networking services required by the branch server, including DHCP, DNS, TFTP, PXE, and FTP.

1.11.3.1. Set Up a Branch Server Networking

The branch server can be configured to use networking in many different ways. The most common ways
provide either a dedicated or shared LAN for terminals.

1.11.3.1.1. Set Up a Branch Server with a Dedicated LAN

In this configuration, the branch server requires at least two network interfaces: one acts as a WAN to communicate with the SUSE Manager server, and the other one acts as an isolated LAN to communicate with terminals.

This configuration allows for the branch server to provide DHCP, DNS, TFTP, PXE, and FTP services to terminals. These services can be configured with Salt formulas in the SUSE Manager Web UI.

Procedure: Setting Up a Branch Server with a Dedicated LAN

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. In the Branch Network section, set these parameters:
   ◦ Keep Dedicated NIC checked.
   ◦ In the NIC field, enter the name of the network device that is connected to the internal LAN.
   ◦ In the IP field, enter the static IP address to be assigned to the branch server on the internal LAN.
   ◦ In the Netmask field, enter the network mask of the internal LAN.
3. Check Enable Route if you want the branch server to route traffic from internal LAN to WAN.
   ◦ Check Enable NAT if you want the branch server to convert addresses from internal LAN to WAN.
   ◦ Select the bind DNS forwarder mode.
   ◦ Check DNS forwarder fallback if you want to rely on an external DNS if the branch DNS fails.
   ◦ Specify the working directory, and the directory owner and group.

1.11.3.1.2. Set up a Branch Server with a Shared Network

In this configuration, the branch server has only one network interface card, which is used to connect to the SUSE Manager server as well as the terminals.

This configuration allows for the branch server to provide DNS, TFTP, PXE, and FTP services to terminals. These services can be configured with Salt formulas in the SUSE Manager Web UI. Optionally, the branch server can also provide DHCP services in this configuration.
If DHCP services are not provided by the branch server, ensure that your external DHCP configuration is set correctly:

- The next-server option must point to the branch server for PXE boot to work.
- The filename option must correctly identify the network boot program (by default, this is /boot/pxelinux).
- The domain-name-servers option must point to the branch server for correct host name resolution.

Procedure: Setting Up a Branch Server with a Shared Network

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. In the Branch Network section, set these parameters:
   - Keep Dedicated NIC unchecked.
   - Enable services on the branch server’s firewall. Ensure you include DNS, TFTP, and FTP services.
   - Select the bind DNS forwarder mode.
   - Check DNS forwarder fallback if you want to rely on an external DNS if the branch DNS fails.
   - Specify the working directory, and the directory owner and group.

1.11.3.2. Set up Branch Server Terminal Naming

In this configuration it is required to fill at least Branch Identification. This identifies Branch Server in Retail subsystem and is also used to better organize terminals with their respective branch servers.

Procedure: Setting up a Branch Server Identification

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. In the Terminal Naming section, enter the Branch Identification string.
3. Click [Save] to save your changes.
4. Apply the highstate.

It is also possible to set various options about terminal naming, for more information about terminal naming see Retail > Retail-terminal-names.

1.11.4. DHCPd Formula

The DHCPd formula is used to configure the DHCP service on the branch server.
Procedure: Configuring DHCP

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.

2. Check the Dhcpd formula, and click [Save].

3. Navigate to the Formulas › Dhcpd tab, and set these parameters:
   - In the Domain Name field, enter the domain name for the branch server (for example: branch1.example.com).
   - In the Domain Name Server field, enter either the IP address or resolvable FQDN of the branch DNS server (for example: 192.168.1.5).
   - In the Listen Interfaces field, enter the name of the network interface used to connect to the local branch network (for example: eth1).

4. Navigate to the Network Configuration (subnet) section, and use these parameters for Network1:
   - In the Network IP field, enter the IP address of the branch server network (for example: 192.168.1.0).
   - In the Netmask field, enter the network mask of the branch server network (for example: 255.255.255.0).
   - In the Domain Name field, enter the domain name for the branch server network (for example: branch1.example.com).

5. In the Dynamic IP Range section, use these parameters to configure the IP range to be served by the DHCP service:
   - In the first input box, set the lower bound of the IP range (for example: 192.168.1.51).
   - In the second input box, set the upper bound of the IP range (for example: 192.168.1.151).

6. In the Broadcast Address field, enter the broadcast IP address for the branch network (for example: 192.168.1.255).

7. In the Routers field, enter the IP address to be used by routers in the branch server network (for example: 192.168.1.5).

8. In the Next Server field, enter the hostname or IP address of the branch server (for example: 192.168.1.5).

9. In the Filename field, if you are booting a client using PXE, type the path to the PXE bootloader. There is usually no need to change the default value of /boot/pxelinux.0.

10. In the Filename Efi field, if you are booting a UEFI client using PXE, type the path to the PXE bootloader. There is usually no need to change the default value of /boot/shim.efi.

11. In the Filename Http field, if you are booting a UEFI client using HTTP, type http://branchserver/saltboot/boot/shim.efi.

12. Click [Save Formula] to save your configuration.

13. Apply the highstate.
1.11.5. Image Synchronization Formula

The Image Synchronization formula is used to configure when OS images are synchronized to the branch server, and to specify which images to synchronize.

If this formula is not enabled, synchronization must be started manually, and all images will be synchronized.

**Procedure: Configuring Image Synchronization**

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. Check the **Image Synchronize** formula, and click **[Save]**.
3. Navigate to the **Formulas › Image Synchronize** tab, and set these parameters:
   - Check the **Include Image Synchronization in Highstate** field to have image synchronization occur every time highstate is applied. This ensures that you do not have to perform image synchronization manually, however it requires a high bandwidth environment.
   - In the **Synchronize only the listed images** field, click **[Add item]** to add the images you want to have synchronized automatically. Alternatively, you can leave this list blank to have all images synchronized.
4. Click **[Save Formula]** to save your configuration.
5. Apply the highstate.

The Image Synchronization state does not delete cached images. If you are running out of disk space, check the size of the Salt client cache directory, and delete it if required. By default, the directory is located at `/var/cache/salt/minion`.

1.11.6. Liberate Formula

The liberate formula migrates systems from EL clients such as AlmaLinux, CentOS 7, Oracle Linux 9, Rocky Linux 9, or even Red Hat Enterprise Linux 9 to SUSE Liberty Linux. With this formula the conversion will take place during the client onboarding on Uyuni.

1.11.6.1. Install the formula

Install the formula with:

```
zypper in liberate-formula`
```

1.11.6.2. Configure Uyuni

To provide all the software channels for SUSE Liberty Linux on the Uyuni Server, proceed as follows.
Procedure: Providing SUSE Customer Center credentials

2. Navigate to My Organization, and select your organization.
3. Navigate to Users > Organization Credentials and take note of your organization username and password.
4. In the Uyuni Web UI, navigate to Admin > Setup Wizard > Organization Credentials to add the credentials to your Uyuni Server.
5. Click [Add new credential], and enter the SUSE Customer Center username and password noted in a previous step.

Procedure: Synchronizing the SLL/SLES-ES channels:

1. In the Uyuni Web UI, navigate to Admin > Setup Wizard > Products
2. Select the SUSE Liberty Linux Channels that you will use:
   - EL7: SUSE Linux Enterprise Server with Expanded Support 7 x86_64
   - EL8: RHEL or SLES ES or CentOS 8 Base
   - EL9: RHEL and Liberty 9 Base
3. Click the top right button [Add products].

Initial synchronization can take considerable time. You can check progress by accessing the server machine via SSH and monitoring the logs using:

tail -f /var/log/rhn/reposync/*

Procedure: Creating one Activation Key per SUSE Liberty Linux parent channel

1. Note: Activation Keys are the way to register systems and automatically assign them to the required software and configuration channels corresponding to them.
2. In the Uyuni Web UI, navigate to Systems > Activation Keys, and click the top right button [Create key].
3. In the Activation Key dialog, set the fields:

   **Description**
   
   Enter some text describing the activation key.

   **Key**
   
   Enter the identifier of the key. For example [literal sll9-default for EL 9 systems. Note: Keys will have a numeric prefix depending on the organization, so that they are unique.

   **Usage**
   
   Leave blank.
Base Channel

Select one base channel:

- EL7: RHEL x86_64 Server 7
- EL8: RHEL8-Pool for x86_64
- EL9: EL9-Pool for x86_64

Child Channel

Include all child channels.

Add-On system type

Leave all blank.

Contact Method

Default

Universal Default

Leave unchecked.

4. Click [Create Activation Key].

1.11.6.3. Add Liberate formula and assign it to activation keys

When installed, the formula can be assigned to an Activation Key by creating a System Group:

Procedure: Assigning system group and assigning liberate formula

1. In the Uyuni Web UI, navigate to Systems › System Groups, and click the [Create Group] button in top right corner.

2. In the dialog, fill in the following data:

   Name
   liberate

   Description
   Systems to be converted to SUSE Liberty Linux

3. From the liberate System Group page, navigate go to the Formulas tab.

4. Select the Liberate formula, and click [Save]. A new tab called Liberate will appear.

5. On the Liberate tab, you see the Reinstall all packages after conversion option. Keep it checked if you want to reinstall all the packages during the migration. This way you ensure all the packages will have SUSE signatures and no previous package will be kept. If you do not want to change the state of your system during the migration, uncheck this option and click the [Save Formula] button. In this case, you can re-install the packages later.
Now a system group exists that has assigned the Liberate formula. This formula will be applied only once to migrate the system to SUSE Liberty Linux, even if you run it multiple times. With the next procedure, assign the system group to the Activation Key.

Procedure: Assigning the system group to the Activation Key

1. In the Uyuni Web UI, navigate to Systems › Activation Keys.
2. Select the Activation Key, for example s119-default for the EL 9 systems.
3. From the Activation Key page navigate to the Groups › Join tab, select the libereate group, and click the [Join Selected Groups] button. The group will be assigned to the Activation Key.

Procedure: Applying migrate directly during registration

1. From the Activation Key page, navigate to the Details tab.
2. Navigate to the Configuration File Deployment section, and check the Deploy configuration files to systems on registration option.
3. Click [Update Activation Key].

When you register a system with this key it will perform the migration automatically.

1.11.6.4. Register a new system and proceed to the migration

There are two ways to onboard (or register) a new client with the Activation Key:

With the Web UI and selecting the activation key

This is intended for a one-off registration or for testing purposes.

With a bootstrap script with an assigned activation key

This is intended to be used for mass registration.

1.11.6.4.1. Register with the Web UI

Technically, this will start an SSH connection to the client and run the bootstrap script to register it.

Procedure: Registering with the Web UI and selecting the activation key

1. In the Uyuni Web UI, navigate to Systems › Bootstraping.
2. In the Bootstrap Minions dialog, fill the entries:

   Host
   
   Hostname of the client to register

   SSH Port
   
   Leave blank to use default, which is 22
User
Enter user or leave blank for root

Authentication Method
Select if you want to use password or provide a SSH Private Key

- Password: password to access the system
- SSH Private Key: file with the private key
- SSH Private Key Passphrase: In case a private key was provided that requires a passphrase to unlock, provide it here

Activation Key
Select from the menu the activation key to be used, for example sll9-default

Reactivation Key
Leave blank, it will not be used here

Proxy
Leave as None if you are not using a proxy

3. Click the [Bootstrap] button to start the registration.

A notification will show on top of the page stating that the client is being registered.

1.11.6.4.2. Register with a bootstrap script

Procedure: Creating bootstrap script

1. In the Uyuni Web UI, navigate to Admin › Manager Configuration › Bootstrap Script.

2. Fill the fields of the bootstrap script configuration dialog:

Uyuni/SUSE Manager server hostname
This should be set to the hostname that the client will use to reach the server, as well as the hostname Note: a Certificate will be used associated to this name for the client systems, as it was configured in the initial setup. If it’s changed, a new certificate shall be created

SSL cert location
Path, in the server, to the filename provided as a certificate to register it. Keep it as it is.

Bootstrap using Salt
Select this checkbox to apply salt states, like the one we added via configuration channel. It is required to perform the migration.

Enable Client GPG checking
Select this checkbox to ensure all packages installed come from the proper sources, in this case, SUSE Liberty Linux signed packages.
Enable Remote Configuration
Leave unchecked.

Enable Remote Commands
Leave unchecked.

Client HTTP Proxy
Leave blank. This is in case the client requires a proxy to access the server.

Client HTTP Proxy username
Leave blank.

Client HTTP Proxy password
Leave blank.

3. Click the [Update] button to refresh the bootstrap.sh script.

The bootstrap script generated is reachable via HTTP. For example, for a server named example.org it will be at https://example.org/pub/bootstrap/. Accessing the server via SSH, the bootstrap script is available in /srv/www/htdocs/pub/bootstrap/.

Procedure: Modifying and running the bootstrap script

1. Make a copy of the bootstrap script generated on the server in /srv/www/htdocs/pub/bootstrap/. For example:

```
cd /srv/www/htdocs/pub/bootstrap/
cp bootstrap.sh bootstrap-sll9.sh
```

2. Edit bootstrap-sll9.sh, and add the activation key created earlier. For example, sll9-default:

```
ACTIVATION_KEYS=sll9-default
```

3. Run the modified bootstrap-sll9.sh script. On the client, as root, run:

```
curl -Sks https://example.org/pub/bootstrap/bootstrap-sll9.sh | /bin/bash
```

For more information about running bootstrap scripts, see xref:client-configuration:registration-bootstrap.adoc#registering.clients.bootstrap.register[].

4. Apply highstate to migrate the client to SUSE Liberty Linux.
1.11.6.5. For already registered clients

Software channels, system group membership, and formulas can be assigned to any already registered client. This method makes use of the bootstrap script created above for onboarding new systems.

1. In the Uyuni Web UI, open the System Details page of any registered client you want to migrate to SUSE Liberty Linux.

2. Click the Reactivation tab. If there is already a key listed, you can use it. If not, click [Generate New Key], and copy the entire key. The key will start with [literal]re-`.

3. SSH into this client and set the environment variable to be the key that you copied:

   ```
   export REACTIVATION_KEY=re-xxxxxxxxxxxxxxxx
   ```

1. Run the bootstrap script you created above, and the system will re-register using the same profile as before, but with the newly assigned SUSE Liberty Linux context.

1.11.7. Monitoring Formula

The monitoring services in Uyuni are configured using formulas with forms. The package is installed by default, and contains these formulas:

- Grafana
- Prometheus
- Prometheus Exporters

For more information about using monitoring, see Administration › Monitoring.

1.11.7.1. Grafana

Procedure: Configuring the Grafana Formula

1. Navigate to the Formulas › Grafana tab, and set these parameters in the Grafana section:
   
   - Check the Enabled box to enable Grafana visualizations.

   Initial admin password is used on the first run only, and Grafana UI prompts to change it. This field cannot be used to change the Grafana password. For more information on how to change the password, see Administration › Monitoring.

2. For each Prometheus data source you want to use, in the Datasources › Prometheus section, click [+], and set these parameters:
   
   - In the Datasource name field, type a name to identify the data source.
   - In the Prometheus URL field, type the used protocol, the location of the Prometheus
server, and append port 9090. For example, http://example.com:9090. In case TLS encryption is enabled in Prometheus formula make sure to use https protocol and FQDN.

- In the fields Prometheus server username and Prometheus server password, enter basic authentication credentials for Prometheus server matching the ones in Prometheus formula.

3. In the Dashboards section, check the dashboards you want to use:
   - Uyuni server dashboard
   - Uyuni clients dashboard
   - PostgreSQL dashboard
   - Apache HTTPD dashboard
   - Kubernetes cluster dashboard
   - Kubernetes etcd dashboard
   - Kubernetes namespaces dashboard

4. Click [Save Formula] to save your configuration.

1.11.7.2. Prometheus

Procedure: Configuring the Prometheus Formula

1. Navigate to the Formulas > Prometheus tab, and set these parameters in the Prometheus section:
   - Check the Enabled box to enable Prometheus monitoring.
   - In the Scrape interval field, type the frequency of data scraping, in seconds. For example, 15 will scrape data every fifteen seconds.
   - In the Evaluation interval field, type the frequency of rules evaluation, in seconds. For example, 15 will evaluate alerting and aggregation rules every fifteen seconds.

2. In the TLS section, set these parameters:
   - Check the Enabled box to enable the secure configuration on Prometheus server.
   - In the Server Certificate field, type the path to the TLS server certificate.
   - In the Server Key field, type the path to the TLS server key.
   - In the User field, type the user name for Prometheus server.
   - In the Password Hash field, type the password for Prometheus server hashed with bcrypt.

3. In the Uyuni Server section, set these parameters:
   - Check the Enabled box to enable monitoring on this server.
   - Check the Autodiscover clients box to enable Prometheus to automatically find and monitor new clients when they are added to the server.
   - In the Username field, type the user name of the Prometheus account on the server.
In the **Password** field, type the password of the Prometheus account on the server.

In the **Targets TLS** section, set these parameters:

- Check the **Enabled** box to enable the secure configuration for auto-discovered targets.
- In the **CA Certificate** field, type the path to the Certificate Authority certificate.
- In the **Client Certificate** field, type the path to the TLS client certificate for authentication.
- In the **Client Key** field, type the path to the TLS client key for authentication.

4. In the **Alerting** section, set these parameters:

- Check the **Enable local Alertmanager service** box to enable the alert manager service.
- Check the **Use local Alertmanager** box to use the local alert manager service.

5. For each alert manager you want to use, in the **Alerting > Alertmanagers** section, click [+], and set these parameters:

- In the **IP Address:Port** field, type the location of the alert manager target, including the port number.

6. To use a rule file, in the **Alerting > Rule Files** section, click [+], and set these parameters:

- In the **Rule Files** field, type the location of the rule file you want to use.

7. To add a custom scrape configuration, in the **User defined scrape configurations** section, click [+], and set these parameters:

- In the **Job name** field, type a unique job name for your configuration.
- In the **Files** field, type the location pattern of file service discovery files you want to use.

For more information, see the upstream documentation [https://prometheus.io/docs/prometheus/latest/configuration/configuration/#file_sd_config](https://prometheus.io/docs/prometheus/latest/configuration/configuration/#file_sd_config).

8. Click **[Save Formula]** to save your configuration.

The formula does not generate and deploy the TLS certificates and keys. Ensure the files are present on the Salt client and readable for the user prometheus before applying the highstate. For more information about generating client and server certificates, see **Administration > Monitoring**.

### 1.11.7.3. Prometheus Exporters

**Procedure: Configuring the Prometheus Exporters Formula**

1. Navigate to the **Formulas > Prometheus Exporters** tab, and set these parameters in the **Node Exporter** section:

- Check the **Enabled** box to enable the node exporter.
- In the **Arguments** field, type any customized arguments for this exporter. For example,
2. In the Apache Exporter section:
   ◦ Check the Enabled box to enable the Apache exporter.
   ◦ In the Arguments field, type any customized arguments for this exporter. For example, 
     `--telemetry.address=:9117`.

3. In the Postgres Exporter section:
   ◦ Check the Enabled box to enable the PostreSQL exporter.
   ◦ In the Data source Name field, type the name of the data source to use.
   ◦ In the Arguments field, type any customized arguments for this exporter. For example, 
     `--web.listen-address=:9187`.

4. In the TLS section:
   ◦ Check the Enabled box to enable the secure configuration.
   ◦ In the CA Certificate field, type the path to the Certificate Authority certificate.
   ◦ In the Server Certificate field, type the path to the TLS server certificate.
   ◦ In the Server Key field, type the path to the TLS server key.

5. Click [Save Formula] to save your configuration.

### 1.11.7.3.1. File-based service discovery

It is possible to define monitored targets using file-based service discovery provided in the Prometheus formula. This is a basic example demonstrating the usage:

```json
[{
  "targets": [ "<client1>:9100", "<client2>:9100" ],
  "labels": {
    "role": "<suma-client>",
    "job": "<suma-refclient>"
  }
}, {
  "targets": [ "<server>:80" ],
  "labels": {
    "role": "<suma-server>",
    "job": "<suma-refhost>",
    "__metrics_path__": "/rhn/metrics"
  }
}]
```

For more information, see [https://prometheus.io/docs/guides/file-sd/](https://prometheus.io/docs/guides/file-sd/).

### 1.11.7.3.2. TLS certificates and keys

The formula does not generate and deploy the TLS certificates and keys. Ensure the files are present on the Salt client and readable for the user `prometheus` before applying the highstate. For more information about generating client and server certificates, see [Administration > Monitoring](#).
1.11.7.4. Activate Forms

When you have completed and saved all the forms, apply the highstate.

For more information about using monitoring, see Administration » Monitoring.

1.11.8. PXE Formula

The PXE formula is used to configure PXE booting on the branch server.

Procedure: Configuring PXE Booting

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. Select the Pxe formula, and click Save.
3. Navigate to the Formulas » Pxe tab, and set these parameters:
   - In the Kernel Filename field, keep the default value.
   - In the Initrd Filename field, keep the default value.
   - If the terminals connecting to this branch server are running arm64 architecture, check the Enable ARM64 UEFI boot box. Leave unchecked for x86-64.
   - In the Kernel Filename for ARM64 field, keep the default value.
   - In the Initrd Filename for ARM64 field, keep the default value.
   - In the Kernel Command Line Parameters field, keep the default value. For more information about possible values, see Saltboot Kernel Command Line Parameters.
   - In the PXE root directory field, enter the path to the Saltboot directory (for example, /srv/saltboot).
4. Click Save Formula to save your configuration.
5. Apply the highstate.

1.11.8.1. Saltboot Kernel Command Line Parameters

Saltboot supports common kernel parameters and Saltboot-specific kernel parameters. All the parameters can be entered in the Kernel Command Line Parameters field of the PXE formula.

**kiwidebug=1**

Starts a shell on tty2 during boot and enables debug logging in Salt.

Do not use this parameter in a production environment as it creates a major security hole. This parameter should be used only in a development environment for debug purposes.
**MASTER**

Overrides auto-detection of the Salt master. For example:

```
MASTER=myproxy.domain.com
```

**SALT_TIMEOUT**

Overrides the local boot fallback timeout if the Salt master does not apply the Saltboot state within this timeout (default: 60 seconds). For example:

```
SALT_TIMEOUT=300
```

**DISABLE_HOSTNAME_ID**

If the terminal has a hostname assigned by DHCP, it is by default used as a minion ID. Setting this option to 1 disables this mechanism, and SMBios information will be used as a minion ID.

**DISABLE_UNIQUE_SUFFIX**

Setting this option to 1 disables adding random generated suffix to terminal minion ID.

If you set this parameter make sure your terminal has either a unique hostname provided by DHCP and DNS, or the terminal hardware comes with a unique serial number stored in its SMBios memory. Otherwise there is a risk of terminal minion ID duplicity, and bootstrapping the minion will fail.

The following parameters (MINION_ID_PREFIX, salt_device, root) are usually autoconfigured and should be used only in specific conditions such as debugging or development:

**MINION_ID_PREFIX**

Branch ID set in the Branch Network formula form.

**salt_device**

Device that contains the Salt configuration.

**root**

Device that contains the already deployed root file system. Used for falling back to local boot.

### 1.11.9. Saltboot Formula

The Saltboot formula is used to configure disk images and partitioning for the selected hardware type.

- The Saltboot formula is meant to be used as a group formula. Enable and configure Saltboot formula for hardware type groups.

- To apply changes to a terminal, terminal needs to be restarted. Applying highstate does not have any effect on running terminals.
Procedure: Configuring the Hardware Type Group with Saltboot

1. Open the details page for your new hardware type group, and navigate to the Formulas tab.

2. Select the Saltboot formula and click [Save].

3. Navigate to the **Formulas** › **Saltboot** tab.

4. In the **Disk 1** section, set these parameters:
   - In the **Disk symbolic ID** field, enter a custom name for the disk (for example, disk1).
   - In the **Device type** field, select DISK.
   - In the **Disk device** field, select the device that corresponds to the device name on the target machine or asterisk *, see Disk Selection in Saltboot Formula.
   - In the **RAID level** field, leave it empty.
   - In the **Disk Label** field, select gpt.

5. In the **Partition** section, set these parameters for **Partition 1:**
   - In the **Partition symbolic ID** field, enter a custom name for the partition (for example, p1).
   - In the **Partition size** use value 500.
   - In the **Device mount point** use /boot/efi.
   - In the **Filesystem format** use vfat.
   - In the **OS Image to deploy** field, leave it empty.
   - In the **Partition encryption password** field, leave it empty.
   - In the **Partition flags** use boot.

6. In the **Partition** section, set these parameters for **Partition 2:**
   - In the **Partition symbolic ID** field, enter a custom name for the partition (for example, p2).
   - In the **Partition size** field, specify a size for the partition in Mebibytes (MiB).
   - In the **Device mount point** field, select a location to mount the partition (for example, /data).
   - In the **Filesystem format** field, select your preferred format (for example, xfs).
   - In the **OS Image to deploy** field, leave it empty.
   - In the **Partition encryption password** field, enter a password if you want to encrypt the partition.
   - In the **Partition flags** field, leave it empty.

7. In the **Partition** section, set these parameters for **Partition 3:**
   - In the **Partition symbolic ID** field, enter a custom name for the partition (for example, p3).
   - In the **Partition size** field, specify a size for the partition in Mebibytes (MiB).
In the Device mount point field, leave it empty.
In the Filesystem format field, select swap.
In the OS Image to deploy field, leave it empty.
In the Partition encryption password field, enter a password if you want to encrypt the partition.
In the Partition flags field, select swap.

8. In the Partition section, set these parameters for Partition 4:
   - In the Partition symbolic ID field, enter a custom name for the partition (for example, p4).
   - In the Partition size field, leave it empty. This will ensure the partition uses up all remaining space.
   - In the Device mount point field, select /.
   - In the Filesystem format field, leave it empty.
   - In the OS Image to deploy field, enter the name of the image to deploy.
   - In the Image version field, leave it empty. This will ensure you use the latest available version.
   - In the Partition encryption password field, enter a password if you want to encrypt the partition.
   - In the Partition flags field, leave it empty.

9. Click [Save Formula] to save your configuration.

1.11.9.1. Special Partition Types

The Saltboot formula helps you with setting up special partition types.

For terminal to be able to boot locally, either BIOS grub or EFI partition must be configured.

1.11.9.1.1. BIOS grub Partition

A BIOS grub partition is needed for local booting from a GPT disk on non-EFI machines. For more information, see https://en.wikipedia.org/wiki/BIOS_boot_partition.

In the formula, enter the following options:

<table>
<thead>
<tr>
<th>Partition Symbolic ID: p1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition Size (MiB): 50</td>
</tr>
<tr>
<td>Partition Flags: bios_grub</td>
</tr>
</tbody>
</table>

Leave the other fields empty.
1.11.9.1.2. EFI Partition

An EFI partition is needed for local booting on EFI machines. Partition Table Type must be GPT. For more information, see https://en.wikipedia.org/wiki/EFI_system_partition.

In the formula, enter the following options:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition Symbolic ID</td>
<td>p1</td>
</tr>
<tr>
<td>Partition Size (MiB)</td>
<td>500</td>
</tr>
<tr>
<td>Device Mount Point</td>
<td>/boot/efi</td>
</tr>
<tr>
<td>Filesystem Format</td>
<td>vfat</td>
</tr>
<tr>
<td>Partition Flags</td>
<td>boot</td>
</tr>
</tbody>
</table>

Leave the other fields empty.

1.11.9.2. Disk Selection in Saltboot Formula

When there is only one disk present on target hardware (including USB drives), use an asterisk * to automatically select the disk device.

When there are multiple disks, use an asterisk * in the device path. In this example, SATA disks are differentiated from USB disks:

```
/dev/disk/by-path/*-ata-1
/dev/disk/by-path/*usb*
```

If the entered value does not contain /, the entered value is automatically prepended by /dev/disk/by-path/. For example, *usb* is the same as /dev/disk/by-path/*usb*.

If you prefer to select specific devices, you can this format in the disk device field:

- symbolic names (for example: /dev/sda)
- by-path (for example: /dev/disk/by-path/..)
- by-id (for example: /dev/disk/by-id/...)

To see a list of available devices from the command prompt, press Esc while waiting for key approval.

1.11.9.3. Troubleshooting the Saltboot Formula

**msdos Disklabel Limitations**

On the msdos disk label, you can create a maximum of four primary partitions. Extended partitions are not supported. If you need more than four partitions, use the GPT disk label instead.

For more information on troubleshooting problems with the Saltboot formula, see Administration > Troubleshooting.
1.11.10. TFTPd Formula

The TFTPd formula is used to configure the TFTP service on the Uyuni for Retail branch server.

Procedure: Configuring TFTP

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. Select the Tftpd formula, and click [Save].
3. Navigate to the Formulas › Tftpd tab, and set these parameters:
   - In the Internal Network Address field, enter the IP address of the branch server (for example: 192.168.1.5).
   - In the TFTP Base Directory field, enter the path to the Saltboot directory (for example, /srv/saltboot).
   - In the Run TFTP Under User field, enter saltboot.
4. Click [Save Formula] to save your configuration.
5. Apply the highstate.

1.11.11. Virtualization Guest Formula

The Virtualization Guest formula is used to configure settings for virtual machine.

Open the Tuning pull-down checklist and select from the following virtual machine performance tuning settings:

- Disable IRQ balancing.
- Disable Kernel Samepage Merging (KSM). Reduces performance overhead by not sharing memory across virtual machines.
- Optimizations for KVM passed through host CPU. Requires the KVM hint-dedicated option to be set on the VM definition.

When configured, click [Save Formula] to save your configuration, and apply the highstate.

1.11.12. Virtualization Host Formula

The Virtualization Host formula is used to configure settings for a virtualization host.

Hypervisor

Select KVM or Xen as the hypervisor.

Create default storage pool

Create default virtual network

- Default pool: Open the pull-down list and enter the directory name of the default storage pool.
• Default net: Open the pull-down list and configure the default virtual network by setting the Mode (NAT or Bridge) and the Bridge name.

**Tuning**

Enable IOMMU support (x86_64).

When configured, click [Save Formula] to save your configuration, and apply the highstate.

### 1.11.13. VsFTPd Formula

The VsFTPd formula is used to configure the FTP service on the branch server.

**Procedure: Configuring VsFTPd**

1. In the SUSE Manager Web UI, open the details page for the branch server, and navigate to the Formulas tab.
2. Select the Vsftpd formula, and click [Save].
3. Navigate to the Formulas › Vsftpd tab, and set these parameters:
   - In the FTP server directory field, enter /srv/saltboot.
   - In the Internal Network Address field, enter the IP address of the branch server (for example: 192.168.1.5).
   - All other fields can retain their default values.
4. Click [Save Formula] to save your configuration.
5. Apply the highstate.

### 1.11.14. Yomi Formula

The Yomi (yet one more installer) installer for SUSE and openSUSE operating systems is configured using formulas with forms.

The yomi-formula package provides these formulas:

- Yomi
- Yomi Storage
- Yomi Bootloader
- Yomi Software
- Yomi Services
- Yomi Users

**Procedure: Install the Yomi Formulas with Forms**

1. On the Uyuni Server, at the command prompt, as root, install the yomi-formula package:
zypper in yomi-formula

2. Restart services:

systemctl restart salt-master.service

When the formula package is installed, you need to install the PXE Yomi image on the client, boot the client you want to provision, and enable the Yomi formulas on the client. For more information on preparing Yomi clients for provisioning, see Specialized-guides › Salt.

Procedure: Configuring the Yomi Formula

1. Navigate to the Formulas › Yomi tab, and set these parameters in the General Configuration section:
   - Check the Events box to allow monitoring.
   - In the Reboot field, select yes to instruct the client to reboot after installation.
   - Check the Snapper box if you are using the btrfs file system on the client.
   - In the Locale field, select the region and encoding for systemd to use on the client. For example: en_US.utf8 for US English and UTF-8.
   - In the Keymap field, select the appropriate keyboard layout. For example: us for a US keyboard layout.
   - In the Timezone field, select the timezone for the client to use. For example: America/New_York for EST.
   - In the Hostname field, enter the hostname for the client to use. Leave this blank if you are using DHCP to provide the hostname.
   - In the Machine Id field, enter a machine identification number for the client. Leave this blank to have systemd generate one automatically.
   - In the Target field, enter a systemd target unit.

2. Click [Save Formula] to save your configuration.

Procedure: Configuring the Yomi Storage Formula

1. Navigate to the Formulas › Yomi Storage tab, and set these parameters in the Partitions › Config section:
   - In the Labels field, select the default partition table type to use.
   - In the Initial Gap field, select the default amount of space to leave before the first partition. For example: 1 MB, or use 0 to leave no space between partitions.

2. For each device that you want to configure, in the Partitions › Devices section, click [+] and set these parameters:
   - In the Device field, type the mount point for the device. For example, /dev/sda.
In the **Label** field, select the partition table type to use, if it is different from the default label you selected.

In the **Initial Gap** field, select the amount of space to leave before the first partition, if it is different from the default space you specified.

3. For each partition that you want to create, in the **Partitions > Devices > Partitions** section, click **[+]**, and set these parameters:

   ◦ In the **Partition Number** field, enter a number for the partition. The number you enter here is appended to the device name to identify the partition. For example, partition number 1 on device /dev/sda can be identified as /dev/sda1.

   ◦ In the **Partition Name** field, enter a name for the partition. Leave this blank if you have entered a partition number in the previous field.

   ◦ In the **Partition Size** field, enter a size for the partition. For example: 500 MB. Use rest to use all the remaining free space.

4. For each file system that you want to create, in the **Filesystems** section, click **[+]**, and set these parameters:

   ◦ In the **Partition** field, select the partition to create the file system on. For example, /dev/sda1.

   ◦ In the **Filesystem** field, select the file system type to create.

   ◦ In the **Mountpoint** field, type the mount point for the file system. For example: / for root.

5. Click **[Save Formula]** to save your configuration.

If you want to use LVM or RAID on your devices, click **[+]** in the appropriate sections, and complete the details for your environment.

**Procedure: Configuring the Yomi Bootloader Formula**

1. Navigate to the **Formulas > Yomi Bootloader** tab, and set these parameters in the **Bootloader** section:

   ◦ In the **Device** field, type the path for the bootloader. For example, /dev/sda.

   ◦ In the **Timeout** field, select the number of seconds grub will wait before booting the default menu entry.

   ◦ In the **Kernel** field, type any additional kernel parameters you want to use. Any kernel parameters you add here will be appended to the **GRUB_CMDLINE_LINUX_DEFAULT** line during boot.

   ◦ In the **Terminal** field, type the terminal to use for both terminal input and output.

   ◦ In the **Serial Command** field, type parameters for using the serial port. Use this only if you are using the serial console as the default terminal.

   ◦ In the **Gfxmode** field, type the resolution to use for the graphical terminal. Use this only if you are using the graphical console as the default terminal.
Procedure: Configuring the Yomi Software Formula

1. Navigate to the Formulas › Yomi Software tab, and set these parameters in the Software › Configuration section:
   - Check the Minimal box to use a minimal installation, which only installs packages listed as Required.

2. For each repository that you want to add, in the Software › Repositories section, click [+], and set these parameters:
   - In the Repository Name field, type a name for the repository.
   - In the Repository URL field, type the location of the repository.

3. To add packages from each repository, in the Software › Packages section, click [+], and set these parameters:
   - In the Software › Packages field, type the names of the packages to install, or type a pattern to search for the appropriate packages. For example, pattern:enhanced_base glibc-locale, or kernel-default.

4. In the Software › Image section, set these parameters:
   - In the Image URL field, type the location of the operating system ISO image to use.
   - In the Md5 field, type the MD5 hash to use to verify the ISO.

5. In the SUSEConect › Config section, set these parameters:
   - In the Registration Code field, type the registration code for the client you are installing. You can obtain this code from SUSE Customer Center.
   - In the Email field, type the administrator email address to use.
   - In the Url field, type the address of the registration server to use. For example, use https://scc.suse.com, to register with SUSE Customer Center.
   - In the Version field, type the version of the product you are registering.

6. For each product that you want to register, in the SUSEConect › Products section, click [+], and set these parameters:
   - In the Product field, type each product you want to register. For example, <product_name>/1.1/x86, for version 1.1 with x86 architecture.
   - In the SUSEConect › Packages field, type the names of the packages to install, or type a pattern to search for the appropriate packages. For example, pattern:enhanced_base glibc-locale, or kernel-default.

7. Click [Save Formula] to save your configuration.

Procedure: Configuring the Yomi Services Formula
1. Navigate to the **Formulas > Yomi Services** tab, and set these parameters:
   - Check the **Install salt-minion** box to install and configure the client as a Salt client.

2. For each service you want to enable, in the **Services > Enabled** section, click [+], and set these parameters:
   - In the **Service** field, type the name of the service to enable. For example, **salt-minion**.

3. For each service you want to disable, in the **Services > Disabled** section, click [+], and set these parameters:
   - In the **Service** field, type the name of the service to disable.

4. Click **[Save Formula]** to save your configuration.

**Procedure: Configuring the Yomi Users Formula**

1. Navigate to the **Formulas > Yomi Users** tab.

2. For each user you want to create, in the **Users** section, click [+], and set these parameters:
   - In the **Username** field, type the name of the new user.
   - In the **Password Hash** field, type the hashed version of the password to use.

3. To add a certificate for each user, in the **Users > Certificates** section, click [+], and add the certificate to the **Certificate** field.

4. Click **[Save Formula]** to save your configuration.

When you have completed and saved all the forms, apply the highstate.

For more information about using Yomi, see **Specialized-guides > Salt**.

### 1.11.15. Custom Salt Formulas

This section contains information about writing custom formulas, including formulas with forms. You can write your own custom formulas, and make them available to client systems in the Uyuni Web UI.

For information about the formulas provided by Uyuni, see **Specialized-guides > Salt**.

#### 1.11.15.1. File Structure Overview

RPM-based formulas must be placed in a specific directory structure to ensure that they work correctly. A formula contains two separate directories: **states** and **metadata**. Folders in these directories need to have exactly matching names.

The formula states directory contains anything necessary for a Salt state to work independently. This includes .sls files, a map.jinja file and any other required files. This directory should only be modified by RPMs and should not be edited manually. For example, the **locale-formula states** directory is located in:
To create formulas with forms, the metadata directory contains a `form.yml` file. The `form.yml` file defines the forms for Uyuni. The metadata directory also contains an optional `metadata.yml` file with additional information about the formula. For example, the `locale-formula` metadata directory is located in:

```
/usr/share/susemanager/formulas/metadata/locale/
```

If you have a custom formula that is not in an RPM, it must be in a state directory configured as a Salt file root. Custom state formula data must be in:

```
/srv/salt/<custom-formula-name>/
```

Custom metadata information must be in:

```
/srv/formula_metadata/<custom-formula-name>/
```

All custom folders must contain a `form.yml` file. These files are detected as form recipes and are applied to groups and systems from the Web UI:

```
/srv/formula_metadata/<custom-formula-name>/form.yml
```

The Salt formula directory changed in Uyuni 4.0. The old directory location, `/usr/share/susemanager/formulas`, will continue to work for some time. You should ensure that you update to the new directory location, `/usr/share/salt-formulas/` as soon as possible.

1.11.15.2. Define Formula with Forms Data

Uyuni requires a file called `form.yml`, to describe how formula data should look within the Web UI. The `form.yml` file is used by Uyuni to generate the desired formula with forms, with values editable by a user.

The file contains a list of editable attributes that start with a `$` sign. These attributes are used to determine how to display the formula in the Uyuni Web UI.

For example, the `form.yml` that is included with the `locale-formula` is in:

```
/usr/share/susemanager/formulas/metadata/locale/form.yml
```

Part of that file looks like this:
All values that start with a $ sign are annotations used to display the UI that users interact with. These annotations are not part of pillar data itself and are handled as metadata.

This section lists the available attributes:

$\text{Type}$

The most important attribute is the $\text{type}$ attribute. It defines the type of the pillar value and the form-field that is generated. The supported types are:

- text
- password
- number
- url
- email
- date
- time
- datetime
- boolean
- color
- select
- group
- edit-group
- namespace
- hidden-group (obsolete, renamed to namespace)

The text attribute is the default and does not need to be specified explicitly.

Many of these values are self-explanatory:
• The text type generates a simple text field
• The password type generates a password field
• The color type generates a color picker

The group, edit-group, and namespace (formerly hidden-group) types do not generate an editable field and are used to structure form and pillar data. All these types support nesting.

The group and namespace types differ slightly. The group type generates a visible border with a heading. The namespace type shows nothing visually, and is only used to structure pillar data.

The edit-group type allows you to structure and restrict editable fields in a more flexible way. The edit-group type is a collection of items of the same kind. Collections can have these four shapes:

• List of primitive items
• List of dictionaries
• Dictionary of primitive items
• Dictionary of dictionaries

The size of each collection is variable. Users can add or remove elements.

For example, edit-group supports the $minItems and $maxItems attributes, which simplifies complex and repeatable input structures. These, and also itemName, are optional.

$default

Allows you to specify a default value to be displayed. This default value will be used if no other value is entered. In an edit-group it allows you to create initial members of the group and populate them with specified data.

$optional

This type is a Boolean attribute. If it is true and the field is empty in the form, then this field will not be generated in the formula data and the generated dictionary will not contain the field name key. If it is false and the field is empty, the formula data will contain a <field name>: null entry.

$ifEmpty

This type is used if the field is empty. This usually occurs because the user did not provide a value. The ifEmpty type can only be used when $optional is false or not defined. If $optional is true, then $ifEmpty is ignored. In this example, the DP2 string would be used if the user leaves the field empty:

displayName:
  $type: string
  $ifEmpty: DP2
$name

Allows you to specify the name of a value that is shown in the form. If this value is not set, the pillar name is used and capitalized without underscores and dashes. Reference it in the same section with ${name}.

$help and $placeholder

These attributes are used to give a user a better understanding of what the value should be. The $help type defines the message a user sees when hovering over a field. The $placeholder type displays a gray placeholder text in the field.

Use $placeholder only with text fields like text, password, email or date fields. Do not add a placeholder if you also use $default, as it will hide the placeholder.

$key

Applicable only if the edit-group has the shape of a dictionary. When the pillar data is a dictionary, the $key attribute determines the key of an entry in the dictionary.

For example:

```yaml
user_passwords:
  $type: edit-group
  $minItems: 1
  $prototype:
    $key:
      $type: text
      $type: text
    $default:
      alice: secret-password
      bob: you-shall-not-pass
```

Pillar:

```yaml
user_passwords:
  alice:
    secret-password
  bob:
    you-shall-not-pass
```

$minItems and $maxItems

In an edit-group, $minItems and $maxItems specifies the lowest and highest numbers for the group.

$itemName

In an edit-group, $itemName defines a template for the name to be used for the members of the group.

$prototype

In an edit-group, $prototype is mandatory and defines the default pre-filled values for newly added members in the group.
**$scope**

Specifies a hierarchy level at which a value may be edited. Possible values are `system`, `group`, and `readonly`.

The default value is `$scope: system`, allows values to be edited at group and system levels. A value can be entered for each system but if no value is entered the system will fall back to the group default.

The `$scope: group` option makes a value editable only for a group. On the system level you will be able to see the value, but not edit it.

The `$scope: readonly` option makes a field read-only. It can be used to show data to the user, but will not allow them to edit it. This option should be used in combination with the `$default` attribute.

**$visibleIf**

Deprecated in favor of `$visible`.

Allows you to show a field or group if a simple condition is met. An example condition is:

```plaintext
some_group#another_group#my_checkbox == true
```

The left part of the condition is the path to another value, and groups are separated by `#` signs. The middle section of the condition should be either `==` for a value to be equal or `!=` for values that should be not equal. The last field in the statement can be any value which a field should have or not have.

The field with this attribute associated with it will be shown only when the condition is met. In this example the field will be shown only if `my_checkbox` is checked. The ability to use conditional statements is not limited to checkboxes. It may also be used to check values of select-fields, text-fields, and similar.

A checkbox should be structured like this:

```plaintext
some_group:
    $type: group

another_group:
    $type: group

my_checkbox:
    $type: boolean
```

Relative paths can be specified using prefix dots. One dot indicates a sibling, two dots indicate a parent, and so on. This is mostly useful for `edit-group`. 
If you use multiple groups with the attribute, you can allow a user to select an option and show a completely different form, dependent upon the selected value.

Values from hidden fields can be merged into the pillar data and sent to the client. A formula must check the condition again and use the appropriate data. For example:

```python
show_option:
  $type: checkbox
some_text:
  $visibleIf: .show_option == true

{% if pillar.show_option %}
do_something:
  with: {{ pillar.some_text }}
{% endif %}
```

$\textit{values}$

Can only be used together with $\text{type}$ $\text{use}$ to specify the different options in the select-field. $\textit{values}$ must be a list of possible values to select. For example:

```python
select_something:
  $type: select
$\textit{values}: ["option1", "option2"]
```

Or:

```python
select_something:
  $type: select
$\textit{values}:
  - option1
  - option2
```
$visible

Allows you to show a field or group if a condition is met. You must use the jexl expression language to write the condition.

Example structure:

```yaml
some_group:
  $type: group
  another_group:
    $type: group
    my_checkbox:
      $type: boolean
```

An example condition is:

```python
formValues.some_group.another_group.my_checkbox == true
```

The field with this attribute will only show if the condition is met. In this example, the field will show only if `my_checkbox` is checked. You can also choose other elements for the conditional statement, such as select fields or text fields.

If you use multiple groups with the attribute, users can select an option that will show a completely different form, depending on the selected value.

Values from hidden fields can be merged into the pillar data and sent to the client. A formula must check the condition again and use the appropriate data. For example:

```python
show_option:
  $type: checkbox
some_text:
  $visible: this.parent.value.show_option == true

{% if pillar.show_option %}
do_something:
  with: {{ pillar.some_text }}
{% endif %}
```

$disabled

Allows you to disable a field or group if a condition is met. You must use the jexl expression language to write the condition.

If specified at group level it will disable all fields in that group.

$required

Fields with this attribute are mandatory. Supports using the jexl expression language.
**$match**

Allows using a regular expression to validate the content of a text field.

It supports the regular expression features existing in JavaScript.

Example:

```plaintext
hardware:
  $type: text
  $name: Hardware Type and Address
  $placeholder: Enter hardware-type hardware-address (for example, "ethernet AA:BB:CC:DD:EE:FF")
  $help: Hardware Identifier - prefix is mandatory
```

### 1.11.15.2.1. Expression language

You must use the **jexl** expression language to write conditions.

Given a structure like this:

```plaintext
some_group:
  $type: group
  another_group:
    $type: group
      my_checkbox:
        $type: boolean
```

An example condition is:

```plaintext
formValues.some_group.another_group.my_checkbox == true
```

Absolute paths must begin with `formValues`.

Specify relative paths using `this.parent.value` to define the value of the parent.

You can also refer to the parent of the parent, with `this.parent.parent.value`. This is mostly useful for `edit-group` elements.

Example for relative paths:
Click \[\text{Add}\] to fill the form with the default values.

The formula is called \texttt{hd-partitions} and will appear as \texttt{Hd Partitions} in the Web UI.
To remove the definition of a partition click the minus symbol in the title line of an inner group.

When you are finished, click **[Save Formula]**.
1.11.15.3. Writing Salt Formulas

Salt formulas are pre-written Salt states. You can use Jinja to configure formulas with pillar data.

Basic Jinja syntax is:

```
pillar.some.value
```

When you are sure a pillar exists, use this syntax:

```
salt['pillar.get']('some:value', 'default value')
```

You can also replace the pillar value with grains. For example, `grains.some.value`.

Using data this way makes the formula configurable. In this example, a specified package is installed in the `package_name` pillar:

```
install_a_package:
pkg.installed:
  - name: {{ pillar.package_name }}
```

You can also use more complex constructs such as `if/else` and `for-loops` to provide greater functionality:
Another example:

```jinja
{% for service in pillar.services %}
start_{{ service }}:
  service.running:
    - name: {{ service }}
{% endfor %}
```

Jinja also provides other helpful functions. For example, you can iterate over a dictionary:

```jinja
{% for key, value in some_dictionary.items() %}
do_something_with_{{ key }}: {{ value }}
{% endfor %}
```

You can have Salt manage your files (for example, configuration files for a program), and change them with pillar data.

In this example, Salt copies the file `salt-file_roots/my_state/files/my_program.conf` on the server to `/etc/my_program/my_program.conf` on the client and template it with Jinja:

```
/etc/my_program/my_program.conf:
  file.managed:
    - source: salt://my_state/files/my_program.conf
    - template: jinja
```

This example allows you to use Jinja in the file, like the previous example for states:

```jinja
some_config_option = {{ pillar.config_option_a }}
```

1.11.15.4. Separate Data

Separating data from a state can increase flexibility and make it easier to re-use. You can do this by writing values into a separate file named `map.jinja`. This file must be within the same directory as the state files.

This example sets `data` to a dictionary with different values, depending on which system the state runs on. It will also merge data with the pillar using the `some.pillar.data` value so you can access `some.pillar.data.value` by using `data.value`.

```jinja
{% if pillar.installSomething %}
something:
  pkg.installed
{% else %}
anotherPackage:
  pkg.installed
{% endif %}
```
You can choose to override defined values from pillars. For example, by overriding `some.pillar.data.package` in this example:

```bash
{% set data = salt['grains.filter_by']({
    'Suse': {
        'package': 'packageA',
        'service': 'serviceA'
    },
    'RedHat': {
        'package': 'package_a',
        'service': 'service_a'
    }
}), merge=salt['pillar.get']('some:pillar:data')) %}
```

When you have created a map file, you can maintain compatibility with multiple system types while accessing deep pillar data in a simpler way.

Now you can import and use `data` in any file. For example:

```bash
{% from "some_folder/map.jinja" import data with context %}
install_package_a:
    pkg.installed:
        - name: {{ data.package }}
```

You can define multiple variables by copying the `{% set ...%}` statement with different values and then merge it with other pillars. For example:

```bash
{% set server = salt['grains.filter_by']({
    'Suse': {
        'package': 'my-server-pkg'
    }
}), merge=salt['pillar.get']('myFormula:server')) %
{% set client = salt['grains.filter_by']({
    'Suse': {
        'package': 'my-client-pkg'
    }
}), merge=salt['pillar.get']('myFormula:client')) %
```

To import multiple variables, separate them with a comma. For example:

```bash
{% from "map.jinja" import server, client with context %}
```

For more information about conventions to use when writing formulas, see https://docs.saltstack.com/en/latest/topics/development/conventions/formulas.html.

1.11.15.5. Generated Pillar Data

Pillar data is generated by Uyuni when events occur like generating the highstate. You can use an external pillar script to generate pillar data for packages and group IDs, and include all pillar data for a system:
The process is executed like this:

1. The `suma_minion.py` script starts and finds all formulas for a system by checking the `group_formulas.json` and `server_formulas.json` files.

2. The script loads the values for each formula (groups and from the system) and merges them with the highstate. By default, if no values are found, a group overrides a system if `$scope: group`.

3. The script also includes a list of formulas applied to the system in a pillar named `formulas`.

This structure makes it possible to include states. In this example, the top file is specifically generated by the `mgr_master_tops.py` script. The top file includes a state called `formulas` for each system. This includes the `formulas.sls` file located in `
/usr/share/susemanager/formulas/states` or `/usr/share/salt-formulas/states/`. The content looks similar to this:

```
include: {{ pillar["formulas"] }}
```

This pillar includes all formulas that are specified in the pillar data generated from the external pillar script.

Formulas should be created directly after Uyuni is installed. If you encounter any problems with formulas check these things first:

- The external pillar script (`suma_minion.py`) must include formula data.
- Data is saved to `/srv/susemanager/formula_data` and the pillar and `group_pillar` sub-directories. These directories should be automatically generated by the server.
- Formulas must be included for every client listed in the top file. Currently this process is initiated by the `mgr_master_tops.py` script which includes the `formulas.sls` file located in `/usr/share/susemanager/formulas/states/` or `/usr/share/salt-formulas/states/`. This directory must be a salt file root. File roots are configured on the salt-master (Uyuni) located at `/etc/salt/master.d/susemanager.conf`.

### 1.12. Salt SSH

Salt SSH allows Salt commands and states to be issued directly over SSH. SSH connections are created on demand, when the server executes an action on a client.

For more information about Salt SSH, see [https://docs.saltstack.com/en/latest/topics/ssh/](https://docs.saltstack.com/en/latest/topics/ssh/).

#### 1.12.1. SSH Connection Methods

In Uyuni there are two SSH connection methods, `ssh-push` and `ssh-push-tunnel`. In both
methods the server initiates an SSH connection to the client to execute a Salt call.

In the ssh-push method, the package manager works as normal, and the HTTP or HTTPS connection is directly created.

In the ssh-push-tunnel method, the server creates an HTTP or HTTPS connection through an SSH tunnel. The HTTP connection initiated by the package manager is redirected through the tunnel using /etc/hosts aliasing. Use this method for in-place firewall environments that block HTTP or HTTPS connections between server and client.

1.12.2. Salt SSH Integration

As with all Salt calls, Uyuni invokes salt-ssh via the salt-api.

Salt SSH relies on a roster to obtain details such as hostname, ports, and the SSH parameters of a client. Uyuni keeps these details in the database and makes them available to Salt SSH by using the uyuni roster module, or by generating a temporary roster file on bootstrapping new clients with the Web UI. The location of the temporary roster file is supplied to Salt SSH using the roster-file option. For the registered clients roster set to uyuni is used instead to get the roster from the database with uyuni salt roster module.

It is not recommended to run salt-ssh as root user. This can cause permission issues the next time Uyuni tries to use Salt SSH. Use mgr-salt-ssh, which changes the effective user to salt and avoids file permission issues. If you want to use mgr-salt-ssh with a user other than root, the user should have the permission to change effective user to salt.

mgr-salt-ssh uses roster set to uyuni by default, if neither roster nor roster-file specified in the command line. It helps to call Salt commands to the registered Salt SSH clients with no roster file generation.

1.12.3. Authentication

Salt SSH supports both password and key authentication. Uyuni uses both methods:

Password authentication is used only when bootstrapping. During the bootstrap step the key of the server is not authorized on the client and therefore a password must be used for a connection to be made. The password is used transiently in a temporary roster file used for bootstrapping and the roster file is removed on finishing processing the event. This password is not stored.

All other common Salt calls use key authentication. During the bootstrap step the SSH key of the server is authorized on the client and added to the client /.ssh/authorized_keys file. Subsequent calls no longer require a password.

1.12.4. User Account

The user for Salt SSH calls made by Uyuni is taken from the ssh_push_sudo_user setting. By
default, the user is root.

If bootstrapping with default settings fail, check whether the client allows root login with ssh.

If the value of `ssh_push_sudo_user` is not root, then the `--sudo` options of `salt-ssh` are used. For this user you must configure the `NOPASSWD` option in the `sudoers` file. At least, set the python binary with the version number; for example:

```
<USER> ALL=(ALL) NOPASSWD:/usr/bin/python3.6
```

### 1.12.5. HTTP Redirection

The `ssh-push-tunnel` method requires traffic to be redirected through an SSH tunnel. This allows traffic to bypass firewalls blocking a direct connection between the client and the server.

This is achieved by using port 1233 in the repository URL:

```
https://suma-server:1233/repourl...
```

You can alias the `suma-server` hostname to `localhost` in `/etc/hosts`:

```
127.0.0.1 localhost suma-server
```

The server creates a reverse SSH tunnel that connects `localhost:1233` on the client to `suma-server:443`:

```
ssh ... -R 1233:suma-server:443
```

This means that the package manager will actually connect to `localhost:1233`, which is then forwarded to `suma-server:443` by the SSH tunnel.

The package manager can contact the server only if the tunnel is open, which occurs only when the server executes an action on the client.

Manual package manager operations that require server connectivity are not possible in this case.

### 1.12.6. Call Sequence

Salt SSH calls run in this sequence:

1. Set `roster` parameter to `uyuni` for registered clients or prepare the Salt roster on bootstrapping for the call
1. Create remote port forwarding option if the contact method is ssh-push-tunnel

2. Compute the ProxyCommand if the client is connected through a proxy

3. Create Roster content

The roster content contains:

- hostname
- user
- port
- remote_port_forwards: The remote port forwarding SSH option
- ssh_options: Other ssh options:
  - ProxyCommand: If the client connects through a proxy
- timeout: defaults to 180 seconds
- minion_opts:
  - master: Set to the minion ID if the contact method is ssh-push-tunnel
- ssh_pre_flight: The path to the shell script executed on the client before running any Salt command
- ssh_pre_flight_args: The list of arguments to call the pre flight script on the client

1.12.7. Bootstrap Sequence

This section describes the sequence of events when clients are registered to a Salt master. While bootstrapping is a type of Salt SSH call, the sequence differs slightly from regular SSH calls.

Bootstrapping uses Salt SSH for communication between the master and the client. This happens for both regular and SSH clients.

1. For a regular Salt client, generate and pre-authorize the Salt key of the client.

2. For an SSH client, if a proxy was selected, retrieve the SSH public key of the proxy using the mgrutil.chain_ssh_cmd runner. The runner copies the public key of the proxy to the server using SSH. If needed, it can chain multiple SSH commands to reach the proxy across multiple hops.

3. Generate pillar data for bootstrap. The pillar data is compiled and stored on the Salt master, and retrieved by the client.

4. Generate the roster for bootstrapping into a temporary file on the client. You can use the roster by passing it to the Salt API, with this command:
For `bootstrap_state`, use `bootstrap` for regular clients or `ssh_bootstrap` for SSH clients.

The way the client retrieves the pillar data depends on the contact method you have chosen for your client:

- If you are using the `ssh-push-tunnel` contact method, ensure you have completed the remote port forwarding option.
- If the client connects through a proxy, ensure you have completed the `ProxyCommand` option. This depends on your proxy configuration, including how many proxies you need to connect through.

Pillar data contains:

- `mgr_server`: The hostname of the Salt master
- `mgr_origin_server`: The hostname of the Uyuni Server
- `minion_id`: The hostname of the client to bootstrap
- `contact_method`: The connection type
- `mgr_sudo_user`: The user for `salt-ssh`
- `activation_key`: If selected
- `minion_pub`: The pre-authorized public client key
- `minion_pem`: The pre-authorized private client key
- `proxy_pub_key`: The public SSH key that was retrieved from the proxy if the target is an SSH client and a proxy was selected

The roster content contains:

- `hostname`
- `user`
- `password`
- `port`
- `remote_port_forwards`: the remote port forwarding SSH option
- `ssh_options`: other SSH options:
  - `ProxyCommand` if the client connects through a proxy
- `timeout`: defaults to 180 seconds
- `ssh_pre_flight`: The path to the `pre_flight` shell script (default: `/usr/share/susemanager/salt-ssh/preflight.sh`)
• `ssh_pre_flight_args`: The list of arguments to call the pre flight script on the client

This image provides an overview of the Salt SSH bootstrap process.

**Figure 1. Salt SSH Bootstrap Process**

1.12.8. Proxy Support

Salt SSH works with Uyuni Proxy by chaining the SSH connection from one server or proxy to the next. This is also known as a multi-hop or multi-gateway SSH connection.

Uyuni uses `ProxyCommand` to redirect SSH connections through proxies. This option invokes an arbitrary command that is expected to connect to the SSH port on the target host. The SSH process uses standard input and output of the command to communicate with the remote SSH daemon.

`ProxyCommand` replaces a TCP/IP connection. It does not perform any authorization or encryption. Its role is simply to create a byte stream to the remote SSH daemon port.
This image depicts a client connecting to a server that is behind a gateway. In this example `netcat` is used to pipe port 22 of the target host into the SSH standard input/output:

```
ssh -o ProxyCommand=<stdio/stdout to remote port> ... 
ssh -o ProxyCommand=’ssh gateway nc foo.com 22’ root@foo.com
```

The Salt SSH calls run in this sequence when a proxy is in use:

1. Uyuni initiates the SSH connection.
2. ProxyCommand uses SSH to create a connection from the server to the client through the proxies.

This example uses ProxyCommand with two proxies and the `ssh-push` method:

```
# Connect the server to the first proxy:
/usr/bin/ssh -i /srv/susemanager/salt/salt_ssh/mgr_ssh_id -o StrictHostKeyChecking=no -o User=mgrsshtunnel proxy1

# Connect the first proxy to the second, and forward standard input/output on the client to client:22 using the `\'-W\'` option:
/usr/bin/ssh -i /var/lib/spacewalk/mgrsshtunnel/.ssh/id_susemanager_ssh_push -o StrictHostKeyChecking=no -o User=mgrsshtunnel -W client:22 proxy2
```
This example uses `ProxyCommand` with two proxies and the `ssh-push-tunnel` method:

```plaintext
ssh -i salt_ssh_id -o ProxyCommand="ssh -i ssh_push_id proxyOne ssh -i
  ssh_push_id proxyTwo -W minion:22" root@minion <cmd>
```

```
# Connect the server to the first proxy:
/usr/bin/ssh -i /srv/susemanager/salt/salt_ssh/mgr_ssh_id -o
  User=mgrsshtunnel  proxy1

# Connect the first proxy to the second:
/usr/bin/ssh -i /home/mgrsshtunnel/.ssh/id_susemanager_ssh_push -o
  User=mgrsshtunnel  proxy2

# Connect the second proxy to the client and open an reverse tunnel (-R
1233:proxy2:443) from the client to the HTTPS port on the second proxy:
/usr/bin/ssh -i /home/mgrsshtunnel/.ssh/id_susemanager_ssh_push -o User=root
  -R 1233:proxy2:443 client

# Connect the client to itself and forward the standard input/output of the
server to the SSH port of the client (-W client:22). This is equivalent to `ssh ...
proxy2 netcat client 22` and is needed because SSH does not allow both the reverse tunnel (-R 1233:proxy2:443) and
the standard input/output forward (-W client:22) in the same command:
/usr/bin/ssh -i /root/.ssh/mgr_own_id -W client:22 -o User=root client
```
1.12.9. Users and SSH Key Management

To connect to a proxy, the parent server or proxy uses a specific user called `mgrsshtunnel`. When `mgrsshtunnel` connects, the SSH configuration of the proxy will force the execution of `/usr/sbin/mgr-proxy-ssh-force-cmd`. This is a simple shell script that allows only the execution of `scp`, `ssh`, or `cat` commands.

The connection to the proxy or client is authorized using SSH keys in this sequence:

1. The server connects to the client and to the first proxy using the key in `~/srv/susemanager/salt/salt_ssh/mgr_ssh_id`.
2. Each proxy has its own key pair in `~/home/mgrsshtunnel/.ssh/id_susemanager_ssh_push`.
3. Each proxy authorizes the key of the parent proxy or server.
4. The client authorizes its own key.
1.12.10. Repository Access with a Proxy

When Uyuni connects to a repository using a proxy, it can use either `ssh-push` or `ssh-push-tunnel`.

In both methods the client connects to the proxy to retrieve package and repository information.

In the `ssh-push` method, the package manager connects directly to the proxy using HTTP or HTTPS. This works in cases where there is no firewall between the client and the proxy that blocks HTTP connections initiated by the client.

In the `ssh-push-tunnel` method, the HTTP connection to the proxy is redirected through a reverse SSH tunnel.

In the `ssh-push-tunnel` method, the HTTP connection to the proxy is redirected through a reverse SSH tunnel.
1.12.11. Proxy Setup

When the spacewalk-proxy package is installed on the proxy, the mgrsshtunnel user is created.

The initial configuration with configure-proxy.sh occurs using this sequence:

1. An SSH key pair is generated, or an existing keypair is imported.
2. The SSH key of the parent server or proxy is retrieved to authorize it on the proxy.
3. The ssh daemon on the proxy is configured to restrict the mgrsshtunnel user. This is done by the mgr-proxy-ssh-push-init script, which is called from configure-proxy.sh. It does not have to be manually invoked.

The parent key is retrieved by calling an HTTPS endpoint on the parent server or proxy. The first endpoint tried is https://$PARENT/pub/id_susemanager_ssh_push.pub. If the parent is a proxy then this will return the public SSH key of the proxy.

If a 404 error is received from that endpoint, then the parent is assumed to be a server not a proxy, and https://$PARENT/rhn/manager/download/saltssh/pubkey is tried instead.

If an SSH key exists at /srv/susemanager/salt/salt_ssh/mgr_ssh_id.pub on the server it is returned.

If the public key does not exist because salt-ssh has not been invoked yet, a key will be generates by calling the mgrutil.ssh_keygen runner.

Salt SSH generates a keypair the first time it is invoked with /srv/susemanager/salt/salt_ssh/mgr_ssh_id. The sequence in this section is needed if a proxy is configured before Salt SSH was invoked for the first time.
1.12.12. Rotate SSH keys

The SSH key is used on salt-ssh managed systems. Additionally, it is used on Uyuni Proxies for the user mgrssh_tunnel. Normal systems managed with the Salt default method are not affected and do not have this key configured.

Procedure: Rotating SSH keys

1. On the Uyuni Server, as user root, change to user salt:

```
su -s /bin/bash - salt
```

2. Create a new SSH key:

```
ssh-keygen -N "" -t rsa -q -f /var/lib/salt/.ssh/new_mgr_ssh_id
```

3. Copy the public key into the Salt filesystem to make it usable in a Salt state:

```
cp /var/lib/salt/.ssh/new_mgr_ssh_id.pub /srv/susemanager/salt/salt_ssh/
```

4. Change back to user root again:

```
exit
```

5. Rollout the new key to all systems that need it. Applying the util.mgr_rotate_saltssh_key state will limit the changes to salt-ssh managed systems and proxies:

```
salt '*' state.apply util.mgr_rotate_saltssh_key
mgr-salt-ssh '*' state.apply util.mgr_rotate_saltssh_key
```

6. Move the old key away and make the new key the standard key. Rename mgr_ssh_id key to disabled_mgr_ssh_id and new_mgr_ssh_id key to mgr_ssh_id in the SSH keystore of the user salt as well as in the Salt filesystem for the public keys:

```
su -s /bin/bash - salt
cd .ssh
mv mgr_ssh_id disabled_mgr_ssh_id
mv mgr_ssh_id.pub disabled_mgr_ssh_id.pub
mv new_mgr_ssh_id mgr_ssh_id
mv new_mgr_ssh_id.pub mgr_ssh_id.pub
cd /srv/susemanager/salt/salt_ssh/
mv mgr_ssh_id.pub disabled_mgr_ssh_id.pub
mv new_mgr_ssh_id.pub mgr_ssh_id.pub
```

7. OPTIONAL: When containerized proxies exist, re-create the configuration to get the new SSH key.
into the proxy configuration. Restart the containers with the new configuration. It is also possible to change the existing configuration on the podman host (ssh.yaml). Change the value of server_ssh_key_pub with the content of the current mgr_ssh_id.pub.

8. To remove the disabled keys from the authorized_keys’ files of the [literal]salt-ssh” managed systems and the proxies, apply the state a second time:

```
salt '*' state.apply util.mgr_rotate_saltssh_key
mgr-salt-ssh '*' state.apply util.mgr_rotate_saltssh_key
```

1.13. Salt Rate Limiting

Salt is able to run commands in parallel on a large number of clients. This can potentially create large amounts of load on your infrastructure. You can use these rate-limiting parameters to control the load in your environment.

These parameters are all configured in the /etc/rhn/rhn.conf configuration file.

Salt commands that are executed from the command line are not subject to these parameters.

1.13.1. Batching

There are two parameters that control how actions are sent to clients, one for the batch size, and one for the delay.

When the Uyuni Server sends a batch of actions to the target clients, it will send it to the number of clients determined in the batch size parameter. After the specified delay period, commands will be sent to the next batch of clients. The number of clients in each subsequent batch is equal to the number of clients that have completed in the previous batch.

Choosing a lower batch size will reduce system load and parallelism, but might reduce overall performance for processing actions.

The batch size parameter sets the maximum number of clients that can execute a single action at the same time. Adjust the java.salt_batch_size parameter. Defaults to 200.

Increasing the delay increases the chance that multiple clients will have completed before the next action is issued (more clients are grouped together in subsequent batches), resulting in fewer overall commands, and reducing load.

The batch delay parameter sets the amount of time, in seconds, to wait after a command from the previous batch is processed before beginning to process the command on the next client. Adjust the java.salt_batch_delay parameter. Defaults to 1.0 seconds.
1.13.2. Disabling the Salt Mine

In older versions, Uyuni used a tool called Salt mine to check client availability. The Salt mine would cause clients to contact the server every hour, which created significant load. With the introduction of a more efficient mechanism in Uyuni 3.2, the Salt mine is no longer required. Instead, the Uyuni Server uses Taskomatic to ping only the clients that appear to have been offline for twelve hours or more, with all clients being contacted at least once in every twenty four hour period by default. You can adjust this by changing the `web.system_checkin_threshold` parameter in `rhn.conf`. The value is expressed in days, and the default value is 1.

Newly registered Salt clients will have the Salt mine disabled by default. If the Salt mine is running on your system, you can reduce load by disabling it. This is especially effective if you have a large number of clients.

Disable the Salt mine by running this command on the server:

```bash
salt '*' state.sls util.mgr_mine_config_clean_up
```

This will restart the clients and generate some Salt events to be processed by the server. If you have a large number of clients, handling these events could create excessive load. To avoid this, you can execute the command in batch mode with this command:

```bash
salt --batch-size 50 '*' state.sls util.mgr_mine_config_clean_up
```

You will need to wait for this command to finish executing. Do not end the process with `Ctrl+C`.

1.14. Scaling Minions (Large Scale Deployments)

Uyuni is designed by default to work on small and medium scale installations. For installations with more than 1000 clients per Uyuni Server, adequate hardware sizing and parameter tuning must be performed.

For more information on managing large scale deployments, see `Specialized-guides › Large-deployments`.
Chapter 2. Large Deployments Guide Overview

Updated: 2024-04-29

Uyuni is designed by default to work on small and medium scale installations. For installations with more than 1000 clients per Uyuni Server, adequate hardware sizing and parameter tuning must be performed.

There is no hard maximum number of supported systems. Many factors can affect how many clients can reliably be used in a particular installation. Factors can include which features are used, and how the hardware and systems are configured.

There are two main ways to manage large scale deployments. You can manage them with a single Uyuni Server, or you can use multiple servers in a hub. Both methods are described in this book.

With large scale environments one should also consider the usage of Uyuni Proxies. Sizing and location of the Proxies will dependent on the deployment topology. For more information see Installation-and-upgrade › Install-proxy.

Additionally, if you are operating within a Retail environment, you can use Uyuni for Retail to manage large deployments of point-of-service terminals. There is an introduction to Uyuni for Retail in this book.

Tuning and monitoring large scale deployments can differ from smaller installations. This book contains guidance for both tuning and monitoring within larger installations.

2.1. Hardware Requirements

Not all problems can be solved with better hardware, but choosing the right hardware is an absolute necessity for large scale deployments.

The minimum requirements for the Uyuni Server are:

- Eight or more recent x86-64 CPU cores.
- 32 GiB RAM. For installations with thousands of clients, use 64 GB or more.
- Fast I/O storage devices, such as locally attached SSDs. For PostgreSQL data directories, we recommend locally attached RAID-0 SSDs.

If the Uyuni Server is virtualized, enable the elevator=none kernel command line option, for the best input/output performance. You can check the current status with cat /sys/block/<DEVICE>/queue/scheduler. This command will display a list of available schedulers with the currently active one in brackets. To change the scheduler before a reboot, use echo none > /sys/block/<DEVICE>/queue/scheduler.

The minimum requirements for the Uyuni Proxy are:

- One Uyuni Proxy per 500-1000 clients, depending on available network bandwidth.
- Two or more recent x86-64 CPU cores.
2.2. Using a Single Server to Manage Large Scale Deployments

This section discusses how to set up a single Uyuni Server to manage a large number of clients. It contains some recommendations for hardware and networking, and an overview of the tuning parameters that you need to consider in a large scale deployment.

2.2.1. Operation Recommendations

This section contains a range of recommendations for large scale deployments.

Always start small and scale up gradually. Monitor the server as you scale to identify problems early.

2.2.1.1. Client Onboarding Rate

The rate at which Uyuni can onboard clients is limited and depends on hardware resources. Onboarding clients at a faster rate than Uyuni is configured for will build up a backlog of unprocessed keys. This slows down the process and can potentially exhaust resources. We recommend that you limit the acceptance key rate programmatically. A safe starting point would be to onboard a client every 15 seconds. You can do that with this command:
2.2.1.2. Clients and the RNG

All communication to and from clients is encrypted. During client onboarding, Salt uses asymmetric cryptography, which requires available entropy from the Random Number Generator (RNG) facility in the kernel. If sufficient entropy is not available from the RNG, it will significantly slow down communications. This is especially true in virtualized environments. Ensure enough entropy is present, or change the virtualization host options.

You can check the amount of available entropy with the `cat /proc/sys/kernel/random/entropy_avail`. It should never be below 100-200.

2.2.1.3. Clients Running with Unaccepted Salt Keys

Idle clients which have not been onboarded, that is clients running with unaccepted Salt keys, consume more resources than idle clients that have been onboarded. Generally, this consumes about an extra 2.5 Kb/s of inbound network bandwidth per client. For example, 1000 idle clients will consume about 2.5 Mb/s extra. This consumption will reduce almost to zero when onboarding has been completed for all clients. Limit the number of non-onboarded clients for optimal performance.

2.2.1.4. Disabling the Salt Mine

In older versions, Uyuni used a tool called Salt Mine to check client availability. The Salt Mine would cause clients to contact the server every hour, which created significant load. Since the introduction of a more efficient mechanism in Uyuni 3.2, the Salt Mine is no longer required. Instead, the Uyuni Server uses Taskomatic to ping only the clients that appear to have been offline for twelve hours or more, with all clients being contacted at least once in every twenty four hour period by default. You can adjust this by changing the `web.system_checkin_threshold` parameter in `rhn.conf`. The value is expressed in days, and the default value is 1.

Newly registered clients will have the Salt Mine disabled by default. If the Salt Mine is running on your system, you can reduce load by disabling it. This is especially effective if you have a large number of clients.

Disable the Salt Mine by running this command on the server:

```
salt '*' state.sls util.mgr_mine_config_clean_up
```

This will restart the clients and generate some Salt events to be processed by the server. If you have a large number of clients, handling these events could create excessive load. To avoid this, you can execute the command in batch mode with this command:

```
salt --batch-size 50 '*' state.sls util.mgr_mine_config_clean_up
```
You will need to wait for this command to finish executing. Do not end the process with Ctrl+C.

2.2.1.5. Disable Unnecessary Taskomatic Jobs

To minimize wasted resources, you can disable non-essential or unused Taskomatic jobs.

You can see the list of Taskomatic jobs in the Uyuni Web UI, at Admin › Task Schedules.

To disable a job, click the name of the job you want to disable, select Disable Schedule, and click [Update Schedule].

To delete a job, click the name of the job you want to delete, and click [Delete Schedule].

We recommend disabling these jobs:

- Daily comparison of configuration files: compare-configs-default
- Hourly synchronization of Cobbler files: cobbler-sync-default
- Daily gatherer and subscription matcher: gatherer-matcher-default

Do not attempt to disable any other jobs, as it could prevent Uyuni from functioning correctly.

2.2.1.6. Swap and Monitoring

It is especially important in large scale deployments that you keep your Uyuni Server constantly monitored and backed up.

Swap space use can have significant impacts on performance. If significant non-transient swap usage is detected, you can increase the available hardware RAM.

You can also consider tuning the Server to consume less memory. For more information on tuning, see Specialized-guides › Salt.

2.2.1.7. AES Key Rotation

Communications from the Salt Master to clients is encrypted with a single AES key. The key is rotated when:

- The salt-master process is restarted, or
- Any minion key is deleted (for example, when a client is deleted from Uyuni)

After the AES key has been rotated, all clients must re-authenticate to the master. By default, this happens next time a client receives a message. If you have a large number of clients (several thousands), this can cause a high CPU load on the Uyuni Server. If the CPU load is excessive, we recommend that you delete keys in batches, and in off-peak hours if possible, to avoid overloading the server.

For more information, see:
2.3. Multiple Servers with Hub to Manage Large Scale Deployments

If you need to manage a large number of clients, in most cases you can do so with a single Uyuni Server, tuned appropriately. However, if you need to manage tens of thousands of clients, you might find it easier to use multiple Uyuni Servers, in a hub, to manage them.

Uyuni Hub helps you manage very large deployments. The typical Hub topology looks like this:

![Uyuni Hub Diagram]

To copy contents between servers, you can use Inter-Server Synchronization (ISS). ISS allows you to export data from one server (source) and import it on another (target) server. For more information, see Administration › Iss_intro.

2.3.1. Hub Requirements

To set up a Hub installation, you require:

- One central Uyuni Server, which acts as the Hub Server.
- One or more additional Uyuni Servers, registered to the Hub. This document refers to these additional servers as peripheral servers.
- Any number of clients registered to the peripheral servers.
- Ensure the Hub Server and all peripheral servers are running Uyuni 5.0 or higher.

2.3.1.1. Hub Server

Deploy the central Uyuni Server, which acts as the Hub Server, with `mgradm`. For more information about deploying with `mgradm`, see Installation-and-upgrade › Container-deployment.

The Hub Server must not have clients registered to it. Clients should only be registered to the peripheral servers.
2.3.1.2. Peripheral Servers

Deploy the peripheral servers with `mgradm`. For more information about deploying with `mgradm`, see Installation-and-upgrade › Container-deployment.

Peripheral servers must be registered to the Hub Server as foreign clients. Standard registration methods cannot be used. Use `mgradm` on the peripheral server host:

```
mgradm hub register --api-server <HUB_FQDN> --api-user <HUB_ADMIN> --api-password <HUB_ADMIN_PASSWORD>
```

The peripheral servers must be registered to the Hub Server directly, do not use a proxy.

Optionally, it is possible to register the podman container host as a standard client (Salt minion) and manage it from Uyuni Server.

You need credentials to access the XMLRPC APIs on each server, including the Hub Server.

2.3.2. Hub Installation

Before you begin, you need to install the `hub-xmlrpc-api` package, and configure the Hub Server to use the API.

*Procedure: Installing and Configuring the Hub XMLRPC API*

1. On the Hub Server, or on a host that has access to all peripheral servers' XMLRPC APIs, install the `hub-xmlrpc-api` package. The package is available in the Uyuni 2024.05 repositories.

2. **OPTIONAL:** Set the Hub XMLRPC API service to start automatically at boot time, and start it immediately:

```
sudo systemctl enable hub-xmlrpc-api.service
sudo systemctl start hub-xmlrpc-api.service
```

3. **OPTIONAL:** Check that these parameters in the `/etc/hub/hub.conf` configuration file are correct:

   - **HUB_API_URL:** URL to the Hub Server XMLRPC API endpoint. Use the default value if you are installing `hub-xmlrpc-api` on the Hub Server.

   - **HUB_CONNECT_TIMEOUT:** the maximum number of seconds to wait for a response when connecting to a Server. Use the default value in most cases.

   - **HUB_REQUEST_TIMEOUT:** the maximum number of seconds to wait for a response when calling a Server method. Use the default value in most cases.

   - **HUB_CONNECT_USING_SSL:** use HTTPS instead of HTTP for communicating with peripheral Servers. Recommended for a secure environment.

4. Restart services to pick up configuration changes.
To use HTTPS to connect to peripheral Servers, you must set the HUB_CONNECT_USING_SSL parameter to true, and ensure that the SSL certificates for all the peripheral Servers are installed on the machine where the hub-xmlrpc-api service runs. Do this by copying the RHN-ORG-TRUSTED-SSL-CERT certificate file from each peripheral Server's http://<server-url>/pub/ directory to /etc/pki/trust/anchors/, and run update-ca-certificates.

2.3.3. Using the Hub API

Make sure the hub-xmlrpc-api service is started:

```bash
systemctl start hub-xmlrpc-api
```

Once it is running, connect to the service at port 2830 using any XMLRPC-compliant client libraries.

For examples, see Specialized-guides › Large-deployments.

Logs are saved in /var/log/hub/hub-xmlrpc-api.log. Logs are rotated weekly, or when the log file size reaches the specified limit. By default, the log file size limit is 10 MB.

2.3.4. Hub XMLRPC API Namespaces

The Hub XMLRPC API operates in a similar way to the Uyuni API. For Uyuni API documentation, see https://documentation.suse.com/suma.

The Hub XMLRPC API exposes the same methods that are available from the server’s XMLRPC API, with a few differences in parameter and return types. Additionally, the Hub XMLRPC API supports some Hub-specific end points which are not available in the Uyuni API.

The Hub XMLRPC API supports three different namespaces:

- The hub namespace is used to target the Hub XMLRPC API Server. It supports Hub-specific XMLRPC endpoints which are primarily related to authentication.

- The unicast namespace is used to target a single server registered in the hub. It redirects any call transparently to one specific server and returns any value as if the server's XMLRPC API endpoint was used directly.

- The multicast namespace is used to target multiple peripheral servers registered in the hub. It redirects any call transparently to all the specified servers and returns the results in the form of a map.

- If you do not specify a namespace, all calls are transparently redirected to the underlying Uyuni Server XMLRPC API of the Hub Server. This allows you to call all available methods on the Uyuni Server XMLRPC API.

Methods called without specifying any of the above namespaces will be forwarded to the normal
XMLRPC API of the hub. This is the API exposed on ports 80 and 443.

Some important considerations for hub namespaces:

- Individual server IDs can be obtained using `client.hub.listServerIds(hubSessionKey)`.

- The unicast namespace assumes all methods receive `hubSessionKey` and `serverID` as their first two parameters, then any other parameter as specified by the regular Server API.

  ```
  client.unicast.[namespace].[methodName](hubSessionKey, serverId, param1, param2)
  ```

- The `hubSessionKey` can be obtained using different authentication methods. For more information, see `Specialized-guides › Large-deployments`.

- The multicast namespace assumes all methods receive `hubSessionKey`, a list of `ServerID` values, then lists of per-server parameters as specified by the regular server XMLRPC API. The return value is a map, with `Successful` and `Failed` entries for each server involved in the call.

  ```
  client.multicast.[namespace].[methodname](hubSessionKey, [serverId1, serverId2], [param1_s1, param1_s2], [param2_s1, param2_s2])
  ```

### 2.3.5. Hub XMLRPC API Authentication Modes

The Hub XMLRPC API supports three different authentication modes:

- Manual mode (default): API credentials must be explicitly provided for each server.

- Relay mode: the credentials used to authenticate with the Hub are also used to authenticate to each server. You must provide a list of servers to connect to.

- Auto-connect mode: credentials are reused for each server, and any peripheral server you have access to is automatically connected.

#### 2.3.5.1. Authentication Examples

This section provides examples of each authentication method.

*Example: Manual Authentication*

In manual mode, credentials have to be explicitly provided for each peripheral server before you can connect to it.

A typical workflow for manual authentication is:

1. Credentials for the Hub are passed to the `login` method, and a session key for the Hub is returned (`hubSessionKey`).
2. Using the session key from the previous step, Uyuni Server IDs are obtained for all the peripheral servers attached to the Hub via the `hub.listServerIds` method.

3. Credentials for each peripheral server are provided to the `attachToServers` method. This performs authentication against each server’s XMLRPC API endpoint.

4. A multicast call is performed on a set of servers. This is defined by `serverIds`, which contains the IDs of the servers to target. In the background, `system.list_system` is called on each server’s XMLRPC API.

5. Hub aggregates the results and returns the response in the form of a `map`. The map has two entries:
   - **Successful**: list of responses for those peripheral servers where the call succeeded.
   - **Failed**: list of responses for those peripheral servers where the call failed.

If you want to call a method on just one Uyuni Server, then Hub API also provides a unicast namespace. In this case, the response will be a single value and not a map, in the same way as if you called that Uyuni server’s API directly.

Listing 8. Example Python Script for Manual Authentication:

```python
#!/usr/bin/python3
import xmlrpc.client

HUB_XMLRPC_API_URL = "<HUB_XMLRPC_API_URL>"
HUB_USERNAME = "<USERNAME>"
HUB_PASSWORD = "<PASSWORD>"
client = xmlrpc.client.ServerProxy(HUB_XMLRPC_API_URL, verbose=0)

hubSessionKey = client.hub.login(HUB_USERNAME, HUB_PASSWORD)

# Get the server IDs
serverIds = client.hub.listServerIds(hubSessionKey)

# For simplicity, this example assumes you are using the same username and password here, as on the hub server.
# However, in most cases, every server has its own individual credentials.
usernames = [HUB_USERNAME for s in serverIds]
passwords = [HUB_PASSWORD for s in serverIds]

# Each server uses the credentials set above, `client.hub.attachToServers` needs
# them passed as lists with as many elements as there are servers.
client.hub.attachToServers(hubSessionKey, serverIds, usernames, passwords)

# Perform the operation
systemsPerServer = client.multicast.system.list_systems(hubSessionKey, serverIds)
successfulResponses = systemsPerServer["Successful"]["Responses"]
failedResponses = systemsPerServer["Failed"]["Responses"]

for system in successfulResponses:
    print(system)

# Log out
client.hub.logout(hubSessionKey)
```

Example: Relay Authentication
In relay authentication mode, the credentials used to sign in to the Hub API are also used to sign in into the APIs of the peripheral servers the user wants to work with. In this authentication mode, it is assumed that the same credentials are valid for every server, and that they correspond to a user with appropriate permissions.

After signing in, you must call the attachToServers method. This method defines the servers to target in all subsequent calls.

A typical workflow for relay authentication is:

1. Credentials for the Hub are passed to the loginWithAuthRelayMode method, and a session key for the Hub is returned (hubSessionKey).
2. Using the session key from the previous step, Uyuni Server IDs are obtained for all the peripheral servers attached to the Hub via the hub.listServerIds method.
3. A call to attachToServers is made, and the same credentials used to sign in to the Hub are passed to each server. This performs authentication against each server’s XMLRPC API endpoint.
4. A multicast call is performed on a set of servers. This is defined by serverIds, which contains the IDs of the servers to target. In the background, system.list_system is called on each server’s XMLRPC API.
5. Hub aggregates the results and returns the response in the form of a map. The map has two entries:
   - Successful: list of responses for those peripheral servers where the call succeeded.
   - Failed: list of responses for those peripheral servers the call failed.
Listing 9. Example Python Script for Relay Authentication:

```python
#!/usr/bin/python3
import xmlrpc.client
HUB_XMLRPC_API_URL = "<HUB_XMLRPC_API_URL>"
HUB_USERNAME = "<USERNAME>"
HUB_PASSWORD = "<PASSWORD>"
client = xmlrpc.client.ServerProxy(HUB_XMLRPC_API_URL, verbose=0)

hubSessionKey = client.hub.loginWithAuthRelayMode(HUB_USERNAME, HUB_PASSWORD)
# Get the server IDs
serverIds = client.hub.listServerIds(hubSessionKey)

# Authenticate those servers (same credentials will be used as of hub to authenticate)
client.hub.attachToServers(hubSessionKey, serverIds)

# Perform the needed operation
systemsPerServer = client.multicast.system.list_systems(hubSessionKey, serverIds)
successfulResponses = systemsPerServer["Successful"]["Responses"]
failedResponses = systemsPerServer["Failed"]["Responses"]

for system in successfulResponses:
    print(system)

# Log out
client.hub.logout(hubSessionKey)
```

Example: Auto-Connect Authentication

Auto-connect mode is similar to relay mode, it uses the Hub credentials to sign in into all peripheral servers. However, there is no need to use the `attachToServers` method, as auto-connect mode connects to all available peripheral servers. This occurs at the same time as you sign in to the Hub.

A typical workflow for auto-connect authentication is:

1. Credentials for the Hub are passed to the `loginWithAutoconnectMode` method, and a session key for the Hub is returned (`hubSessionKey`).
2. A multicast call is performed on a set of servers. This is defined by `serverIds`, which contains the IDs of the servers to target. In the background, `system.list_system` is called on each server’s XMLRPC API.
3. Hub aggregates the results and returns the response in the form of a map. The map has two entries:
   - `Successful`: list of responses for those peripheral servers where the call succeeded.
   - `Failed`: list of responses for those peripheral servers where the call failed.
Listing 10. Example Python Script for Auto-Connect Authentication:

```python
#!/usr/bin/python3
import xmlrpc.client
HUB_XMLRPC_API_URL = "<HUB_XMLRPC_API_URL>"
HUB_USERNAME = "<USERNAME>"
HUB_PASSWORD = "<PASSWORD>"
client = xmlrpc.client.ServerProxy(HUB_XMLRPC_API_URL, verbose=0)
loginResponse = client.hub.loginWithAutoconnectMode(HUB_USERNAME, HUB_PASSWORD)
hubSessionKey = loginResponse["SessionKey"]

# Get the server IDs
serverIds = client.hub.listServerIds(hubSessionKey)

# Perform the needed operation
systemsPerServer = client.multicast.system.list_systems(hubSessionKey, serverIds)
successfulResponses = systemsPerServer["Successful"]["Responses"]
failedResponses = systemsPerServer["Failed"]["Responses"]

for system in successfulResponses:
    print(system)

# Log out
client.hub.logout(hubSessionKey)
```

2.3.6. Hub Reporting

The Hub prepares and provides content for multiple peripheral Uyuni Servers. The goal of the reporting feature is to get data from these Servers back and have combined reporting data available on the Hub. The data is made available for external Reporting Tools.

2.3.6.1. Architecture

The main database is a PostgreSQL database in the Uyuni Hub system. It stores all the information collected from all the servers, and aggregates them. Every peripheral Server has its own reporting database where the information is collected for that system. In summary:

- the DB in Uyuni Hub stores, collects and aggregates data coming from all the DBs of the peripheral Servers,
- the DB in Uyuni Hub stores also its own data from the systems directly connected and managed by the Hub,
- the DB in peripheral Uyuni Server stores its own data,
- the reporting tool can be connected either to the Hub or to any Uyuni Server.

2.3.6.2. Setup

The reporting database and schema are set up by default using the local PostgreSQL server. The reporting database is a separate database accessible via the network.
As a requirement we expect all server certificates are signed by the same Root Certificate Authority (CA). The whole Uyuni Hub environment should be using only one Root CA.

### 2.3.6.2.1. Create a DB user for the reporting

Before connecting an external Reporting Tools to the Database, a user with read-only permission should be created. For doing that, it is possible to use `uyuni-setup-reportdb-user`.

**usage:** `uyuni-setup-reportdb-user [options]`

**options:**
- `--help`  
  show this help message and exit
- `--non-interactive`  
  Switches to non-interactive mode
- `--dbuser=DBUSER`  
  Report DB User
- `--dbpassword=DBPASSWORD`  
  Report DB Password
- `--add`  
  Add the new user
- `--delete`  
  Delete the user
- `--modify`  
  Set a new password

### 2.3.6.3. Database Schema

The schema exports the most important tables from the main Uyuni Database as a de-normalized variant containing only data which are relevant for a report.

Ready-to-use reports are provided as views, aggregating data over multiple tables.

Every table gets an extra id column (`mgm_id`) specifying the Uyuni server which provided the data. On a single Uyuni Server this column has the standard value 1 which represent localhost. On the Uyuni Hub it is replaced with the real server id the managed server has in the hub database.

Another common additional field is `synced_date`, which represents the time when the data were exported from the main Uyuni Server database.

Navigate to Help › Report Database Schema to find the schema documentation.

### 2.3.6.4. Data Generation and Update

Uyuni uses taskomatic jobs to generate the data for the reporting database and to get the data from the peripheral servers to the hub.

To generate and update the data on a peripheral server, the responsible schedule is called `update-reporting-default`. It is executed by default daily at midnight.

On the Uyuni Hub there is a second schedule which is important. To fetch the reporting data from all
registered peripheral servers, the task with the name `update-reporting-hub-default` is executed daily at 1:30 AM.

All times are in the local timezone of the server. If the peripheral servers are in different timezones, it is recommended to align all the schedules. Make sure that all reporting data are gathered at a specific point in time, and `update-reporting-hub-default` is running when all peripheral servers have actually finished their local jobs.

### 2.3.6.4.1. Tuning Reporting Jobs

#### 2.3.6.4.2. report_db_batch_size

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of rows fetched and written from one database to the other in one shot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Out of memory errors or on processing speed problems.</td>
</tr>
<tr>
<td>Value default</td>
<td>2000</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>500 - 5000</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>report_db_batch_size = 4000</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage. Monitor memory usage closely before and after the change.</td>
</tr>
</tbody>
</table>

#### 2.3.6.4.3. report_db_hub_workers

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of workers requesting data from peripheral servers on a hub at the same point in time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>The number of peripheral servers increases significantly (more than 100), or a faster synchronization is required.</td>
</tr>
<tr>
<td>Value default</td>
<td>2</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>2 - 10</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>report_db_hub_workers = 5</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage. Monitor memory and cpu usage of the hub closely before and after the change. Also monitor the load, cpu and memory usage of the reporting database of the hub.</td>
</tr>
</tbody>
</table>

2.4. Managing Large Scale Deployments in a Retail Environment

Uyuni for Retail 2024.05 is an open source infrastructure management solution, optimized and tailored specifically for the retail industry. It uses the same technology as SUSE Manager, but is customized to address the needs of retail organizations.

Uyuni for Retail is designed for use in retail situations where customers can use point-of-service terminals to purchase or exchange goods, take part in promotions, or collect loyalty points. In addition to retail installations, it can also be used for novel purposes, such as maintaining student computers in an educational environment, or self-service kiosks in banks or hospitals.

Uyuni for Retail is intended for use in installations that include servers, workstations, point-of-service terminals, and other devices. It allows administrators to install, configure, and update the software on their servers, and manage the deployment and provisioning of point-of-service machines.

Point-of-Service (POS) terminals can come in many different formats, such as point-of-sale terminals, kiosks, digital scales, self-service systems, and reverse-vending systems. Every terminal, however, is provided by a vendor, who set basic information about the device in the firmware. Uyuni for Retail accesses this vendor information to determine how best to work with the terminal in use.

In most cases, different terminals will require a different operating system (OS) image to ensure they work correctly. For example, an information kiosk has a high-resolution touchscreen, where a cashier terminal might only have a very basic display. While both of these terminals require similar processing and network functionality, they will require different OS images. The OS images ensure that the different display mechanisms work correctly.

For more information about setting up and using Uyuni for Retail, see Retail › Retail-overview.

2.5. Tuning Large Scale Deployments

Uyuni is designed by default to work on small and medium scale installations. For installations with more than 1000 clients per Uyuni Server, adequate hardware sizing and parameter tuning must be performed.

The instructions in this section can have severe and catastrophic performance impacts when improperly used. In some cases, they can cause Uyuni to completely cease functioning. Always test changes before implementing them in a production environment. During implementation, take care when changing parameters. Monitor performance before and after each change, and revert any steps that do not produce the expected result.
Tuning is not required on installations of fewer than 1000 clients. Do not perform these instructions on small or medium scale installations.

2.5.1. The Tuning Process

Any Uyuni installation is subject to a number of design and infrastructure constraints that, for the purposes of tuning, we call environmental variables. Environmental variables can include the total number of clients, the number of different operating systems under management, and the number of software channels.

Environmental variables influence, either directly or indirectly, the value of most configuration parameters. During the tuning process, the configuration parameters are manipulated to improve system performance.

Before you begin tuning, you will need to estimate the best setting for each environment variable, and adjust the configuration parameters to suit.

To help you with the estimation process, we have provided you with a dependency graph. Locate the environmental variables on the dependency graph to determine how they will influence other variables and parameters.

Environmental variables are represented by graph nodes in a rectangle at the top of the dependency graph. Each node is connected to the relevant parameters that might need tuning. Consult the relevant sections in this document for more information about recommended values.

Tuning one parameter might require tuning other parameters, or changing hardware, or the infrastructure. When you change a parameter, follow the arrows from that node on the graph to determine what other parameters might need adjustment. Continue through each parameter until you have visited all nodes on the graph.

Key to the Dependency Graph
• 3D boxes are hardware design variables or constraints
• Oval-shaped boxes are software or system design variables or constraints
• Rectangle-shaped boxes are configurable parameters, color-coded by configuration file:
  ◦ Red: Apache httpd configuration files
  ◦ Blue: Salt configuration files
  ◦ Brown: Tomcat configuration files
  ◦ Grey: PostgreSQL configuration files
  ◦ Purple: /etc/rhn/rhn.conf
• Dashed connecting lines indicate a variable or constraint that might require a change to another parameter
• Solid connecting lines indicate that changing a configuration parameter requires checking another one to prevent issues

After the initial tuning has been completed, you will need to consider tuning again in these cases:

• If your tuning inputs change significantly
• If special conditions arise that require a certain parameter to be changed. For example, if specific warnings appear in a log file.
• If performance is not satisfactory

To re-tune your installation, you will need to use the dependency graph again. Start from the node where significant change has happened.

2.5.2. Environmental Variables

This section contains information about environmental variables (inputs to the tuning process).

Network Bandwidth
A measure of the typically available egress bandwidth from the Uyuni Server host to the clients or Uyuni Proxy hosts. This should take into account network hardware and topology as well as possible capacity limits on switches, routers, and other network equipment between the server and clients.

Channel count
The number of expected channels to manage. Includes any vendor-provided, third-party, and cloned or staged channels.

Client count
The total number of actual or expected clients. It is important to tune any parameters in advance of a client count increase, whenever possible.

OS mix
The number of distinct operating system versions that managed clients have installed. This is ordered
by family (SUSE Linux Enterprise, openSUSE, Red Hat Enterprise Linux, or Ubuntu based). Storage and computing requirements are different in each case.

**User count**

The expected maximum amount of concurrent users interacting with the Web UI plus the number of programs simultaneously using the XMLRPC API. Includes `spacecmd`, `spacewalk-clone-by-date`, and similar.

### 2.5.3. Parameters

This section contains information about the available parameters.

#### 2.5.3.1. MaxClients

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of HTTP requests served simultaneously by Apache httpd. Proxies, Web UI, and XMLRPC API clients each consume one. Requests exceeding the parameter will be queued and might result in timeouts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>User count and proxy count increase significantly and this line appears in /var/log/apache2/error_log: […] [mpm_prefork:error] [pid …] AH00161: server reached MaxRequestWorkers setting, consider raising the MaxRequestWorkers setting.</td>
</tr>
<tr>
<td>Value default</td>
<td>150</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>150-500</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/apache2/server-tuning.conf, in the prefork.c section</td>
</tr>
<tr>
<td>Example</td>
<td>MaxClients = 200</td>
</tr>
<tr>
<td>After changing</td>
<td>Immediately change <code>ServerLimit</code> and check <code>maxThreads</code> for possible adjustment.</td>
</tr>
<tr>
<td>Notes</td>
<td>This parameter was renamed to MaxRequestWorkers, both names are valid.</td>
</tr>
</tbody>
</table>
### 2.5.3.2. ServerLimit

<table>
<thead>
<tr>
<th>Description</th>
<th>The number of Apache httpd processes serving HTTP requests simultaneously. The number must equal MaxClients.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>MaxClients changes</td>
</tr>
<tr>
<td>Value default</td>
<td>150</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>The same value as MaxClients</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/apache2/server-tuning.conf, in the prefork.c section</td>
</tr>
<tr>
<td>Example</td>
<td>ServerLimit = 200</td>
</tr>
</tbody>
</table>

### 2.5.3.3. maxThreads

<table>
<thead>
<tr>
<th>Description</th>
<th>The number of Tomcat threads dedicated to serving HTTP requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>MaxClients changes. maxThreads must always be equal or greater than MaxClients</td>
</tr>
<tr>
<td>Value default</td>
<td>150</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>The same value as MaxClients</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/tomcat/server.xml</td>
</tr>
<tr>
<td>Example</td>
<td><code>&lt;Connector port=&quot;8009&quot; protocol=&quot;AJP/1.3&quot; redirectPort=&quot;8443&quot; URIEncoding=&quot;UTF-8&quot; address=&quot;127.0.0.1&quot; maxThreads=&quot;200&quot; connectionTimeout=&quot;20000&quot;/&gt;</code></td>
</tr>
</tbody>
</table>

### 2.5.3.4. connectionTimeout

| Description | The number of milliseconds before a non-responding AJP connection is forcibly closed.                  |

---

99 / 128
### keepAliveTimeout

<table>
<thead>
<tr>
<th>Description</th>
<th>The number of milliseconds without data exchange from the JVM before a non-responding AJP connection is forcibly closed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Client count increases significantly and AH00992, AH00877, and AH01030 errors appear in Apache error logs during a load peak.</td>
</tr>
<tr>
<td>Value default</td>
<td>300000</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>20000-600000</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/tomcat/server.xml</td>
</tr>
</tbody>
</table>
| Example | `<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" URIEncoding="UTF-8" address="127.0.0.1" maxThreads="200" connectionTimeout="1000000" keepAliveTimeout="300000"/>`

### 2.5.3.6. Tomcat’s `-Xmx`

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum amount of memory Tomcat can use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><code>java.message_queue_thread_pool_size</code> is increased or <code>OutOfMemoryException</code> errors appear in <code>/var/log/rhn/rhn_web_ui.log</code></td>
</tr>
<tr>
<td>Value default</td>
<td>1 GiB</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>4-8 GiB</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/tomcat/conf.d/tomcat_java_options.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>JAVA_OPTS=&quot;... -Xmx8G ...&quot;</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage</td>
</tr>
<tr>
<td>More information</td>
<td><a href="https://docs.oracle.com/javase/8/docs/technote/tools/windows/java.html">https://docs.oracle.com/javase/8/docs/technote/tools/windows/java.html</a></td>
</tr>
</tbody>
</table>

### 2.5.3.7. `java.disable_list_update_status`

<table>
<thead>
<tr>
<th>Description</th>
<th>Disable displaying the update status for clients of a system group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>displaying the update status causes timeouts</td>
</tr>
<tr>
<td>Value default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Value recommendation</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>java.disable_list_update_status = true</code></td>
</tr>
<tr>
<td>After changing</td>
<td>?</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>More information</td>
<td><code>man rhn.conf</code></td>
</tr>
</tbody>
</table>

### 2.5.3.8. `java.message_queue_thread_pool_size`

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of threads in Tomcat dedicated to asynchronous operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><code>Client count</code> increases significantly</td>
</tr>
</tbody>
</table>

---

2.5. Tuning Large Scale Deployments | Uyuni 2024.05
### 2.5.3.9. `java.salt_batch_size`

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of minions concurrently executing a scheduled action.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Client count reaches several thousands and actions are not executed quickly enough.</td>
</tr>
<tr>
<td>Value default</td>
<td>200</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>200-500</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/rhn/rhn.conf</td>
</tr>
<tr>
<td>Example</td>
<td><code>java.salt_batch_size = 300</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage. Monitor memory usage closely before and after the change.</td>
</tr>
<tr>
<td>More information</td>
<td>Specialized-guides &gt; Salt</td>
</tr>
</tbody>
</table>

### 2.5.3.10. `java.salt_event_thread_pool_size`

<p>| Value default | 5 |
| Value recommendation | 50 - 150 |
| Location     | /etc/rhn/rhn.conf |
| Example      | <code>java.message_queue_thread_pool_size = 50</code> |
| After changing | Check <code>hibernate.c3p0.max_size</code>, as each thread consumes a PostgreSQL connection, starvation might happen if the allocated connection pool is insufficient. Check <code>thread_pool</code>, as each thread might perform Salt API calls, starvation might happen if the allocated Salt thread pool is insufficient. Check Tomcat's <code>-Xmx</code>, as each thread consumes memory, OutOfMemoryException might be raised if insufficient. |
| Notes         | Incoming Salt events are handled in separate thread pool, see <code>java.salt_event_thread_pool_size</code> |
| More information | man rhn.conf |</p>
<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of threads in Tomcat dedicated to handling of incoming Salt events.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>The number of queued Salt events grows. Typically, this can happen during onboarding of large number of minions with higher value of <code>java.salt_presence_ping_timeout</code>. The number of events can be queried by `echo &quot;select count(*) from susesaltevent;&quot;</td>
</tr>
<tr>
<td>Value default</td>
<td>8</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>20-100</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>java.salt_event_thread_pool_size = 50</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check the length of Salt event queue. Check <code>hibernate.c3p0.max_size</code>, as each thread consumes a PostgreSQL connection, starvation might happen if the allocated connection pool is insufficient. Check <code>thread_pool</code>, as each thread might perform Salt API calls, starvation might happen if the allocated Salt thread pool is insufficient. Check Tomcat's <code>-Xmx</code>, as each thread consumes memory, <code>OutOfMemoryException</code> might be raised if insufficient.</td>
</tr>
<tr>
<td>More information</td>
<td><code>man rhn.conf</code></td>
</tr>
</tbody>
</table>

### 2.5.3.11. `java.salt_presence_ping_timeout`

<table>
<thead>
<tr>
<th>Description</th>
<th>Before any action is executed on a client, a presence ping is executed to make sure the client is reachable. This parameter sets the amount of time before a second command (<code>find_job</code>) is sent to the client to verify its presence. Having many clients typically means some will respond faster than others, so this timeout could be raised to accommodate for the slower ones.</th>
</tr>
</thead>
</table>
Tune when Client count increases significantly, or some clients are responding correctly but too slowly, and Uyuni excludes them from calls. This line appears in /var/log/rhn/rhn_web_ui.log: "Got no result for <COMMAND> on minion <MINION_ID> (minion did not respond in time)"

Value default 4 seconds
Value recommendation 4-400 seconds
Location /etc/rhn/rhn.conf
Example java.salt_presence_ping_timeout = 40

After changing Large java.salt_presence_ping_timeout value can reduce overall throughput. This can be compensated by increasing java.salt_event_thread_pool_size

More information Specialized-guides › Salt

2.5.3.12. java.salt_presence_ping_gather_job_timeout

Description Before any action is executed on a client, a presence ping is executed to make sure the client is reachable. After java.salt_presence_ping_timeout seconds have elapsed without a response, a second command (find_job) is sent to the client for a final check. This parameter sets the number of seconds after the second command after which the client is definitely considered offline. Having many clients typically means some will respond faster than others, so this timeout could be raised to accommodate for the slower ones.

Tune when Client count increases significantly, or some clients are responding correctly but too slowly, and Uyuni excludes them from calls. This line appears in /var/log/rhn/rhn_web_ui.log: "Got no result for <COMMAND> on minion <MINION_ID> (minion did not respond in time)"
### 2.5.3.13. java.taskomatic_channel_repodata_workers

#### Description
Whenever content is changed in a software channel, its metadata needs to be recomputed before clients can use it. Channel-altering operations include the addition of a patch, the removal of a package or a repository synchronization run. This parameter specifies the maximum number of Taskomatic threads that Uyuni will use to recompute the channel metadata. Channel metadata computation is both CPU-bound and memory-heavy, so raising this parameter and operating on many channels simultaneously could cause Taskomatic to consume significant resources, but channels will be available to clients sooner.

#### Tune when
*Channel count* increases significantly (more than 50), or more concurrent operations on channels are expected.

#### Value default
2

#### Value recommendation
2-10

#### Location
/etc/rhn/rhn.conf

#### Example
```java
java.taskomatic_channel_repodata_workers = 4
```

#### After changing
Check *taskomatic.java.maxmemory* for adjustment, as every new thread will consume memory

#### More information
man rhn.conf

---

### 2.5.3.14. taskomatic.java.maxmemory

After changing the value of this parameter, check *taskomatic.java.maxmemory* to ensure it is set appropriately for your environment.
### Description
The maximum amount of memory Taskomatic can use. Generation of metadata, especially for some OSs, can be memory-intensive, so this parameter might need raising depending on the managed OS mix.

### Tune when
`java.taskomatic_channel_repodata_workers` increases, OSs are added to Uyuni (particularly Red Hat Enterprise Linux or Ubuntu), or `OutOfMemoryException` errors appear in `/var/log/rhn/rhn_taskomatic_daemon.log`.

### Value default
4096 MiB

### Value recommendation
4096-16384 MiB

### Location
`/etc/rhn/rhn.conf`

### Example
```
taskomatic.java.maxmemory = 8192
```

### After changing
Check memory usage.

### More information
`man rhn.conf`

### 2.5.3.15. `org.quartz.threadPool.threadCount`

### Description
The number of Taskomatic worker threads. Increasing this value allows Taskomatic to serve more clients in parallel.

### Tune when
`Client count` increases significantly

### Value default
20

### Value recommendation
20-200

### Location
`/etc/rhn/rhn.conf`

### Example
```
org.quartz.threadPool.threadCount = 100
```

### After changing
Check `hibernate.c3p0.max_size` and `thread_pool` for adjustment

### More information
[http://www.quartz-scheduler.org/documentation/2.4.0-SNAPSHOT/configuration.html](http://www.quartz-scheduler.org/documentation/2.4.0-SNAPSHOT/configuration.html)
### 2.5.3.16. `org.quartz.scheduler.idleWaitTime`

<table>
<thead>
<tr>
<th>Description</th>
<th>Cycle time for Taskomatic. Decreasing this value lowers the latency of Taskomatic.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><strong>Client count</strong> is in the thousands.</td>
</tr>
<tr>
<td>Value default</td>
<td>5000 ms</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>1000-5000 ms</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>org.quartz.scheduler.idleWaitTime = 1000</code></td>
</tr>
<tr>
<td>More information</td>
<td><a href="http://www.quartz-scheduler.org/documentation/2.4.0-SNAPSHOT/configuration.html">http://www.quartz-scheduler.org/documentation/2.4.0-SNAPSHOT/configuration.html</a></td>
</tr>
</tbody>
</table>

### 2.5.3.17. `MinionActionExecutor.parallel_threads`

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of Taskomatic threads dedicated to sending commands to Salt clients as a result of actions being executed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><strong>Client count</strong> is in the thousands.</td>
</tr>
<tr>
<td>Value default</td>
<td>1</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>1-10</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>taskomatic.com.redhat.rhn.taskomatic.task.MinionActionExecutor.parallel_threads = 10</code></td>
</tr>
</tbody>
</table>

### 2.5.3.18. `SSHMinionActionExecutor.parallel_threads`

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of Taskomatic threads dedicated to sending commands to Salt SSH clients as a result of actions being executed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><strong>Client count</strong> is in the hundreds.</td>
</tr>
<tr>
<td>Value default</td>
<td>20</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>20-100</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/rhn/rhn.conf</code></td>
</tr>
</tbody>
</table>
Example | taskomatic.com.redhat.rhn.taskomatic.task.SSHMinionActionExecutor.parallel_threads = 40

### 2.5.3.19. hibernate.c3p0.max_size

**Description**
Maximum number of PostgreSQL connections simultaneously available to both Tomcat and Taskomatic. If any of those components requires more concurrent connections, their requests will be queued.

**Tune when**
- `java.message_queue_thread_pool_size` or `maxThreads` increase significantly, or when `org.quartz.threadPool.threadCount` has changed significantly. Each thread consumes one connection in Taskomatic and Tomcat, having more threads than connections might result in starving.

**Value default**
20

**Value recommendation**
100 to 200, higher than the maximum of `java.message_queue_thread_pool_size + maxThreads` and `org.quartz.threadPool.threadCount`

**Location**
/`etc/rhn/rhn.conf`

**Example**
hibernate.c3p0.max_size = 100

**After changing**
Check `max_connections` for adjustment.

**More information**
https://www.mchange.com/projects/c3p0/

### 2.5.3.20. rhn-search.java.maxmemory

**Description**
The maximum amount of memory that the rhn-search service can use.

**Tune when**
- `Client count` increases significantly, and `OutOfMemoryException` errors appear in `journalctl -u rhn-search`.

**Value default**
512 MiB

**Value recommendation**
512-4096 MiB
2.5.3.21. shared_buffers

<table>
<thead>
<tr>
<th>Description</th>
<th>The amount of memory reserved for PostgreSQL shared buffers, which contain caches of database tables and index data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>RAM changes</td>
</tr>
<tr>
<td>Value default</td>
<td>25% of total RAM</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>25-40% of total RAM</td>
</tr>
<tr>
<td>Location</td>
<td>/var/lib/pgsql/data/postgresql.conf</td>
</tr>
<tr>
<td>Example</td>
<td>shared_buffers = 8192MB</td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage.</td>
</tr>
<tr>
<td>More information</td>
<td><a href="https://www.postgresql.org/docs/15/runtime-config-resource.html#GUC-SHARED-BUFFERS">https://www.postgresql.org/docs/15/runtime-config-resource.html#GUC-SHARED-BUFFERS</a></td>
</tr>
</tbody>
</table>

2.5.3.22. max_connections

<table>
<thead>
<tr>
<th>Description</th>
<th>Maximum number of PostgreSQL connections available to applications. More connections allow for more concurrent threads/workers in various components (in particular Tomcat and Taskomatic), which generally improves performance. However, each connection consumes resources, in particular work_mem megabytes per sort operation per connection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>hibernate.c3p0.max_size changes significantly, as that parameter determines the maximum number of connections available to Tomcat and Taskomatic</td>
</tr>
<tr>
<td>Value default</td>
<td>400</td>
</tr>
<tr>
<td><strong>Value recommendation</strong></td>
<td>Depends on other settings, use <code>/usr/lib/susemanager/bin/susemanager-connection-check</code> to obtain a recommendation.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td><code>/var/lib/pgsql/data/postgresql.conf</code></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><code>max_connections = 250</code></td>
</tr>
<tr>
<td><strong>After changing</strong></td>
<td>Check memory usage. Monitor memory usage closely before and after the change.</td>
</tr>
<tr>
<td><strong>More information</strong></td>
<td><a href="https://www.postgresql.org/docs/15/runtime-config-connection.html#GUC-MAX-CONNECTIONS">https://www.postgresql.org/docs/15/runtime-config-connection.html#GUC-MAX-CONNECTIONS</a></td>
</tr>
</tbody>
</table>

### 2.5.3.23. `work_mem`

**Description**
The amount of memory allocated by PostgreSQL every time a connection needs to do a sort or hash operation. Every connection (as specified by `max_connections`) might make use of an amount of memory equal to a multiple of `work_mem`.

**Tune when**
Database operations are slow because of excessive temporary file disk I/O. To test if that is happening, add `log_temp_files = 5120` to `/var/lib/pgsql/data/postgresql.conf`, restart PostgreSQL, and monitor the PostgreSQL log files. If you see lines containing `LOG: temporary file:`, try raising this parameter’s value to help reduce disk I/O and speed up database operations.

**Value recommendation**
`2-20 MB`

**Location**
`/var/lib/pgsql/data/postgresql.conf`

**Example**
`work_mem = 10MB`

**After changing**
check if the Uyuni Server might need additional RAM.

**More information**
[https://www.postgresql.org/docs/15/runtime-config-resource.html#GUC-WORK-MEM](https://www.postgresql.org/docs/15/runtime-config-resource.html#GUC-WORK-MEM)
### 2.5.3.24. effective_cache_size

<table>
<thead>
<tr>
<th>Description</th>
<th>Estimation of the total memory available to PostgreSQL for caching. It is the explicitly reserved memory (shared_buffers) plus any memory used by the kernel as cache/buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Hardware RAM or memory usage increase significantly</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>Start with 75% of total RAM. For finer settings, use shared_buffers + free memory + buffer/cache memory. Free and buffer/cache can be determined via the free -m command (free and buff/cache in the output respectively)</td>
</tr>
<tr>
<td>Location</td>
<td>/var/lib/pgsql/data/postgresql.conf</td>
</tr>
<tr>
<td>Example</td>
<td>effective_cache_size = 24GB</td>
</tr>
<tr>
<td>After changing</td>
<td>Check memory usage</td>
</tr>
<tr>
<td>Notes</td>
<td>This is an estimation for the query planner, not an allocation.</td>
</tr>
<tr>
<td>More information</td>
<td><a href="https://www.postgresql.org/docs/15/runtime-config-query.html#GUC-EFFECTIVE-CACHE-SIZE">https://www.postgresql.org/docs/15/runtime-config-query.html#GUC-EFFECTIVE-CACHE-SIZE</a></td>
</tr>
</tbody>
</table>

### 2.5.3.25. thread_pool

<table>
<thead>
<tr>
<th>Description</th>
<th>The number of worker threads serving Salt API HTTP requests. A higher number can improve parallelism of Uyuni Server-initiated Salt operations, but will consume more memory.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>java.message_queue_thread_pool_size or org.quartz.threadPool.threadCount are changed. Starvation can occur when there are more Tomcat or Taskomatic threads making simultaneous Salt API calls than there are Salt API worker threads.</td>
</tr>
<tr>
<td>Value default</td>
<td>100</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>100-500, but should be higher than the sum of <code>java.message_queue_thread_pool_size</code> and <code>org.quartz.threadPool.threadCount</code></td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/salt/master.d/susemanager.conf</code>, in the <code>rest_cherrypy</code> section.</td>
</tr>
<tr>
<td>Example</td>
<td><code>thread_pool: 100</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check <code>worker_threads</code> for adjustment.</td>
</tr>
<tr>
<td>More information</td>
<td><a href="https://docs.saltstack.com/en/latest/ref/netapi/all/salt.netapi.rest_cherrypy.html#performance-tuning">https://docs.saltstack.com/en/latest/ref/netapi/all/salt.netapi.rest_cherrypy.html#performance-tuning</a></td>
</tr>
</tbody>
</table>

### 2.5.3.26. `worker_threads`

<table>
<thead>
<tr>
<th>Description</th>
<th>The number of salt-master worker threads that process commands and replies from minions and the Salt API. Increasing this value, assuming sufficient resources are available, allows Salt to process more data in parallel from minions without timing out, but will consume significantly more RAM (typically about 70 MiB per thread).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td><strong>Client count</strong> increases significantly, <strong>thread_pool</strong> increases significantly, or SaltReqTimeoutError or Message timed out errors appear in <code>/var/log/salt/master</code>.</td>
</tr>
<tr>
<td>Value default</td>
<td>8</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>8-200</td>
</tr>
<tr>
<td>Location</td>
<td><code>/etc/salt/master.d/tuning.conf</code></td>
</tr>
<tr>
<td>Example</td>
<td><code>worker_threads: 50</code></td>
</tr>
<tr>
<td>After changing</td>
<td>Check <code>memory usage</code>. Monitor memory usage closely before and after the change.</td>
</tr>
</tbody>
</table>

### 2.5.3.27. `pub_hwm`
<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of outstanding messages sent by salt-master. If more than this number of messages need to be sent concurrently, communication with clients slows down, potentially resulting in timeout errors during load peaks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Client count increases significantly and Salt request timed out. The master is not responding. errors appear when pinging minions during a load peak.</td>
</tr>
<tr>
<td>Value default</td>
<td>1000</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>10000-100000</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/salt/master.d/tuning.conf</td>
</tr>
<tr>
<td>Example</td>
<td>pub_hwm: 10000</td>
</tr>
</tbody>
</table>

### 2.5.3.28. zmq_backlog

<table>
<thead>
<tr>
<th>Description</th>
<th>The maximum number of allowed client connections that have started but not concluded the opening process. If more than this number of clients connects in a very short time frame, connections are dropped and clients experience a delay re-connecting.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>Client count increases significantly and very many clients reconnect in a short time frame, TCP connections to the salt-master process get dropped by the kernel.</td>
</tr>
<tr>
<td>Value default</td>
<td>1000</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>1000-5000</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/salt/master.d/tuning.conf</td>
</tr>
<tr>
<td>Example</td>
<td>zmq_backlog: 2000</td>
</tr>
</tbody>
</table>
2.5.3.29. swappiness

<table>
<thead>
<tr>
<th>Description</th>
<th>How aggressively the kernel moves unused data from memory to the swap partition. Setting a lower parameter typically reduces swap usage and results in better performance, especially when RAM memory is abundant.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tune when</td>
<td>RAM increases, or swap is used when RAM memory is sufficient.</td>
</tr>
<tr>
<td>Value default</td>
<td>60</td>
</tr>
<tr>
<td>Value recommendation</td>
<td>1-60. For 128 GB of RAM, 10 is expected to give good results.</td>
</tr>
<tr>
<td>Location</td>
<td>/etc/sysctl.conf</td>
</tr>
<tr>
<td>Example</td>
<td>vm.swappiness = 20</td>
</tr>
</tbody>
</table>

2.5.3.30. Memory Usage

Adjusting some of the parameters listed in this section can result in a higher amount of RAM being used by various components. It is important that the amount of hardware RAM is adequate after any significant change.

To determine how RAM is being used, you will need to check each process that consumes it.

**Operating system**

Stop all Uyuni services and inspect the output of `free -h`.

**Java-based components**

This includes Taskomatic, Tomcat, and `rhn-search`. These services support a configurable memory cap.

**The Uyuni Server**

Depends on many factors and can only be estimated. Measure PostgreSQL reserved memory by checking `shared_buffers`, permanently. You can also multiply `work_mem` and `max_connections`, and multiply by three for a worst case estimate of per-query RAM. You will also need to check the operating system buffers and caches, which are used by PostgreSQL to host copies of database data. These often automatically occupy any available RAM.

It is important that the Uyuni Server has sufficient RAM to accommodate all of these processes, especially OS buffers and caches, to have reasonable PostgreSQL performance. We recommend you keep several gigabytes available at all times, and add more as the database size on disk increases.
Whenever the expected amount of memory available for OS buffers and caches changes, update the `effective_cache_size` parameter to have PostgreSQL use it correctly. You can calculate the total available by finding the total RAM available, less the expected memory usage.

To get a live breakdown of the memory used by services on the Uyuni Server, use this command:

```
pidstat -p ALL -r --human 1 60 | tee pidstat-memory.log
```

This command will save a copy of displayed data in the `pidstat-memory.log` file for later analysis.

### 2.6. Monitoring Large Scale Deployments

You can monitor your Uyuni environment using Prometheus and Grafana. Uyuni Server and Proxy are able to provide self-health metrics. You can also install and manage a number of Prometheus exporters on clients.

Prometheus and Grafana packages are included in the Uyuni Client Tools for SUSE Linux Enterprise 12, SUSE Linux Enterprise 15, CentOS 7, CentOS 8 and openSUSE 15.x.

You need to install Prometheus and Grafana on a machine separate from the Uyuni Server. We recommend you use a managed client as your monitoring server.

For more information on monitoring, see Administration › Monitoring.
Chapter 3. Quick Start: SAP Overview

Updated: 2024-04-29

This guide shows you how to use Uyuni to install and configure an SAP cluster. It guides you through setting up a single Uyuni Server, preparing your client systems, and configuring the cluster using formulas.

- For more information about SAP, see the SAP documentation at https://documentation.suse.com/sles-sap.
- For more information about Uyuni, see the Uyuni documentation at https://documentation.suse.com/suma.

3.1. Prepare Server

Before you start you need to install the Uyuni Server. The method for installing the Uyuni Server varies depending on your hardware and environment.

Uyuni is installed using the SUSE Linux Enterprise Server 15 Unified Installer. During the installation process, when you are prompted for which product to install, select Uyuni Server. The server does not need to have the SUSE Linux Enterprise Server 15 with SAP product installed. For more information about installing the Uyuni Server, see Installation-and-upgrade › Install-server-unified.

When the Uyuni Server is installed, set it up by running the yast2 susemanager_setup command from the command prompt. The setup script prompts you to complete additional details about your server, and give you the URL to use to access the Web UI. For more information about setup, see Installation-and-upgrade › Server-setup.

You need to do some configuration to set up the Uyuni Web UI. In your browser, navigate to the URL of the server, and configure your administration access to the Web UI. For more information about setting up the Web UI, see Installation-and-upgrade › Webui-setup.

Now you can use the Web UI to prepare software channels and activation keys for your clients.

On the Uyuni Server, add the appropriate SAP channels: From the Web UI, add SUSE Linux Enterprise Server 15 for SAP.

Synchronize the Uyuni Server with the SUSE Customer Center. You can do this using the Web UI. Add the new channel to your activation key.

To check if a channel has finished synchronizing navigate to Admin › Setup Wizard and select the Products tab. This dialog displays a completion bar for each product when they are being synchronized.
Software channels can be very large. The initial channel synchronization can sometimes take up to several hours.

When the initial synchronization is complete, we recommended you clone the channel before you work with it. This gives you a backup of the original synchronization data.

3.2. Preparing Clients

Your SAP cluster requires several client systems. Prepare your clients on physical or virtual hardware, and ensure you have SUSE Linux Enterprise Server 15 for SAP installation media ready. You cannot create an SAP cluster without the SUSE Linux Enterprise Server SAP extension, as it provides tooling specific to SAP.

One of the key features of SAP is high availability of the cluster. Every component within an SAP cluster has redundancy and failover protection. When you are preparing your clients, ensure you have enough hardware and infrastructure to allow for this. For more information about hardware requirements, see https://documentation.suse.com/sles-sap/15-SP4/html/SLES-SAP-installation/cha-plan.html#sec-hardware

For more information about the clients you need to set up for an SAP cluster, see https://documentation.suse.com/sbp/all.

3.2.1. Register Clients to the SUSE Customer Center

Each client within your SAP cluster must be registered with the SUSE Customer Center. To obtain your registration code, navigate to https://scc.suse.com/login in your web browser. Log in to your SCC account, or follow the prompts to create a new account. Click the [Subscriptions] tab to see the registration code. When you install SUSE Linux Enterprise Server 15 for SAP the Unified Installer prompts you for the code.

For more information about registering Uyuni with SUSE Customer Center, see Installation-and-upgrade › General-requirements.

3.2.2. Configure the Clients for Clustering

Every client system must have all the other client systems listed in their /etc/hosts file. Open the /etc/hosts file on each client, and add the hostname for each of the other clients.

3.2.3. Create a Shared Storage Device

Each of the clients needs to be able to access a shared disk. The shared disk can be physical hardware connected by ethernet, or you can set up a virtual disk and access it with iSCSI.

If you use a virtual disk, consider hosting it on a separate system. Do not use a client machine to host the shared storage disk.
3.2.4. Download the SAP Installation Software

Download the SAP installation media and save a copy on each client. The software that you require differs depending on your environment. For example, if you are using HANA, you need the SAP HANA platform. If you are using Netweaver, you need different packages. These software packages are provided by SAP, not by SUSE.

Ensure you have saved the installation software in the same file system location on each client. Alternatively, save it to a shared NFS drive.

3.2.5. Configure Clients to Use Latest module.run

Each client needs to be configured to use the latest version of module.run. On each of the client machines, open the /etc/salt/minion configuration file and add or edit this line:

```
use_superseded:
  - module.run
```

Restart the salt-minion process to enable the changes:

```
systemctl restart salt-minion
```

3.2.6. Install Additional Disks for HANA

For the clients that are going to run the HANA database, you require an additional storage device. This device is used to store files required by HANA, which are located in the /hana/ directory.

We recommend that this storage device be at least 20 GB. For some installations, you might require more, and it is possible to use multiple disks to provide this storage. For comprehensive hardware requirements, see https://documentation.suse.com/sbp/all.

3.2.7. Register Clients to the Server

First of all, make sure you have an activation key that is associated with the SLE-Product-SLES_SAP15 base channel. For more information about activation keys, see Client-configuration > Activation-keys.

In the Uyuni Web UI, navigate to Systems > Bootstrapping. Fill in the appropriate details, and make sure you check the Manage System Completely via SSH checkbox. In the Activation Key field, select the SLES for SAP activation key.

For more information about registering, see Client-configuration > Registration-webui.
3.3. Configure Clients

Uyuni uses formulas with forms to configure your SAP clients. There are two formulas that you need to use:

- **Hana** to configure the HANA database
- **Cluster** to configure the clients into a cluster

The formulas are provided by packages that you can download with your package manager. You need to install the formulas on the Uyuni Server. When you have installed the package, you can use the Uyuni Web UI to enable and configure the formulas. As you go through the formula configuration process, provide details of the clients that contain your SAP cluster, to set them up appropriately.

To install the formulas on the Uyuni Server, use your package manager to install these packages:

- saphanabootstrap-formula
- sapnwbootstrap-formula
- drbd-formula
- habootstrap-formula
- salt-shaptools

The order that you enable and configure the formulas is important. You must enable, configure, and apply the HANA formula first. Then you can enable, configure, and apply the cluster formula. If you perform these steps in the wrong order, your SAP installation fails.

### 3.3.1. Enable and Configure the HANA Formula

In the Uyuni Web UI, navigate to **Systems › System List** and click the client to use as the primary client in the cluster.

Navigate to the **Formulas** tab, locate the **Sap Hana Deployment** heading, and check the **Saphanabootstrap** formula in the list. Click **[Save]** and apply the highstate to activate the formula.

When the formula is activated, navigate to the **Formulas › Hana** tab, and complete the details in the form.

Make sure you check **Install required packages** to install everything you need on the client. In the **Nodes** sections, type the short hostname of the client to install the HANA database or the hostname you can retrieve on the command line with:

```
salt '<client-name>' grains.item host
```
3.3. Configure Clients

Provide further details for the installation.

Complete the remaining details according to your environment, click [Save], and apply the highstate. When the highstate is complete, you can go on to apply the cluster formula.

3.3.2. Enable and Configure the Cluster Formula

In the Uyuni Web UI, navigate to Systems > System List and click the client to use as the primary client in the cluster.

Navigate to the Formulas tab, locate the Cluster heading, and check the Habootstrap formula in the list. Click [Save] and apply the highstate to activate the formula.
When the formula is activated, navigate to the **Formulas > Cluster** tab, and complete the details in the form.

Make sure you check **Install required packages** to install everything you need on the client. Give your cluster a name, and specify the hostname of the primary client in the cluster.

Complete the remaining details according to your environment, click **[Save]**, and apply the highstate.
3.3. Configure Clients

On this page you can configure bootstrapping formulas to automatically install and configure software.

Saphanahost_bootstrap Formula

- **HANA**
  - Hostname: [blank]
  - HANA system identifier (SID): [blank]
  - HANA instance name: [blank]
  - SAP user password: [blank]
  - HANA network type: [blank]
  - Install HANA: [blank]

- **Nodes**
  - Hostname: [blank]
  - Hostname: [blank]

- **Install new HANA instance**
  - HANA software path: [blank]
  - Machine user: [blank]
  - Machine password: [blank]
  - Use configuration file: [blank]
  - SAP admin password (blank): [blank]
Chapter 4. GNU Free Documentation License

Copyright © 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a worldwide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections
then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History"). To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retile any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.
The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION
Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".