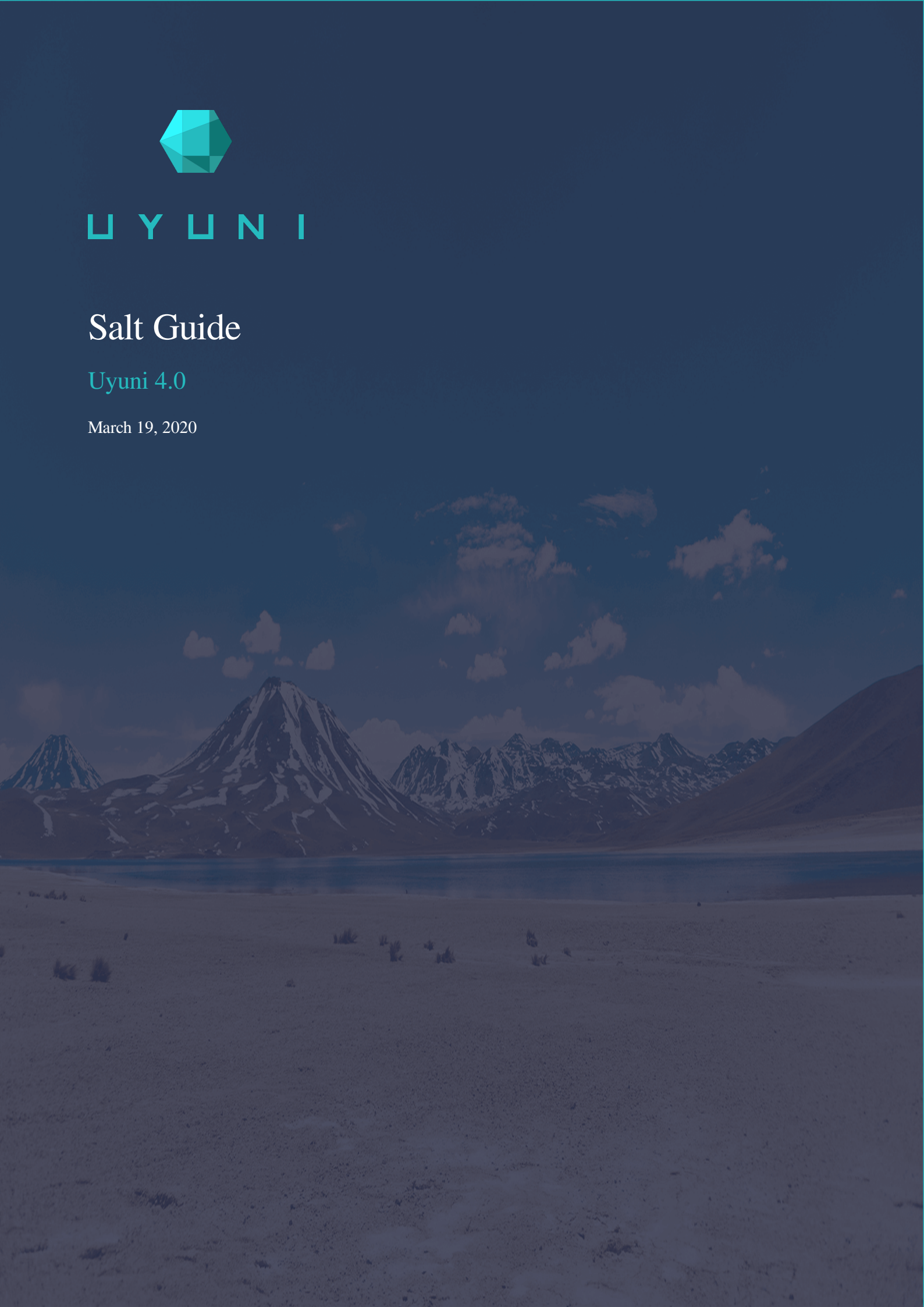


Salt Guide

Uyuni 4.0

March 19, 2020

Table of Contents

GNU Free Documentation License

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 world-wide, 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 retitle 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{ldquo}GNU
   Free Documentation License{rdquo}.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “ with…Texts.” line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
   Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Introduction

Publication Date: 2020-03-19

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 2019.2.0.

Throughout the Uyuni documentation, we use the term `Salt clients` to refer to Salt machines that are connected to and controlled by the Salt master on the Uyuni Server. This is to clearly differentiate them from traditional clients. In other documentation, and in some internal references, Salt clients are sometimes referred to as Salt `minions` instead. This is a difference in terminology only.

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 Salt 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.

Grain

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.

Salt modules are stored on clients and the Uyuni Server within the `/usr/lib/python*/site-packages/salt/` directory.

List all available grains with the `grains.ls` function:

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

You can also use `grains.items` to list collected grain system data:

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

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

Pillar

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.

State

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 written, and then applied to the target systems. This automates the process of bringing a large number of systems into a known state, and then maintaining them.

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

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.

The Salt Command

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

```
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.

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:

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

Target a specific client:

```
salt 'web1.example.com' test.ping
```

Glob Targeting

Target all clients using a particular domain:

```
salt '*example.com' test.ping
```

Target all clients using a particular label:

```
salt 'label*' test.ping
```

List Targeting

Specify a flat list of clients, using their IDs:

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

Regular Expression Targeting

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

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

IP Address Targeting

List available client IP addresses:

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

Target a specific client IP address:

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

Target all clients on a subnet:

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

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

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:

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

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

Salt Function Arguments

Functions accept arguments for any extra data.

For example, the `pkg.install` function requires an argument specifying which package to install:

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

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

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

Salt Useful Commands

This section contains the most used Salt commands.

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

salt-run

Display all clients that are running:

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

Display all clients that are not running:

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

Display the current status of all Salt clients:

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

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

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

salt-cp

Copy a file to a client or set of clients.

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

salt-key -l

List public keys:

```
salt-key -l
```

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 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. 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.

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

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)

Custom user-created states can be made with Uyuni. All user-created Salt state (SLS) files are saved on the Uyuni Server, in the `/srv/susemanager/salt/` directory. Within that directory, each organization has a sub-directory.

Listing 1. Example: SLS File Directory Structure

```
├──── manager_org_DEVEL
│    ├──── 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 ...
```

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. 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.

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

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
```

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
```

In these examples we will 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
```

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 %}
```

Salt File Locations and Structure

This diagram shows the Salt file structure, as it is used by the Uyuni Server. The files are listed in the `/etc/salt/master.d/susemanager.conf` configuration file.

```
# Configure different file roots. Custom salt states should only be placed in /srv/salt.
# Users should not touch other directories listed here.
file_roots:
  base:
    - /usr/share/susemanager/salt
    - /usr/share/salt-formulas/states
    - /usr/share/susemanager/formulas/states
    - /srv/susemanager/salt
    - /srv/salt

# Configure different file roots. Custom salt states should only be placed in /srv/salt.
# Users should not touch other directories listed here.
file_roots:
  base:
    - /usr/share/susemanager/salt
    - /usr/share/salt-formulas/states
    - /usr/share/susemanager/formulas/states
    - /srv/susemanager/salt
    - /srv/salt

# Extension modules path
extension_modules: /usr/share/susemanager/modules

# Master top configuration
master_tops:
  mgr_master_tops: True
```

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 includcde 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.

`/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.

Install with 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. For build assets, see 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.

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 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
```

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, like `parted`, `btrfstools`` among others.

Yomi provides an already prepared image, based on openSUSE Tumbleweed, openSUSE Leap (for Uyuni), or SLE (for SUSE Manager). For Uyuni, the image is pre-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-sle15
```

2. Follow the prompts to complete the installation.

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:

```
cobbler distro add \
  --name=pxe-yomi-image \
  --kernel=/srv/pxe-yomi-image/linux \
  --initrd=/srv/pxe-yomi-image/initrd \
  --boot-files='/srv/tftpboot/pxe-yomi-image/image.initrd=/srv/pxe-yomi-image/image/pxe
-yomi-image-opensuse15.x86_64-1.0.0.initrd /srv/tftpboot/pxe-yomi-
image/image.kernel=/srv/pxe-yomi-image/image/pxe-yomi-image-opensuse15.x86_64-
1.0.0.kernel /srv/tftpboot/pxe-yomi-image/image.md5=/srv/pxe-yomi-image/image/pxe-yomi-
image-opensuse15.x86_64-1.0.0.md5 /srv/tftpboot/pxe-yomi-image/image.xz=/srv/pxe-yomi-
image/image/pxe-yomi-image-opensuse15.x86_64-1.0.0.xz' \
  --kernel-options='rd.kiwi.install.pxe rd.kiwi.install.image=tftp://server-address/pxe-
yomi-image/image.xz rd.kiwi.ramdisk ramdisk_size=1572864 net.ifnames=1'
```

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.

```
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.

```
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:

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

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 https://github.com/openSUSE/yomi.

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: 1MB

Create three partitions:

- Partition 1:
  - Partition Number: 1
  - Partition Size: 1MB
  - Partition Type: boot
- Partition 2:
  - Partition Number: 2
  - Partition Size: 1024MB
  - 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 a new repository: * Repository Name: repo-oss * Repository URL: http://download.opensuse.org/tumbleweed/repo/oss/

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: 1wYJUgpM5$RXMMeASDc035eX.NbYWFl0

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.

Formulas

Formulas are collections of Salt States that have been pre-written by other Salt users and contain generic parameter fields. Formulas allow for reliable reproduction of a specific configuration. Formulas can be installed from RPM packages or an external git repository.

Formulas work best for large, non-trivial, configurations. For trivial tasks, use a state rather than a formula.

Formula data can be managed using the XMLRPC API.

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.

You can use the Uyuni Web UI to apply common 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 **[Salt › Formulas-custom ›]**.

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 (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`.

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.

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.

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.

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.

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 ›]**.

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. Otherwise, keep 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. Otherwise, keep 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.

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.

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.
    - 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.

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 filesystem. Used for falling back to local boot.

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 (for example, `/dev/sda`).
    - 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.

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.

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:

```
Partition Symbolic ID: p1
Partition Size (MiB): 50
Partition Flags: bios_grub
```

Leave the other fields empty.

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:

```
Partition Symbolic ID: p1
Partition Size (MiB): 500
Device Mount Point: /boot/efi
Filesystem Format: vfat
Partition Flags: boot
```

Leave the other fields empty.

Troubleshooting the Saltboot Formula

`msdos` Disklabel Limitations

On `msdos` disk label it is possible to create maximally 4 primary partitions, extended partitions are not supported. This limitation is not present on `GPT` disk label.

For more information on troubleshooting problems with the Saltboot formula, see **[Administration › Tshoot-saltboot ›]**.

TFTPD Formula

The TFTPD formula is an SUSE Manager for Retail formula, used to configure the TFTP service on the 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.

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.

Custom Salt Formulas

Some formulas are provided by default with Uyuni. Other official formulas can be installed as RPM packages. You can also write your own, custom, formulas, and make them available to your systems in the Uyuni Web UI.

This section contains information about installing official formulas, and writing custom formulas.

Install Official Salt Formulas

SUSE releases formulas as RPM packages. Available formulas can be located within the `SUSE-Manager-Server-VERSION-Pool` channel.

If a Salt 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 Salt Formulas from an RPM

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

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

2. Get more information about a formula:

```
zypper info locale-formula
```

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

```
zypper in locale-formula
```

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:

```
/usr/share/salt-formulas/states/locale/
```

The metadata directory contains a `form.yml` file which defines the forms for Uyuni. It also contains an optional `metadata.yml` file that contains additional information about a 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.

Define Formula 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 form, 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:

```
# This file is part of locale-formula.
#
# Foobar is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# Foobar is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with Foobar.  If not, see <http://www.gnu.org/licenses/>.

timezone:
  $type: group

  name:
    $type: select
    $values: ["CET",
              "CST6CDT",
              "EET",
              "EST",
              "EST5EDT",
              "GMT",
              "GMT+0",
              "GMT-0",
```

```
              "GMT0",
              "Greenwich",
              "HST",
              "MET",
              "MST",
              "MST7MDT",
              "NZ",
              "NZ-CHAT",
              "Navajo",
              "PST8PDT",
              "UCT",
              "UTC",
              "Universal",
              "W-SU",
              "WET",
              "Zulu",
              "Etc/GMT+1",
              "Etc/GMT+2",
              "Etc/GMT+3",
              "Etc/GMT+4",
              "Etc/GMT+5",
              "Etc/GMT+6",
              "Etc/GMT+7",
              "Etc/GMT+8",
              "Etc/GMT+9",
              "Etc/GMT+10",
              "Etc/GMT+11",
              "Etc/GMT+12",
              "Etc/GMT-1",
              "Etc/GMT-2",
              "Etc/GMT-3",
              "Etc/GMT-4",
              "Etc/GMT-5",
              "Etc/GMT-6",
              "Etc/GMT-7",
              "Etc/GMT-8",
              "Etc/GMT-9",
              "Etc/GMT-10",
              "Etc/GMT-11",
              "Etc/GMT-12",
              "Etc/GMT-13",
              "Etc/GMT-14",
              "Etc/GMT",
              "Etc/GMT+0",
              "Etc/GMT-0",
              "Etc/GMT0",
              "Etc/Greenwich",
              "Etc/UCT",
              "Etc/UTC",
              "Etc/Universal",
              "Etc/Zulu"
              ]
    $default: CET

  hardware_clock_set_to_utc:
    $type: boolean
    $default: True
...
```

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:

$type

The most important attribute is the `$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:

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

Pillar:

```
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:

```
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 check boxes. It may also be used to check values of select-fields, text-fields, and similar.

A check box should be structured like this:

```
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`.

```
some_group:
  $type: group

  another_group:
    $type: group

    my_checkbox:
      $type: boolean

    my_text:
      $visibleIf: .my_checkbox

  yet_another_group:
    $type: group

    my_text2:
      $visibleIf: ..another_group#my_checkbox
```

If you use multiple groups with the attribute, you can allow a users 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:

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

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

$values

Can only be used together with `$type` Use to specify the different options in the select-field. `$values` must be a list of possible values to select. For example:

```
select_something:
  $type: select
  $values: ["option1", "option2"]
```

Or:

```
select_something:
  $type: select
  $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:

```
some_group:
  $type: group

  another_group:
    $type: group

      my_checkbox:
        $type: boolean
```

An example condition is:

```
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:

```
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 expresion 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:

```
      hardware:
        $type: text
        $name: Hardware Type and Address
        $placeholder: Enter hardware-type hardware-address (e.g. "ethernet
AA:BB:CC:DD:EE:FF")
        $help: Hardware Identifier - prefix is mandatory
        $match: "\\w+ [A-Z]{2}:[A-Z]{2}:[A-Z]{2}:[A-Z]{2}:[A-Z]{2}:[A-Z]{2}"
```

Expression language

You must use the jexl expression language to write conditions.

Given a structure like this:

```
some_group:
  $type: group

  another_group:
    $type: group

      my_checkbox:
        $type: boolean
```

An example condition is:

```
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:

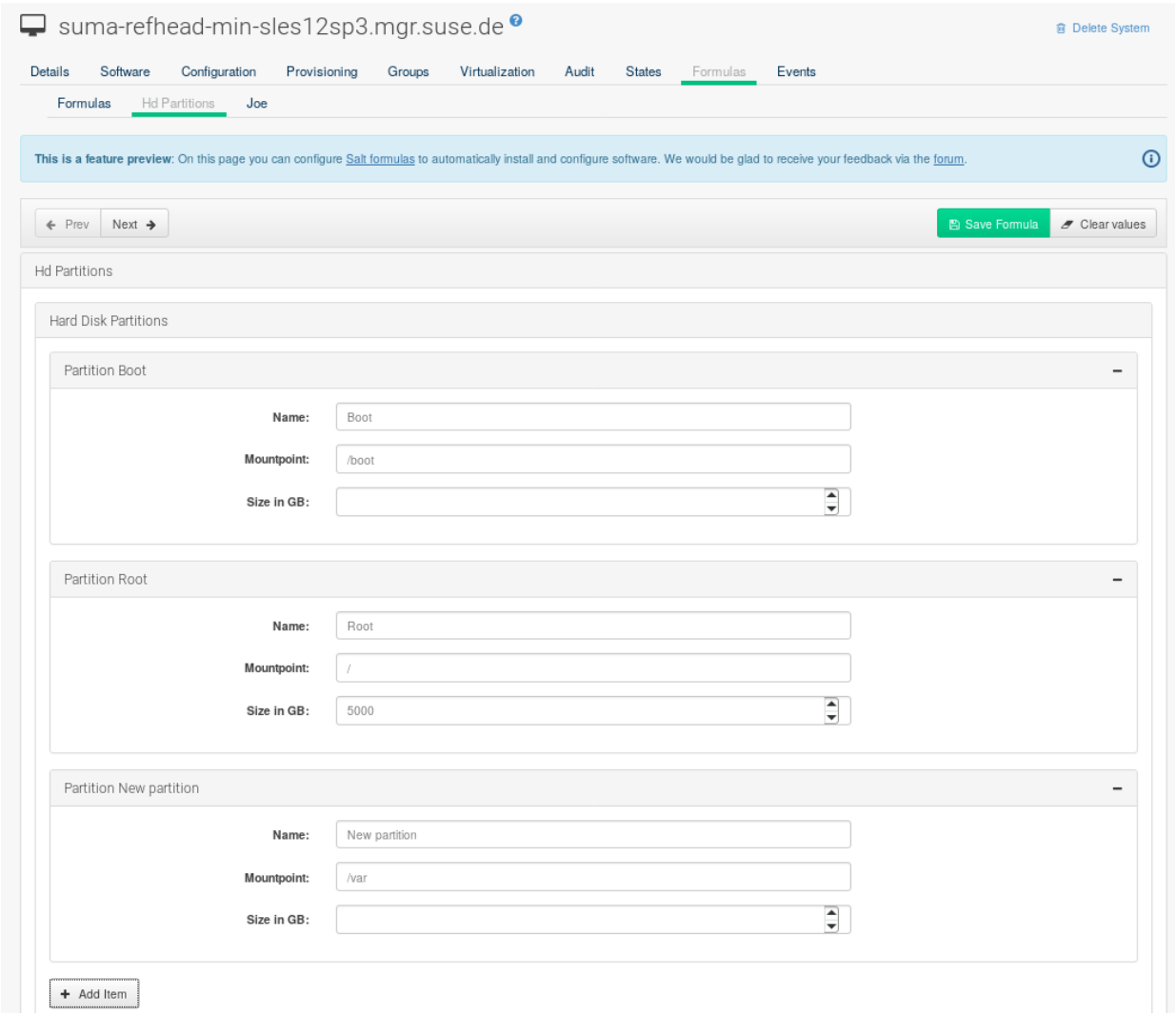
```
some_group:
  $type: group

  another_group:
    $type: group

    my_checkbox:
      $type: boolean

    my_text:
      $visible: this.parent.value.my_checkbox

  yet_another_group:
    $type: group

    my_text2:
      $visible: this.parent.parent.value.another_group.my_checkbox
```

Listing 2. Example: Basic edit-group

```
partitions:
  $name: "Hard Disk Partitions"
  $type: "edit-group"
  $minItems: 1
  $maxItems: 4
  $itemName: "Partition ${name}"
  $prototype:
    name:
      $default: "New partition"
    mountpoint:
      $default: "/var"
    size:
      $type: "number"
      $name: "Size in GB"
  $default:
    - name: "Boot"
      mountpoint: "/boot"
    - name: "Root"
      mountpoint: "/"
      size: 5000
```

Click **[Add]** to fill the form with the default values.

The formula is called `hd-partitions` and will appear as `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]**.

Listing 3. Example: Nested edit-group

```
users:
  $name: "Users"
  $type: edit-group
  $minItems: 2
  $maxItems: 5
  $prototype:
    name:
      $default: "username"
    password:
      $type: password
    groups:
      $type: edit-group
      $minItems: 1
      $prototype:
        group_name:
          $type: text
  $default:
    - name: "root"
      groups:
        - group_name: "users"
        - group_name: "admins"
    - name: "admin"
      groups:
        - group_name: "users"
```

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:

```
{% if pillar.installSomething %}
something:
  pkg.installed
{% else %}
anotherPackage:
  pkg.installed
{% endif %}
```

Another example:

```
{% 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:

```
{% 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:

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

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`.

You can choose to override defined values from pillars. For example, by overriding `some.pillar.data.package` in this example:

```
{% 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:

```
{% 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:

```
{% 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:

```
{% 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.

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:

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

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 a Uyuni installation. 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`.

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/.

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.

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 by generating a temporary roster file for each Salt SSH call. The location of the temporary roster file is supplied to `salt-ssh` using the `--roster-file= option`.

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. 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.

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 the value of `ssh_push_sudo_user` is not root, then the `--sudo` options of `salt-ssh` are used.

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.

Call Sequence

Salt SSH calls run in this sequence:

1. Prepare the Salt roster for the call
    a. Create remote port forwarding option if the contact method is `ssh-push-tunnel`
    b. Compute the `ProxyCommand` if the client is connected through a proxy
    c. Create Roster content
2. Create a temporary roster file
3. Execute a synchronous `salt-ssh` call using the API
4. Remove the temporary roster file

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`

Bootstrap Sequence

Salt SSH is used to bootstrap Salt clients. This happens for both regular and SSH clients.

The bootstrap sequence differs slightly from other Salt SSH calls.

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.
4. If contact method is `ssh-push-tunnel`, fill the remote port forwarding option.
5. If the client connects through a proxy, compute the `ProxyCommand` option. This depends on the path used to connect to the proxy. For example, server to proxy1 to proxy2 to client.
6. Generate the roster for bootstrapping into a temporary file.
7. Execute this command using the Salt API:

   ```
   salt-ssh --roster-file=<temporary_bootstrap_roster> minion state.apply
   certs,<bootstrap_state>`
   ```

   For `bootstrap_state`, use `bootstrap` for regular clients or `ssh_bootstrap` for SSH clients.

Pillar data contains:

- mgr_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:

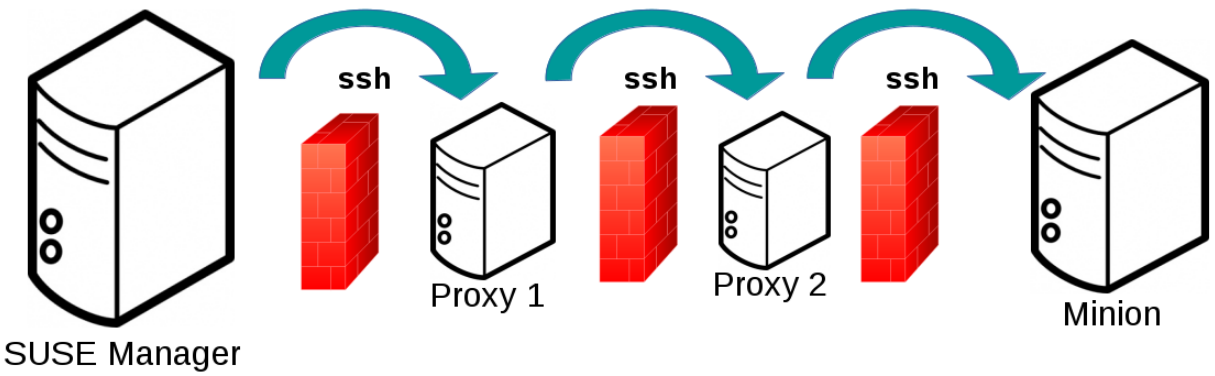
- `hostname`
- `user`
- `password`
- `port`
- `remote_port_forwards`: the remote port forwarding SSH option
- `ssh_options`: other SSH options:
  - `ProxyCommand` if the client connects through a proxy
- `timeout`: defaults to 180 seconds

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

Proxy Support

Salt SSH Bootstrap Process

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.

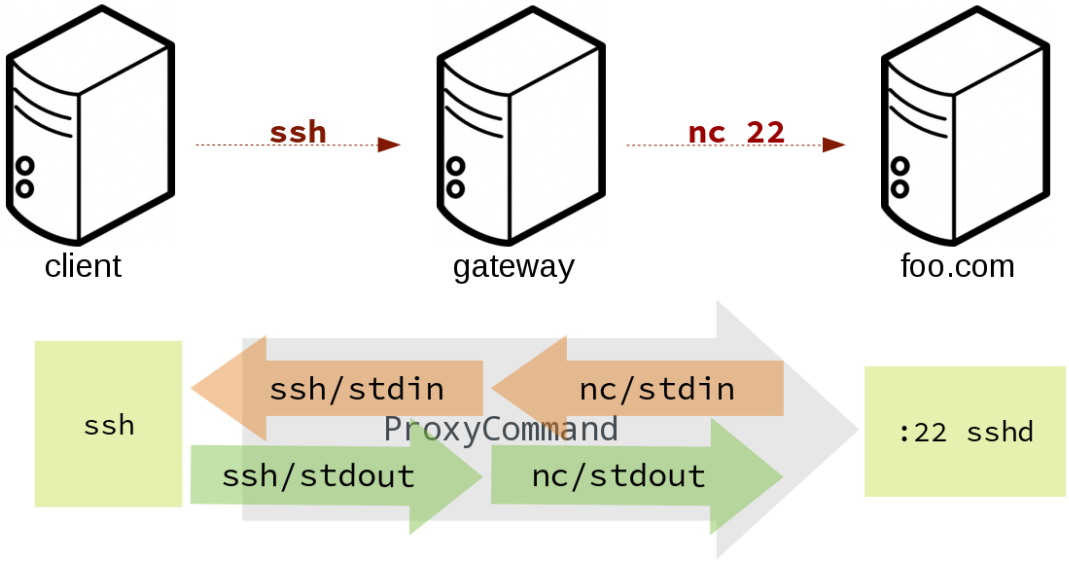


Uyuni uses `ProxyCommand` to redirect SSH connections through proxies. This options invokes an arbitrary command that is expected to connect to the SSH port on the target host. The SSH process uses standard input and output of the command to communicate with the remote SSH daemon.

`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
```



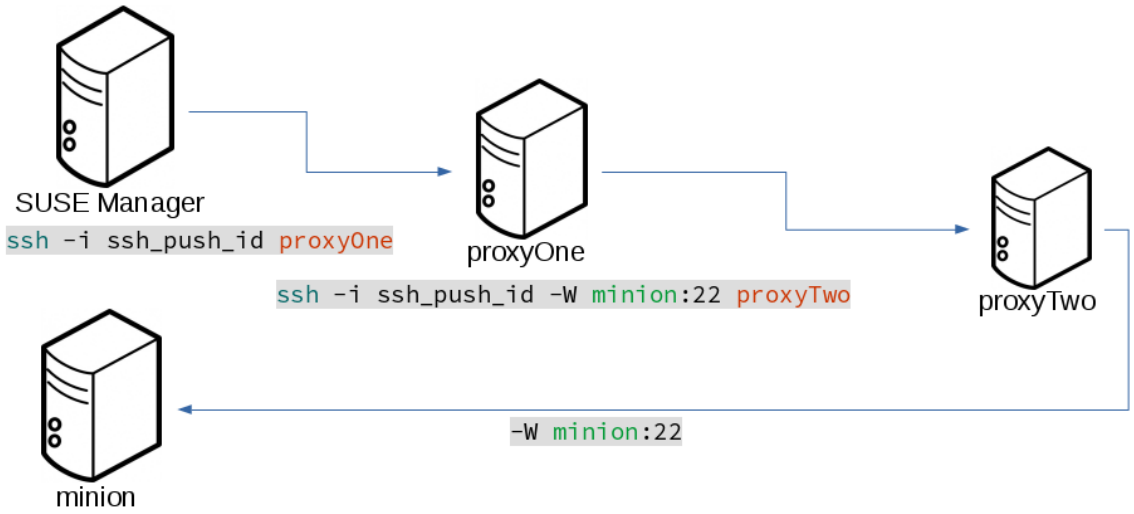
The Salt SSH calls run in this sequence when a proxy is in use:

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:

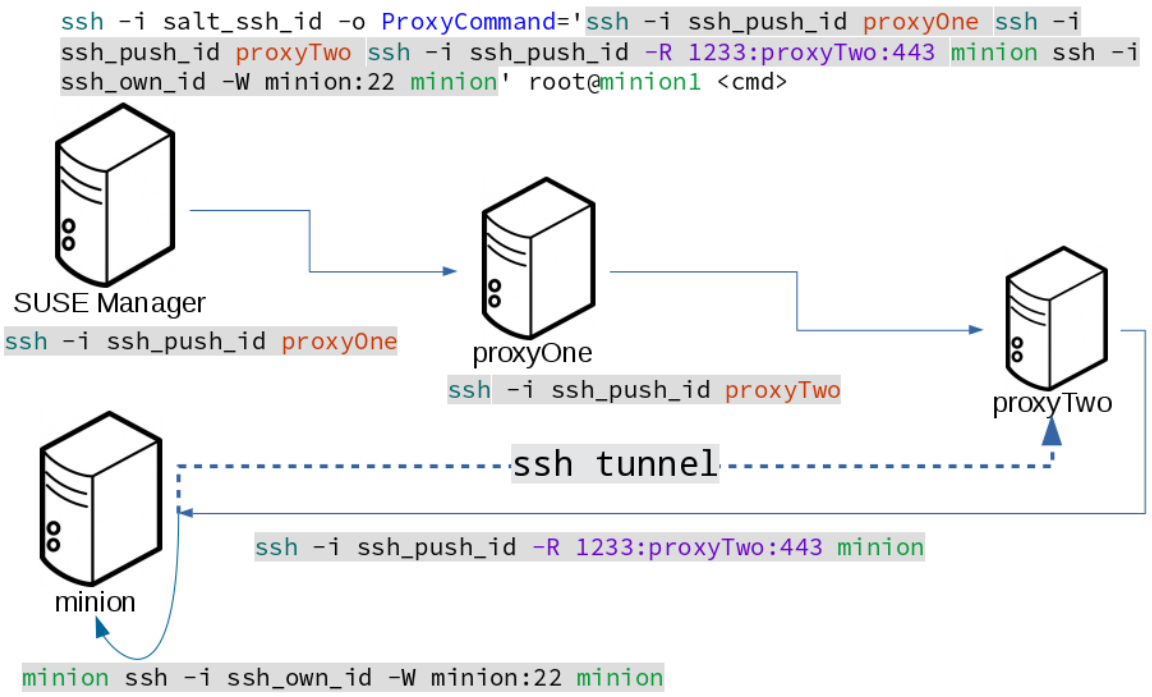
```
# Connect the server to the first proxy:
/usr/bin/ssh -i /srv/susemanager/salt/salt_ssh/mgr_ssh_id -o StrictHostKeyChecking=no -o
User=mgrsshtunnel  proxy1

# 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
```

```
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>
```



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

```
# 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
```

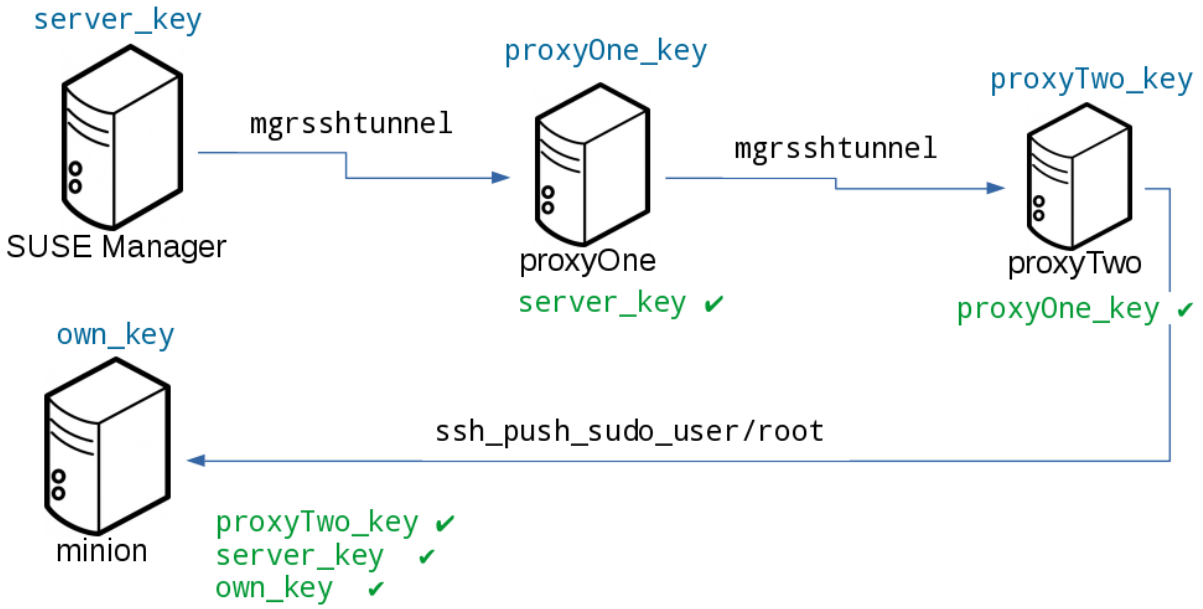


Users and SSH Key Management

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.

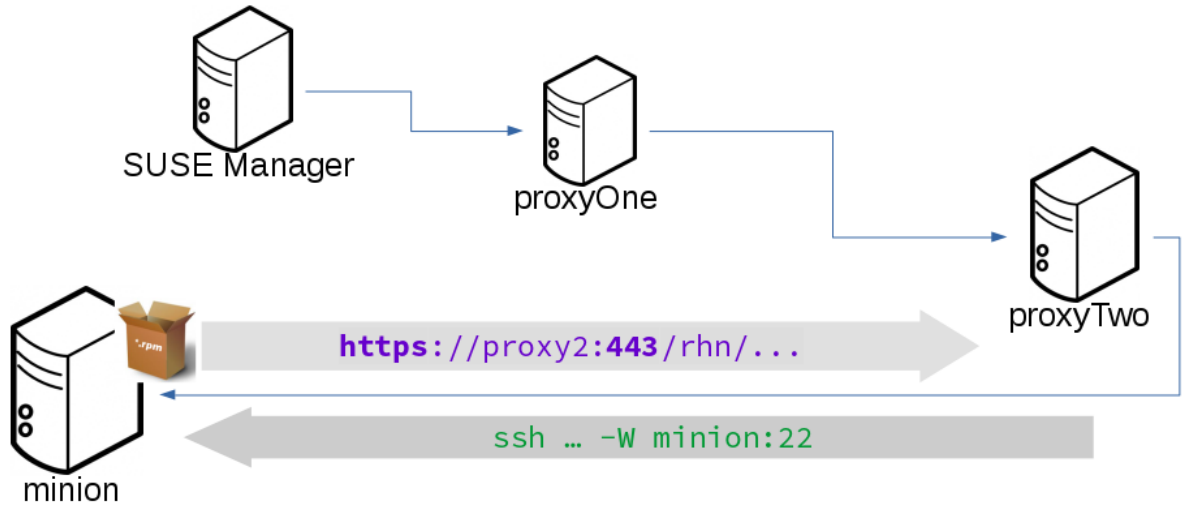


Repository Access with a Proxy

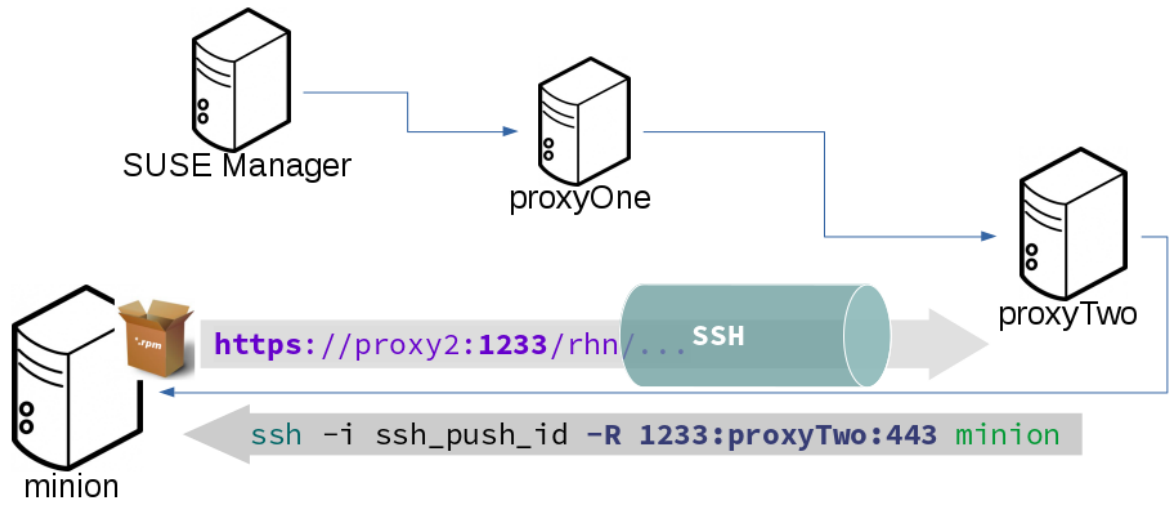
When Uyuni connects to a repository using a proxy, it can use either `ssh-push` or `ssh-push-tunnel`.

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.



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.

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.

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.

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:

```
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`.

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.

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.

Large installations require standard Salt clients. These instructions cannot be used in environments using traditional clients or Salt SSH minions.

Hardware and Infrastructure

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=noop` 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 noop > /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.
- 16 GB RAM, and sufficient storage for caching.

Clients should never be directly attached to the Uyuni Server in production systems.

In large scale installations, the Uyuni Proxy is used primarily as a local cache for content between the server and clients. Using proxies in this way can substantially reduce download time for clients, and decrease Server egress bandwidth use.

The number of clients per proxy will affect the download time. Always take network structure and available bandwidth into account.

We recommend you estimate the download time of typical usage to determine how many clients to connnect to each proxy. To do this, you will need to estimate the number of package upgrades required in every patch cycle. You can use this formula to calculate the download time:

```
Size of updates * Number of clients / Theoretical download speed / 60
```

For example, the total time needed to transfer 400 MB of upgrades through a physical link speed of 1 GB/s to 3000 clients:

```
400 MB  * 3000 / 119 MB/s / 60 = 169 min
```

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.

Salt 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:

```
for k in $(salt-key -l un|grep -v Unaccepted); do salt-key -y -a $k; sleep 15; done
```

Salt Clients and the RNG

All communication to and from Salt 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.

Clients Running with Unaccepted Salt Keys

Clients which have not been onboarded, that is clients running with unaccepted Salt keys, consume more resources than 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.

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:

```
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*.

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.

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 **[Salt › Large-scale-tuning ›]**.

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: * https://docs.saltstack.com/en/latest/topics/tutorials/intro_scale.html#too-many-minions-re-authing * https://docs.saltstack.com/en/getstarted/system/communication.html

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.

We strongly recommend that you contact SUSE Consulting for assistance with tuning.

SUSE will not provide support for catastrophic failure when these advanced parameters are modified without consultation.

Tuning is not required on installations of fewer than 1000 clients. Do not perform these instructions on small or medium scale installations.

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.

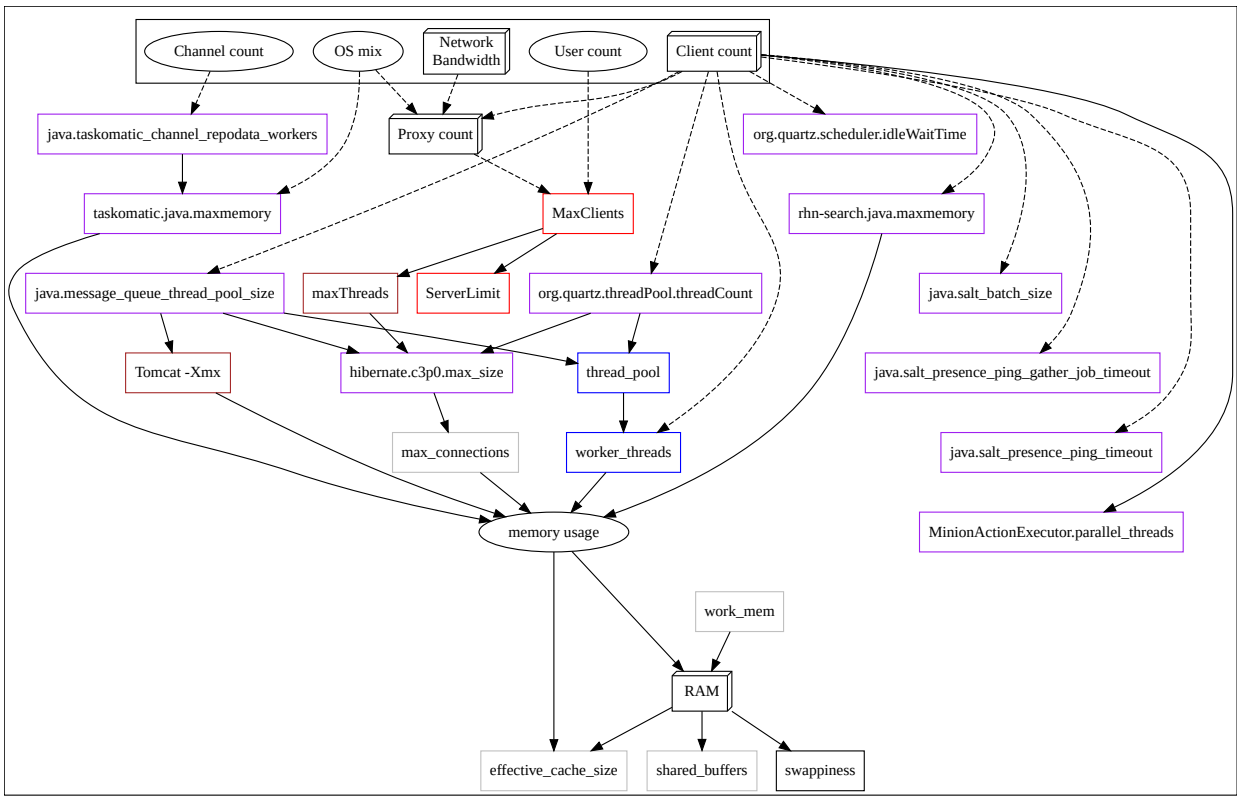
Environmental variables influence, either directly or indirectly, the value of most configuration parameters. During the tuning process, the configuration parameters are manipulated to improve system performance.

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.

Environmental Variables

This section contains information about environmental variables (inputs to the tuning process).

Network Bandwidth

A measure of the typically available egress bandwith 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.

Parameters

This section contains information about the available parameters.

`MaxClients`

Description	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.

Tune when	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`.
Value default	150
Value recommendation	150-500
Location	`/etc/apache2/server-tuning.conf`, in the `prefork.c` section
Example	`MaxClients = 200`
After changing	Immediately change `ServerLimit` and check `maxThreads` for possible adjustment.
Notes	This parameter was renamed to `MaxRequestWorkers`, both names are valid.
More information	https://httpd.apache.org/docs/2.4/en/mod/mpm_common.html#maxrequestworkers

`ServerLimit`

Description	The number of Apache httpd processes serving HTTP requests simultaneously. The number must equal `MaxClients`.
Tune when	`MaxClients` changes
Value default	150
Value recommendation	The same value as `MaxClients`
Location	`/etc/apache2/server-tuning.conf`, in the `prefork.c` section
Example	`ServerLimit = 200`
More information	https://httpd.apache.org/docs/2.4/en/mod/mpm_common.html#serverlimit

`maxThreads`

Description	The number of Tomcat threads dedicated to serving HTTP requests
Tune when	`MaxClients` changes. `maxThreads` must always be equal or greater than `MaxClients`
Value default	150
Value recommendation	The same value as `MaxClients`
Location	`/etc/tomcat/server.xml`
Example	`<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" URIEncoding="UTF-8" address="127.0.0.1" maxThreads="200" connectionTimeout="20000"/>`
More information	https://tomcat.apache.org/tomcat-9.0-doc/config/http.html

Tomcat's `-Xmx`

Description	The maximum amount of memory Tomcat can use
Tune when	`java.message_queue_thread_pool_size` is increased or `OutOfMemoryException` errors appear in `/var/log/rhn/rhn_web_ui.log`
Value default	1 GiB
Value recommendation	4-8 GiB
Location	`/etc/sysconfig/tomcat`
Example	`JAVA_OPTS="··· -Xmx8G ···"`
After changing	Check memory usage
More information	https://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html

`java.message_queue_thread_pool_size`

Description	The maximum number of threads in Tomcat dedicated to asynchronous operations, including handling of incoming Salt events
Tune when	Client count increases significantly
Value default	5
Value recommendation	50 - 150
Location	`/etc/rhn/rhn.conf`
Example	`java.message_queue_thread_pool_size = 50`
After changing	Check `hibernate.c3p0.max_size`, as each thread consumes a PostgreSQL connection, starvation might happen if the allocated connection pool is insufficient. Check `thread_pool`, as each thread might perform Salt API calls, starvation might happen if the allocated Salt thread pool is insufficient. Check Tomcat's `-Xmx`, as each thread consumes memory, `OutOfMemoryException` might be raised if insufficient.
More information	`man rhn.conf`

`java.salt_batch_size`

Description	The maximum number of minions concurrently executing a scheduled action.
Tune when	Client count reaches several thousands and actions are not executed quickly enough.
Value default	200
Value recommendation	200-500
Location	`/etc/rhn/rhn.conf`
Example	`java.salt_batch_size = 300`
After changing	Check memory usage. Monitor memory usage closely before and after the change.
More information	Salt Rate Limiting

java.salt_presence_ping_timeout

Description	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 (`find_job`) 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.
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`
More information	Salt Timeouts

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)"`
Value default	1 second
Value recommendation	1-100 seconds
Location	`/etc/rhn/rhn.conf`
Example	`java.salt_presence_ping_gather_job_timeout = 10`
More information	Salt Timeouts

`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.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`

`taskomatic.java.maxmemory`

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`

`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

`org.quartz.scheduler.idleWaitTime`

Description	Cycle time for Taskomatic. Decreasing this value lowers the latency of Taskomatic.
Tune when	Client count is in the thousands.
Value default	5000 ms
Value recommendation	1000-5000 ms
Location	`/etc/rhn/rhn.conf`
Example	`org.quartz.scheduler.idleWaitTime = 1000`
More information	http://www.quartz-scheduler.org/documentation/2.4.0-SNAPSHOT/configuration.html

`MinionActionExecutor.parallel_threads`

Description	Number of Taskomatic threads dedicated to sending commands to Salt clients as a result of actions being executed.
Tune when	Client count is in the thousands.
Value default	1
Value recommendation	1-10
Location	`/etc/rhn/rhn.conf`
Example	`taskomatic.com.redhat.rhn.taskomatic.task.MinionActionExecutor.parallel_threads = 10`

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/#maxPoolSize

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
Location	`/etc/rhn/rhn.conf`
Example	`rhn-search.java.maxmemory = 4096`
After changing	Check memory usage.

`shared_buffers`

Description	The amount of memory reserved for PostgreSQL shared buffers, which contain caches of database tables and index data.
Tune when	RAM changes
Value default	25% of total RAM
Value recommendation	25-40% of total RAM
Location	`/var/lib/pgsql/data/postgresql.conf`
Example	`shared_buffers = 8192MB`
After changing	Check memory usage.
More information	https://www.postgresql.org/docs/10/runtime-config-resource.html#GUC-SHARED-BUFFERS

`max_connections`

Description	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.
Tune when	`hibernate.c3p0.max_size` changes significantly, as that parameter determines the maximum number of connections available to Tomcat and Taskomatic
Value default	400
Value recommendation	2 * hibernate.c3p0.max_size + 50, if less than 1000
Location	`/var/lib/pgsql/data/postgresql.conf`
Example	`max_connections = 250`
After changing	Check memory usage. Monitor memory usage closely before and after the change.

More information	https://www.postgresql.org/docs/10/runtime-config-connection.html#GUC-MAX-CONNECTIONS

`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	Individual query operations are too slow, and value is below 5 MB
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/10/runtime-config-resource.html#GUC-WORK-MEM

`effective_cache_size`

Description	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.
Tune when	Hardware RAM or memory usage increase significantly
Value recommendation	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)
Location	`/var/lib/pgsql/data/postgresql.conf`
Example	`effective_cache_size = 24GB`

After changing	Check memory usage
Notes	This is an estimation for the query planner, not an allocation.
More information	https://www.postgresql.org/docs/10/runtime-config-query.html#GUC-EFFECTIVE-CACHE-SIZE

`thread_pool`

Description	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.
Tune when	`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.
Value default	100
Value recommendation	100-500, but should be higher than the sum of `java.message_queue_thread_pool_size` and `org.quartz.threadPool.threadCount`
Location	`/etc/salt/master.d/susemanager.conf`, in the `rest_cherrypy` section.
Example	`thread_pool: 100`
After changing	Check `worker_threads` for adjustment.
More information	https://docs.saltstack.com/en/latest/ref/netapi/all/salt.netapi.rest_cherrypy.html#performance-tuning

`worker_threads`

Description	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).
Tune when	Client count increases significantly, `thread_pool` increases significantly, or `SaltReqTimeoutError` or `Message timed out` errors appear in `/var/log/salt/master`.
Value default	8
Value recommendation	8-200
Location	`/etc/salt/master.d/tuning.conf`
Example	`worker_threads: 50`
After changing	Check memory usage. Monitor memory usage closely before and after the change.
More information	https://docs.saltstack.com/en/latest/ref/configuration/master.html#worker-threads

`swappiness`

Description	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.
Tune when	RAM increases, or swap is used when RAM memory is sufficient.
Value default	60
Value recommendation	1-60. For 128 GB of RAM, 10 is expected to give good results.
Location	`/etc/sysctl.conf`
Example	`vm.swappiness = 20`
More information	https://documentation.suse.com/sles/15-SP1/html/SLES-all/cha-tuning-memory.html#cha-tuning-memory-vm

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.