# Client Configuration Guide

Uyuni 4.0

March 16, 2019

# Table of Contents

# Client Configuration Overview

For Uyuni 3 and later, you can use either the traditional or Salt client management frameworks. Optionally you may use a mixture of both, depending on environment and software requirements.

**Salt**

Is an end-to-end data-center automation tool which may also be used outside the scope of SUSE Manager to introduce reactive, real-time orchestration, and configuration management.

# SUSE Customer Center (SCC)

SUSE Customer Center (SCC) is the central place to manage your purchased SUSE subscriptions, helping you access your update channels and get in contact with SUSE experts. The user-friendly interface gives you a centralized view of all your SUSE subscriptions, allowing you to easily find all subscription information you need. The improved registration provides faster access to your patches and updates. SUSE Customer Center is also designed to provide a common platform for your support requests and feedback. Discover a new way of managing your SUSE account and subscriptions via one interface—anytime, anywhere. For more information on using SUSE Customer Center, see: https://scc.suse.com/docs/userguide.

# Activation Keys

Activation keys are used with both traditional and Salt clients to ensure that your clients have the correct software entitlements, are connecting to the appropriate channels, and are subscribed to the relevant groups. Each activation key is bound to an organization, which you can set when you create the key.

This section goes over activation key core concepts. To learn how to create activation keys for manual registration see:

## Managing Activation Keys

### What are Activation Keys?

An `activation key` in Uyuni is a group of configuration settings with a label. You can apply all configuration settings associated with an activation key by adding its label as a parameter to a bootstrap script. Under normal operating conditions best practices suggest using an activation key label in combination with a bootstrap script.

An activation key can specify:

- Channel Assignment
- System Types (Traditionally called Add-on Entitlements)
- Contact Method
- Configuration Files
- Packages to be Installed
- System Group Assignment

Activation keys are just a collection of configuration settings which have been given a label name and then added to a bootstrap script. When the bootstrap script is executed all configuration settings associated with the label are applied to the system the script is run on.

### Provisioning and Configuration

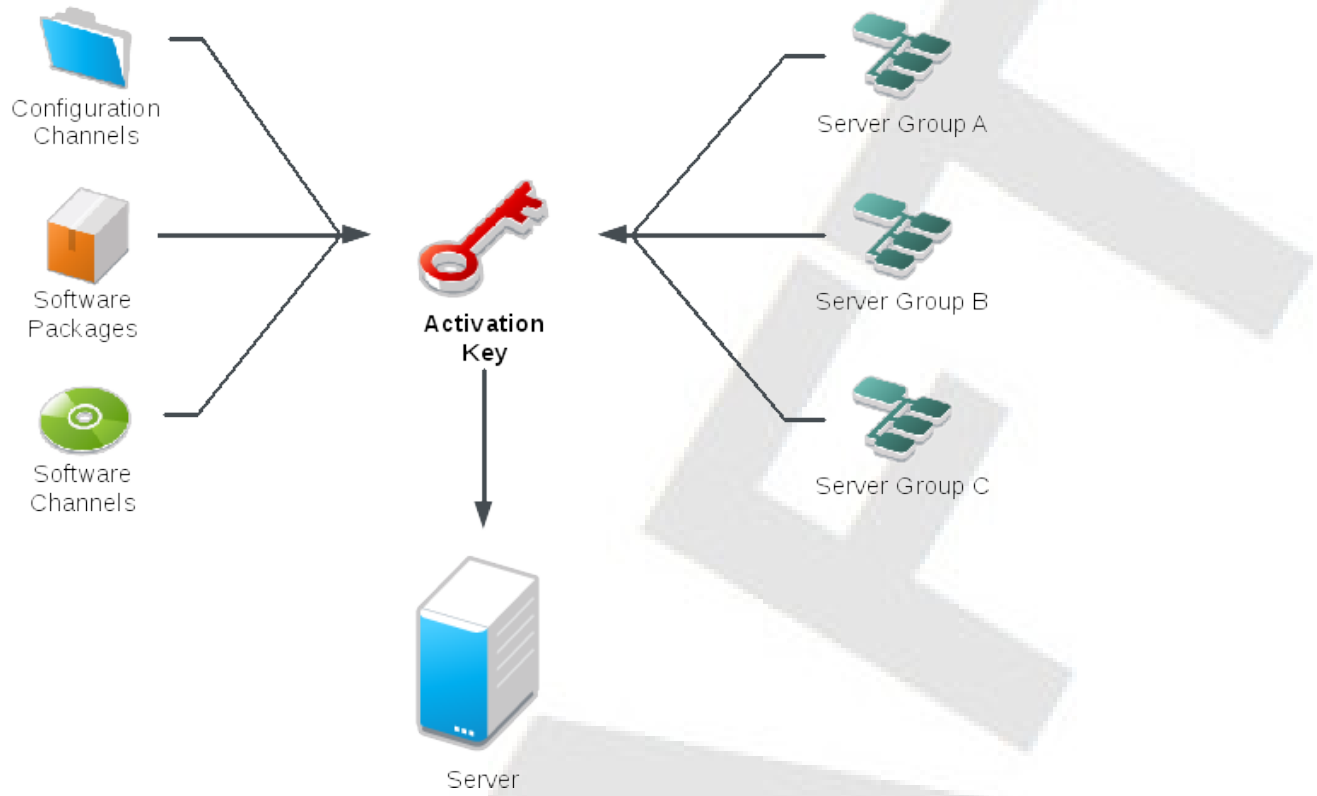



*Figure 1. Provisioning and Configuration Overview*

## Activation Keys Best Practices

There are a few important concepts which should be kept in mind when creating activation keys. The following sections provide insight when creating and naming your activation keys.

### Key Label Naming

One of the most important things to consider during activation key creation is label naming. Creating names which are associated with your organization's infrastructure will make it easier for you when performing more complex operations. When naming key labels keep the following in mind:

- OS naming (mandatory): Keys should always refer to the OS they provide settings for
- Architecture naming (recommended): Unless your company is running on one architecture only, for example x86_64, then providing labels with an architecture type is a good idea.
- Server type naming: What is, or what will this server be used for?
- Location naming: Where is the server located? Room, building, or department?
- Date naming: Maintenance windows, quarter, etc.
- Custom naming: What naming scheme suits your organizations needs?

Example activation key label names:

```
sles12-sp2-web_server-room_129-x86_64
```

```
sles12-sp2-test_packages-blg_502-room_21-ppc64le
```

### Channels which will be Included

When creating activation keys you also need to keep in mind which channels (software sources) will be associated with it.

*Default Base Channel*

Keys should have a specific base channel assigned to it, for example `SLES12-SP2-Pool-x86_64`. If this is not the case Uyuni cannot use specific stages. Using the default base channel is not recommended and may cause problems.

- Channels to be included:
  - suse-manager-tools
- Typical packages to be included:
  - osad (pushing tasks)
    - Installs python-jabberpy and pyxml as dependencies
  - rhncfg-actions (Remote Command, Configuration Managment)
    - Installs rhncfg and rhncfg-client as dependencies

## Combining Activation Keys

You can combine activation keys when executing the bootstrap script on your clients. Combining keys allows for more control on what is installed on your systems and reduces duplication of keys for large complex environments.

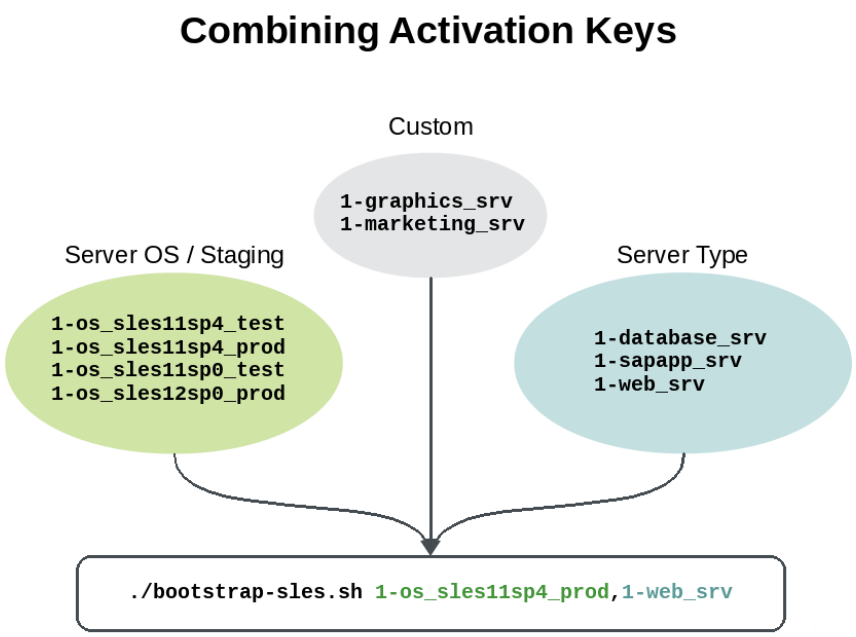



*Figure 2. Combining Activation Keys*

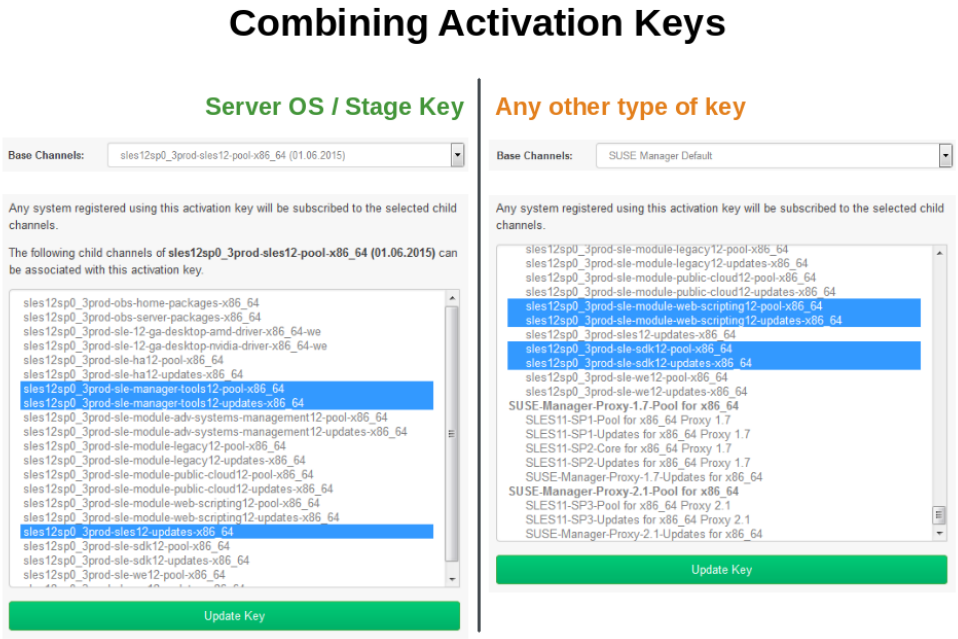



*Figure 3. Combining Activation Keys 2*

## Using Activation Keys and Bootstrap with Traditional Clients (Non-Salt)

Create the initial bootstrap script template from the command line on the Uyuni server with:

```
# mgr-bootstrap
```

This command will generate the bootstrap script and place them in `/srv/www/htdocs/pub/bootstrap`.

Alternatively you may use the Web UI to create your bootstrap script template. For more information, see xref:FILENAME.adoc#s3-sattools-config-bootstrap[].

Use the Web UI to create your keys. From the Web UI proceed to menu:Overview[Tasks

# Traditional vs. Salt

Coming Soon…

# Bootstrapping Overview

Coming Soon…

# Contact Methods

## Selecting a Contact Method

Uyuni provides several methods for communication between client and server. All commands your Uyuni server sends its clients to do will be routed through one of them. Which one you select will depend on your network infrastructure. The following sections provide a starting point for selecting a method which best suits your network environment.

*Contact Methods and Salt*

This chapter is only relevant for traditional clients as Salt clients (minions) utilize a Salt specific contact method. For general information about Salt clients, see [salt.gs.guide.intro].

## Default (the Uyuni Daemon rhnsd)

The Uyuni daemon (`rhnsd`) runs on client systems and periodically connects with Uyuni to check for new updates and notifications. The daemon, which runs in the background, is started by `rhnsd.service` . By default, it will check every 4 hours for new actions, therefore it may take some time for your clients to begin updating after actions have been scheduled for them.

To check for updates, `rhnsd` runs the external `mgr_check` program located in `/usr/sbin/` . This is a small application that establishes the network connection to Uyuni. The SUSE Manager daemon does not listen on any network ports or talk to the network directly. All network activity is done via the `mgr_check` utility.

*Auto accepting (EULAs)*

When new packages or updates are installed on the client using Uyuni, any end user licence agreements (EULAs) are automatically accepted. To review a package EULA, open the package detail page in the Web UI.

This figure provides an overview of the default `rhnsd` process path. All items left of the `Python XMLRPC server` block represent processes running on a Uyuni client.

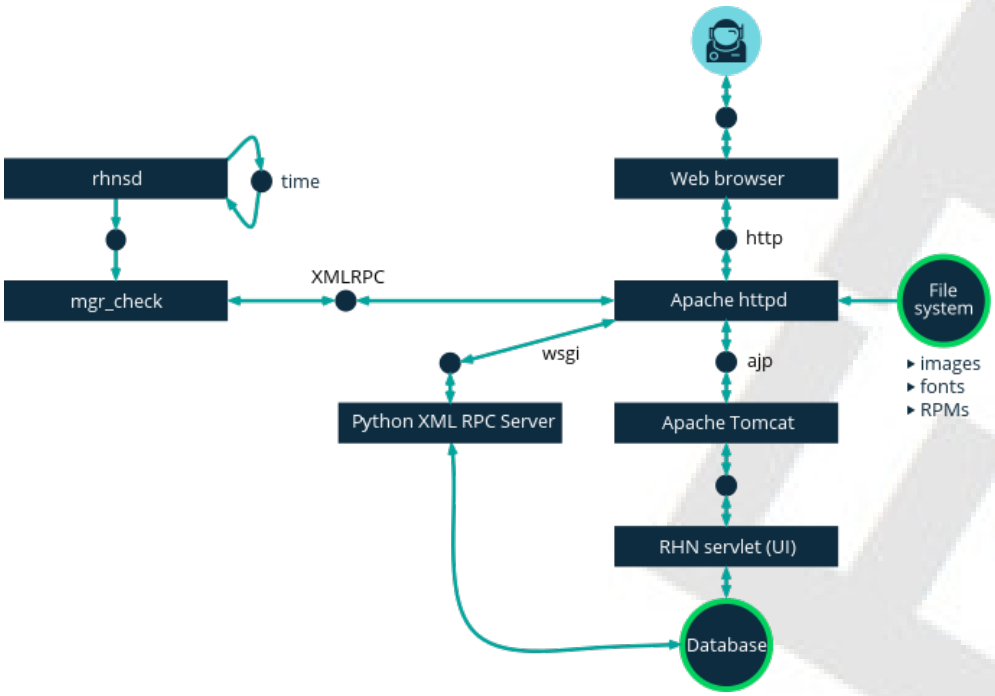



*Figure 4. rhnsd Contact Method*

## Configuring Uyuni rhnsd Daemon

The Uyuni daemon can be configured by editing the file on the client:

```
/etc/sysconfig/rhn/rhnsd
```

This is the configuration file the rhnsd initialization script uses. An important parameter for the daemon is its check-in frequency. The default interval time is four hours (240 minutes). If you modify the configuration file, you must as root restart the daemon with `systemctl rhnsd restart`.

*Minimum Allowed Check-in Parameter*

The minimum allowed time interval is one hour (60 minutes). If you set the interval below one hour, it will change back to the default of 4 hours (240 minutes).

## Viewing rhnsd Daemon Status

You can view the status of rhnsd by typing the command `systemctl status rhnsd` as root .

# Push via SSH

Push via SSH is intended to be used in environments where your clients cannot reach the Uyuni server directly to regularly check in and, for example, fetch package updates.

In detail, this feature enables a Uyuni located within an internal network to manage clients located on a “Demilitarized Zone” (DMZ) outside of the firewall protected network. Due to security reasons, no

system on a DMZ is authorized to open a connection to the internal network and therefore your Uyuni server. The solution is to configure Push via SSH which utilizes an encrypted tunnel from your Uyuni server on the internal network to the clients located on the DMZ. After all actions/events are executed, the tunnel is closed. The server will contact the clients in regular intervals (using SSH) to check in and perform all actions and events.

*Push via SSH Unsupported Actions*

Certain actions are currently not supported on scheduled clients which are managed via Push via SSH. This includes re-installation of systems using the provisioning module.

The following figure provides an overview of the Push via SSH process path. All items left of the `Taskomatic` block represent processes running on a Uyuni client.

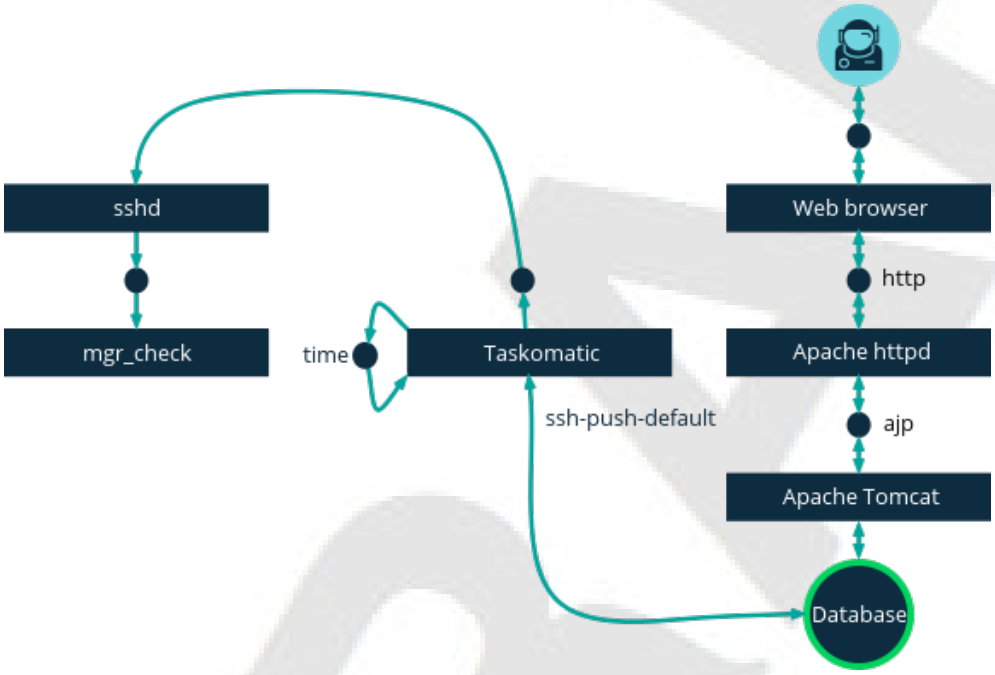



*Figure 5. Push via SSH Contact Method*

## Configuring the Server for Push via SSH

For tunneling connections via SSH, two available port numbers are required, one for tunneling HTTP and the second for tunneling via HTTPS (HTTP is only necessary during the registration process). The port numbers used by default are `1232` and `1233`. To overwrite these, add two custom port numbers greater than 1024 to `/etc/rhn/rhn.conf` like this:

```
ssh_push_port_http = high port 1
ssh_push_port_https = high port 2
```

If you would like your clients to be contacted via their hostnames instead of an IP address, set the following option:

```
ssh_push_use_hostname = true
```

It is also possible to adjust the number of threads to use for opening client connections in parallel. By default two parallel threads are used. Set `taskomatic.ssh_push_workers` in `/etc/rhn/rhn.conf` like this:

```
taskomatic.ssh_push_workers = number
```

## Using sudo with Push via SSH

For security reasons you may desire to use sudo and SSH into a system as a user other than root . The following procedure will guide you through configuring sudo for use with Push via SSH.

*sudo Requirements*

The packages `spacewalk-taskomatic >= 2.1.165.19` and `spacewalk-certs-tools ⇒ 2.1.6.7` are required for using sudo with Push via SSH.

*Procedure: Configuring sudo*

1. Set the following parameter on the server located in `/etc/rhn/rhn.conf` .

   ```
   ssh_push_sudo_user =`user`
   ```

   The server will use sudo to ssh as the configured `user`.

2. You must create the user specified in Procedure: Configuring sudo on each of your clients and the following parameters should be commented out within each client's `/etc/sudoers` file:

   ```
   #Defaults targetpw   # ask for the password of the target user i.e. root
   #ALL    ALL=(ALL) ALL   # WARNING! Only use this together with 'Defaults targetpw'!
   ```

3. Add the following lines beneath the `\## User privilege specification` section of each client's `/etc/sudoers` file:

   ```
   <user> ALL=(ALL) NOPASSWD:/usr/sbin/mgr_check
   <user> ALL=(ALL) NOPASSWD:/home/<user>/enable.sh
   <user> ALL=(ALL) NOPASSWD:/home/<user>/bootstrap.sh
   ```

4. On each client add the following two lines to the `/home/user/.bashrc` file:

```
PATH=$PATH:/usr/sbin
export PATH
```

## Client Registration

As your clients cannot reach the server, you will need to register your clients from the server. A tool for performing registration of clients from the server is included with Uyuni and is called `mgr-ssh-push-init`. This tool expects a client's hostname or IP address and the path to a valid bootstrap script located in the server's filesystem for registration as parameters.

*Specifying Ports for Tunneling before Registering Clients*

The ports for tunneling need to be specified before the first client is registered. Clients already registered before changing the port numbers must be registered again, otherwise the server will not be able to contact them anymore.

`mgr-ssh-push-init` *Disables rhnsd*

The `mgr-ssh-push-init` command disables the `rhnsd` daemon which normally checks for updates every 4 hours. Because your clients cannot reach the server without using the Push via SSH contact method, the `rhnsd` daemon is disabled.

For registration of systems which should be managed via the Push via SSH tunnel contact method, it is required to use an activation key that is configured to use this method. Normal `Push via SSH` is unable to reach the server. For managing activation keys, see Managing Activation Keys.

Run the following command as root on the server to register a client:

```
# mgr-ssh-push-init --client client --register \
/srv/www/htdocs/pub/bootstrap/bootstrap_script --tunnel
```

To enable a client to be managed using Push via SSH (without tunneling), the same script may be used. Registration is optional since it can also be done from within the client in this case. `mgr-ssh-push-init` will also automatically generate the necessary SSH key pair if it does not yet exist on the server:

```
# mgr-ssh-push-init --client`client`--register bootstrap_script
```

When using the Push via SSH tunnel contact method, the client is configured to connect to Uyuni using the high ports mentioned above. Tools like `rhn_check` and `zypper` will need an active SSH session with the proper port forwarding options in order to access the Uyuni API. To verify the Push via SSH tunnel connection manually, run the following command on the Uyuni server:

```
# ssh -i /root/.ssh/id_susemanager -R high port: susemanager :443`client`zypper ref
```

## API Support for Push via SSH

The contact method to be used for managing a server can also be modified via the API. The following example code (python) shows how to set a system's contact method to `ssh-push`. Valid values are:

- `default` (pull)
- `ssh-push`
- `ssh-push-tunnel`

```
client = xmlrpclib.Server(SUMA_HOST + "/rpc/api", verbose=0)
key = client.auth.login(SUMA_LOGIN, SUMA_PASSWORD)
client.system.setDetails(key, 1000012345, {'contact_method' : 'ssh-push'})
```

*Migration and Management via Push via SSH*

When a system should be migrated and managed using Push via SSH, it requires setup using the `mgr-ssh-push-init` script before the server can connect via SSH. This separate command requires human interaction to install the server's SSH key onto the managed client (root password). The following procedure illustrates how to migrate an already registered system:

*Procedure: Migrating Registered Systems*

1. Setup the client using the `mgr-ssh-push-init` script (without `--register`).
2. Change the client's contact method to `ssh-push` or `ssh-push-tunnel` respectively (via API or Web UI).

Existing activation keys can also be edited via API to use the Push via SSH contact method for clients registered with these keys:

```
client.activationkey.setDetails(key, '1-mykey', {'contact_method' : 'ssh-push'})
```

## Proxy Support with Push via SSH

It is possible to use Push via SSH to manage systems that are connected to the Uyuni server via a proxy. To register a system, run `mgr-ssh-push-init` on the proxy system for each client you wish to register. Update your proxy with the latest packages to ensure the registration tool is available. It is necessary to copy the ssh key to your proxy. This can be achieved by executing the following command from the server:

```
{prompt.root}mgr-ssh-push-init --client`proxy`
```

# Push via Salt SSH

Push via Salt SSH is intended to be used in environments where your Salt clients cannot reach the Uyuni server directly to regularly checking in and, for example, fetch package updates.

*Push via SSH*

This feature is not related to Push via SSH for the traditional clients. For Push via SSH, see Push via SSH.

## Overview

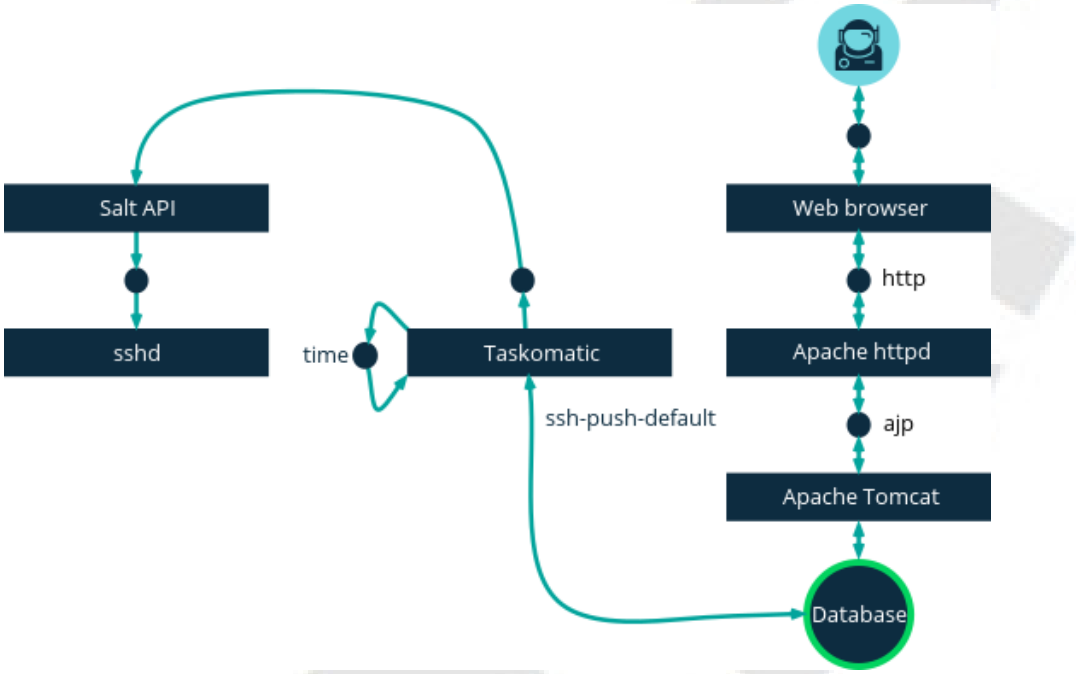



*Figure 6. Push via Salt SSH Contact Method*

Salt provides “Salt SSH” (`salt-ssh`), a feature to manage clients from a server. It works without installing Salt related software on clients. Using Salt SSH there is no need to have minions connected to the Salt master. Using this as a Uyuni connect method, this feature provides similar functionality for Salt clients as the traditional Push via SSH feature for traditional clients.

This feature allows:

- Managing Salt entitled systems with the Push via SSH contact method using Salt SSH.
- Bootstrapping such systems.

## Requirements

- SSH daemon must be running on the remote system and reachable by the `salt-api` daemon (typically running on the Uyuni server).
- Python must be available on the remote system (Python must be supported by the installed Salt).

Currently: python 2.6.

*Unsupported Systems*

Red Hat Enterprise Linux and CentOS versions ⇐ 5 are not supported because they do not have Python 2.6 by default.

## Bootstrapping

To bootstrap a Salt SSH system, proceed as follows:

1. Open the **Bootstrap Minions › ] dialog in the Web UI (menu:Systems[Bootstrapping** ).
2. Fill out the required fields. Select an **Activation Key › ] with the menu:Push via SSH[** contact method configured. For more information about activation keys, see [ref.webui.systems.activ-keys].
3. Check the **Manage system completely via SSH** option.
4. Confirm with clicking the **Bootstrap** button.

Now the system will be bootstrapped and registered in Uyuni. If done successfully, it will appear in the **Systems** list.

## Configuration

There are two kinds of parameters for Push via Salt SSH:

- Bootstrap-time parameters - configured in the **Bootstrapping** page:
  - Host
  - Activation key
  - Password - used only for bootstrapping, not saved anywhere; all future SSH sessions are authorized via a key/certificate pair
- Persistent parameters - configured Uyuni-wide:
  - sudo user - same as in Using sudo with Push via SSH.

## Action Execution

The Push via Salt SSH feature uses a taskomatic job to execute scheduled actions using `salt-ssh`. The taskomatic job periodically checks for scheduled actions and executes them. While on traditional clients with SSH push configured only `rhn_check` is executed via SSH, the Salt SSH push job executes a complete `salt-ssh` call based on the scheduled action.

## Known Limitation

- OpenSCAP auditing is not available on Salt SSH minions.
- Beacons do not work with Salt SSH.

- Installing a package on a system using `zypper` will not invoke the package refresh.
- Virtual Host functions (for example, a host to guests) will not work if the virtual host system is Salt SSH-based.

### For More Information

For more information, see

- https://wiki.microfocus.com/index.php/SUSE_Manager/SaltSSHServerPush
- https://docs.saltstack.com/en/latest/topics/ssh/

## OSAD

OSAD is an alternative contact method between Uyuni and its clients. By default, Uyuni uses `rhnsd`, which contacts the server every four hours to execute scheduled actions. OSAD allows registered client systems to execute scheduled actions immediately.

OSAD has several distinct components:

- The `osa-dispatcher` service runs on the server, and uses database checks to determine if clients need to be pinged, or if actions need to be executed.
- The `osad` service runs on the client. It responds to pings from `osa-dispatcher` and runs `mgr_check` to execute actions when directed to do so.
- The `jabberd` service is a daemon that uses the `XMPP` protocol for communication between the client and the server. The `jabberd` service also handles authentication.
- The `mgr_check` tool runs on the client to execute actions. It is triggered by communication from the `osa-dispatcher` service.

The `osa-dispatcher` periodically runs a query to check when clients last showed network activity. If it finds a client that has not shown activity recently, it will use `jabberd` to ping all `osad` instances running on all clients registered with your Uyuni server. The `osad` instances respond to the ping using `jabberd`, which is running in the background on the server. When the `osa-dispatcher` receives the response, it marks the client as online. If the `osa-dispatcher` fails to receive a response within a certain period of time, it marks the client as offline.

When you schedule actions on an OSAD-enabled system, the task will be carried out immediately. The `osa-dispatcher` periodically checks clients for actions that need to be executed. If an outstanding action is found, it uses `jabberd` to execute `mgr_check` on the client, which will then execute the action.

### Enabling and Configuring OSAD

This section covers enabling the `osa-dispatcher` and `osad` services, and performing initial setup.

OSAD clients use the fully qualified domain name (FQDN) of the server to communicate with the `osa-dispatcher` service.

SSL is required for `osad` communication. If SSL certificates are not available, the daemon on your client systems will fail to connect. Make sure your firewall rules are set to allow the required ports. For more information, see [tab.install.ports.server].

*Procedure: Enabling OSAD*

1. On your Uyuni server, as the root user, start the `osa-dispatcher` service:

```
systemctl start osa-dispatcher
```

2. On each client machine, install the `osad` package from the `Tools` child channel. The `osad` package should be installed on clients only. If you install the `osad` package on your Uyuni Server, it will conflict with the `osa-dispatcher` package.
3. On the client systems, as the root user, start the `osad` service:

```
systemctl start osad
```

   Because `osad` and `osa-dispatcher` are run as services, you can use standard commands to manage them, including `stop`, `restart`, and `status`.

*Configuration and Log Files*

Each OSAD component is configured by local configuration files. We recommend you keep the default configuration parameters for all OSAD components.

| Component | Location | Path to Configuration File |
|---|---|---|
| `osa-dispatcher` | Server | `/etc/rhn/rhn.conf` Section: `OSA configuration` |
| `osad` | Client | `/etc/sysconfig/rhn/osad.conf` |
| `osad` log file | Client | `/var/log/osad` |
| `jabberd` log file | Both | `/var/log/messages` |

*Troubleshooting OSAD*

If your OSAD clients cannot connect to the server, or if the `jabberd` service takes a lot of time responding to port 5552, it could be because you have exceeded the open file count.

Every client needs one always-open TCP connection to the server, which consumes a single file handler. If the number of file handlers currently open exceeds the maximum number of files that `jabberd` is allowed to use, `jabberd` will queue the requests, and refuse connections.

To resolve this issue, you can increase the file limits for `jabberd` by editing the `/etc/security/limits.conf` configuration file and adding these lines:

```
jabbersoftnofile5100
jabberhardnofile6000
```

Calculate the limits required for your environment by adding 100 to the number of clients for the soft limit, and 1000 to the current number of clients for the soft limit. In the example above, we have assumed 500 current clients, so the soft limit is 5100, and the hard limit is 6000.

You will also need to update the `max_fds` parameter in the `/etc/jabberd/c2s.xml` file with your chosen hard limit:

```
<max_fds>6000</max_fds>
```

# Manual Registration

## Client Configuration Overview

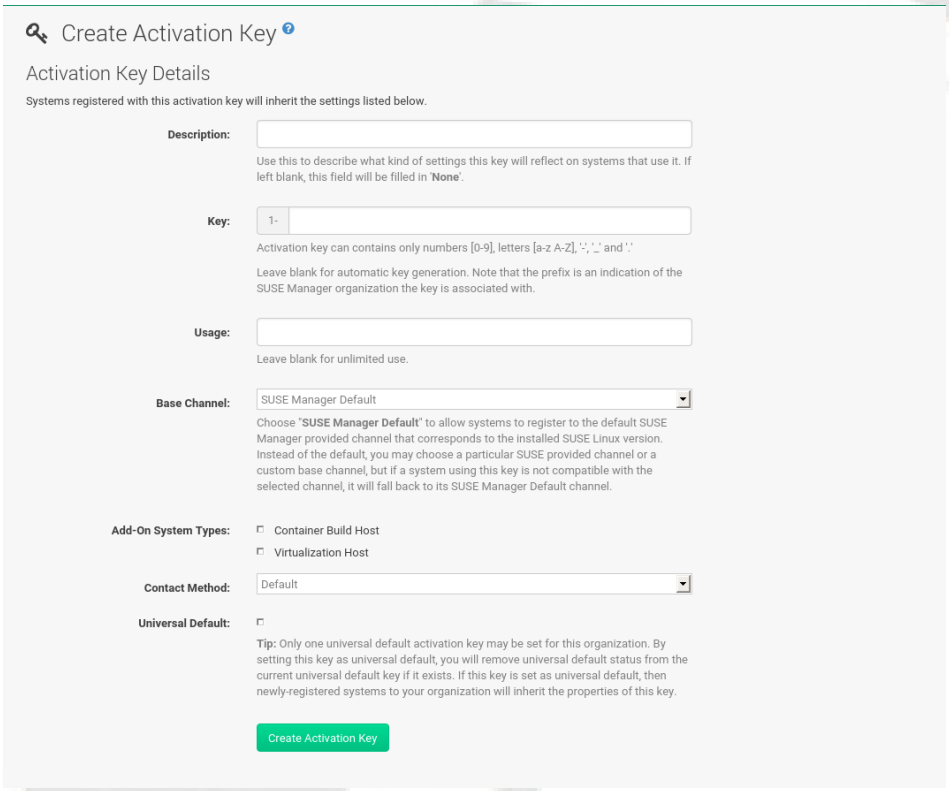
Coming Soon…

## Creating Activation keys

This section contains information on how to create activation keys for both traditional and Salt clients, and provides some best practices for working with activation keys.

*Procedure: Creating Activation Keys*

1. As the administrator login to the Uyuni Web UI.
2. Navigate to **Systems › Activation Keys**.
3. To open the `Activation Key Details` page click the **[ Create Key ]** button in the upper right corner.

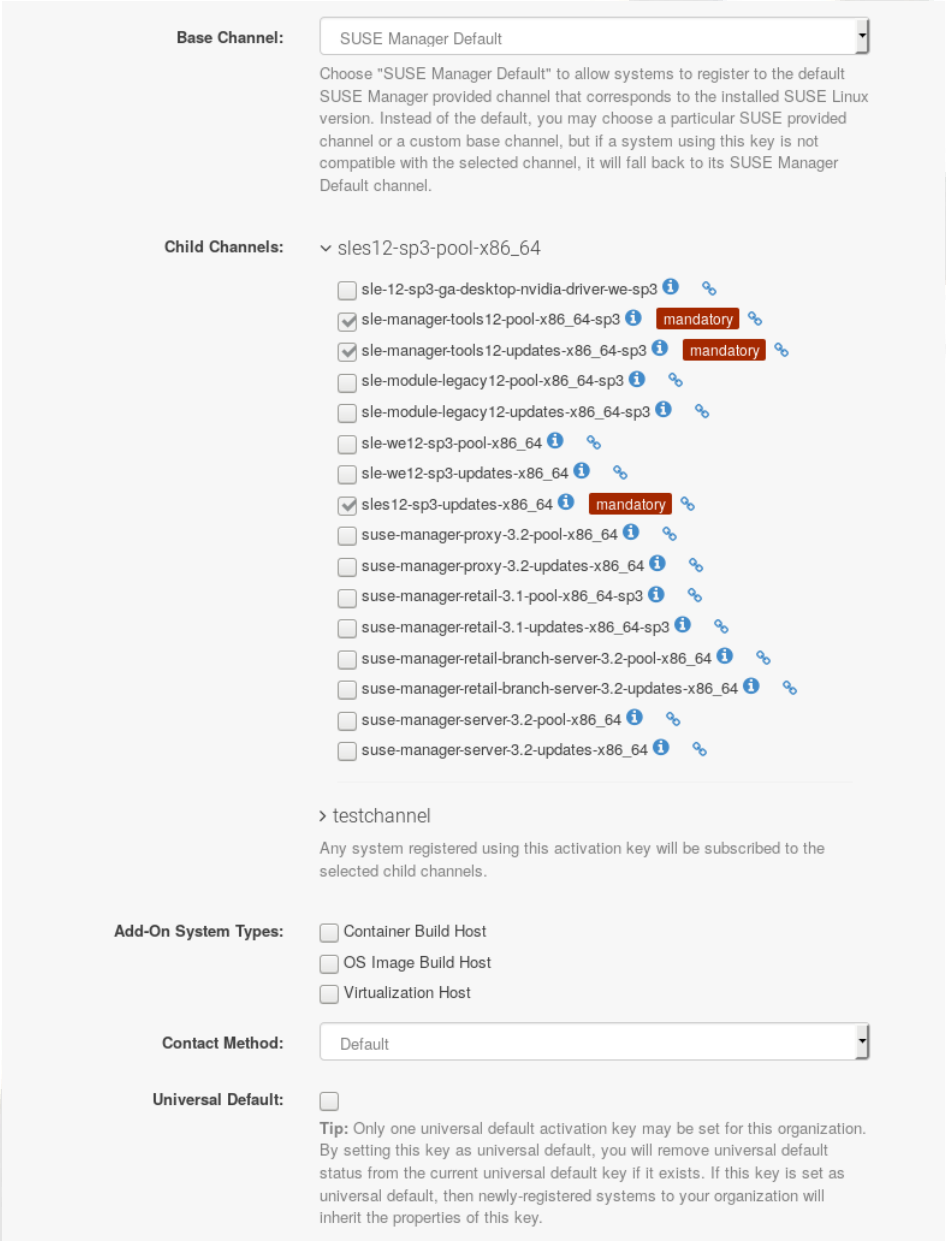



4. On the `Activation Key Details` page in the `Description` field, enter a name for the activation key.
5. In the `Key` field, enter the distribution and service pack associated with the key. For example, `SLES12-SP4` for SUSE Linux Enterprise Server 12 SP4.

*Allowed Characters*

Do not use commas in the `Key` field for any SUSE products. However, you **must** use commas for Red Hat Products. For more information, see xref:FILENAME.adoc#ref.webui.systems.activ-keys[].

6. In the `Base Channels` drop-down box, select the SUSE Linux Enterprise channel that you added during xref:FILENAME.adoc#gs-syncing-with-scc[].
7. When the base channel is selected the list of available child channels will get fetched and displayed in real time below the base channel. Select the child channels you need (for example, the SUSE Manager tools and the updates channels that are actually mandatory).




8. We recommend you leave the `Contact Method` set to `Default`.
9. We recommend you leave the `Universal Default` setting unchecked.

10. Click **[ Update Activation Key ]** to create the activation key.
11. Check the `Configuration File Deployment` check box to enable configuration management for this key, and click **[ Update Activation Key ]** to save this change.

When you create activation keys, keep these best practices in mind:

- Avoid using the `SUSE Manager Default` parent channel. This setting forces Uyuni to choose a parent channel that best corresponds to the installed operating system, which can sometimes lead to unexpected behavior. Instead, we recommend you create activation keys specific to each distribution and architecture.
- If you are using bootstrap scripts, consider creating an activation key for each script. This will help you align channel assignments, package installation, system group memberships, and configuration channel assignments. You will also need less manual interaction with your system after registration.
- If you do not enter a human-readable name for your activation keys, the system will automatically generate a number string, which can make it difficult to manage your keys. Consider a naming scheme for your activation keys to help you keep track of them.
- Note that the `Configuration File Deployment` check box does not appear until after you have created the activation key. Ensure you go back and check the box if you need to enable configuration management.

## Creating the Uyuni Tools Repository

In this section you will create a tools repository on the Uyuni Server for providing client tools. The client tools repository contains packages for installing Salt on minions as well as required packages for registering traditional clients during the bootstrapping procedure. These packages will be installed from the newly generated repository during the registration process. In the following procedure you will create the SUSE Linux Enterprise tools repository.

*Creating a Tools Repository when an SCC Channel has not been Synced*

Before following the procedure to create the tools repository make sure the SUSE vendor channel you will be using with your client has been completely synced. You can check this by running `tail -f /var/log/rhn/reposync/`<CHANNEL_NAME>`.log` as *root*. In the following example replace `version` with the actual version string:

```
# tail -f /var/log/rhn/reposync/sles`version`-pool-x86_64.log
```

Once completed you should see the following output in your terminal:

```
2017/12/12 15:20:32 +02:00 Importing packages started.
2017/12/12 15:22:02 +02:00 1.07 %
...
2017/12/12 15:34:25 +02:00 86.01 %
2017/12/12 15:35:49 +02:00 Importing packages finished.
2017/12/12 15:35:49 +02:00 Linking packages to channel.
...
2017/12/12 15:35:59 +02:00 Sync completed.
```

*Procedure: Generating the Tools Repository for SUSE Linux Enterprise*

1. Open a terminal on the server as root and enter the following command to list available bootstrap repositories:

```
mgr-create-bootstrap-repo -l SLE-`version`-x86_64
```

2. Then invoke the same command using the listed repository as the product label to actually create the bootstrap repository:

```
mgr-create-bootstrap-repo -c SLE-`version`-x86_64
```

3. Uyuni will create and add the client tools to the newly created `repositories` directory located at `/srv/www/htdocs/pub/repositories/`.

This repository is suitable for both Server and Desktop of SUSE Linux Enterprise.

*Support for SUSE Linux Enterprise 15 Products*

If you have mirrored more than one SUSE Linux Enterprise 15 Product (for example, SLES and SLES for SAP Application), you can specify the one you are actually interested in. First check what is avaiable:

```
mgr-create-bootstrap-repo -c SLE-15-x86_64 --with-custom-channel
Multiple options for parent channel found. Please use option
--with-parent-channel <label> and choose one of:
- sle-product-sles15-pool-x86_64
- sle-product-sles_sap15-pool-x86_64
- sle-product-sled15-pool-x86_64
```

Then specify it with `--with-parent-channel`:

```
mgr-create-bootstrap-repo -c SLE-15-x86_64 --with-parent-channel sle-
product-sled15-pool-x86_64
```

# Registering Traditional Clients

## Generating a Bootstrap Script

To register traditional clients, you need to create a template bootstrap script, which can be copied and modified. The bootstrap script you create is executed on the traditional client when it is registered, and ensures all the necessary packages are deployed to the client. There are some parameters contained in the bootstrap script which ensure the client system can be assigned to its base channel, including activation keys, and GPG keys.

It is important that you check the repository information carefully, to ensure it matches the base channel repository. If the repository information does not match exactly, the bootstrap script will not be able to download the correct packages.

*GPG Keys and Uyuni Client Tools*

The GPG key used by Uyuni Client Tools is not trusted by default. When you create your bootstrap script, add a path to the file containing the public key fingerprint with the `ORG_GPG_KEY` parameter.

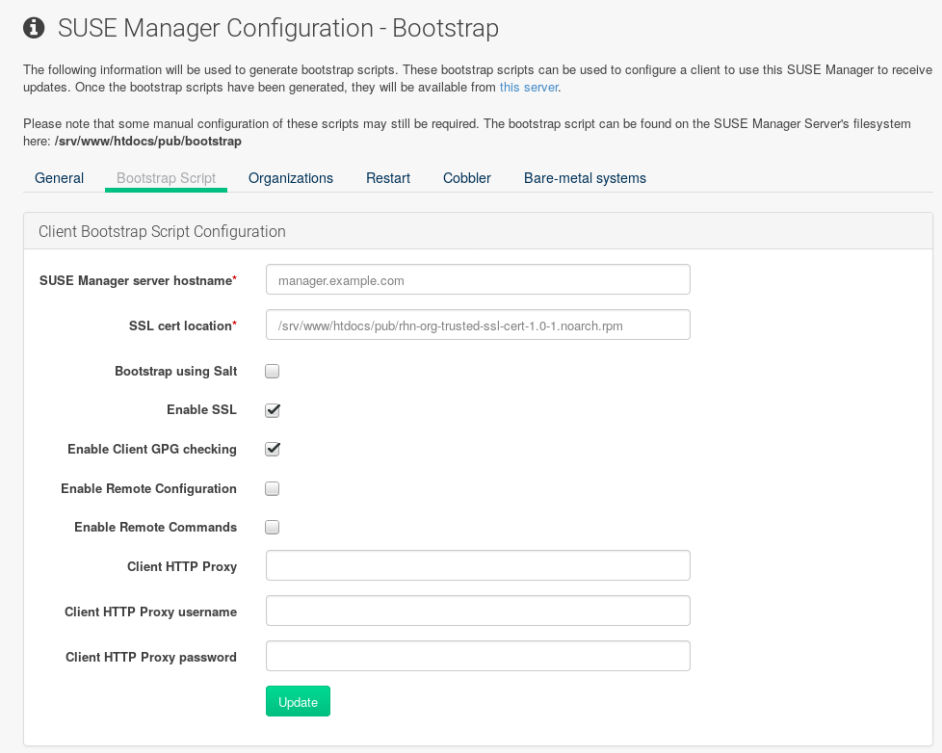
*SLES 15 and Python 3*

SLES 15 uses Python 3 by default. Bootstrap scripts based on Python 2 must be re-created for SLES 15 systems. Attempting to register SLES 15 systems with SUSE Manager using Python 2 bootstrap scripts will fail.

This procedure describes how to generate a bootstrap script.

*Procedure: Creating a Bootstrap Script*

1. From the Uyuni Web UI, browse to **Main Menu › Admin › Manager Configuration › Bootstrap Script**. For more information, see xref:FILENAME.adoc#s3-sattools-config-bootstrap[].
2. In the `SUSE Manager Configuration - Bootstrap` dialog disable `Bootstrap using Salt`. Use default settings and click the **[ Update ]** button.




*Using SSL*

Unchecking **Enable SSL** in the Web UI or setting `USING_SSL=0` in the bootstrap script is not recommended. If you disable SSL nevertheless you will need to manage custom CA certificates to be able to run the registration process successfully.

3. A template bootstrap script is generated and stored on the server's file system in the `/srv/www/htdocs/pub/bootstrap` directory.

```
cd /srv/www/htdocs/pub/bootstrap
```

The bootstrap script is also available at `https://example.com/pub/bootstrap/bootstrap.sh`.

## Editing the Bootstrap Script

In this section you will copy and modify the template bootstrap script you created from xref:FILENAME.adoc#generate.bootstrap.script[].

A minimal requirement when modifying a bootstrap script for use with Uyuni is the inclusion of an activation key. Depending on your organizations security requirements it is strongly recommended to include one or more (GPG) keys (for example, your organization key, and package signing keys). For this tutorial you will be registering with the activation keys created in the previous section.

*Procedure: Modifying the Bootstrap Script*

1. Login as root from the command line on your Uyuni server.
2. Navigate to the bootstrap directory with:

```
cd /srv/www/htdocs/pub/bootstrap/
```

3. Create and rename two copies of the template bootstrap script for use with each of your clients.

```
cp bootstrap.sh bootstrap-sles11.sh
cp bootstrap.sh bootstrap-sles12.sh
```

4. Open `sles12.sh` for modification. Scroll down and modify both lines marked in green. You must comment out `exit 1` with a hash mark (`#`) to activate the script and then enter the name of the key for this script in the `ACTIVATION_KEYS=` field as follows:

```
echo "Enable this script: comment (with #'s) this block (or, at least just"
echo "the exit below)"
echo
#exit 1

# can be edited, but probably correct (unless created during initial install):
# NOTE: ACTIVATION_KEYS *must* be used to bootstrap a client machine.
ACTIVATION_KEYS=1-sles12
ORG_GPG_KEY=
```

5. Once you have completed your modifications save the file and repeat this procedure for the second bootstrap script. Proceed to xref:FILENAME.adoc#connect.first.client[].

*Finding Your Keys*

To find key names you have created, navigate to **Home › Overview › Manage Activation keys › Key Field** in the Web UI. All keys created for channels are listed on this page. You must enter the full name of the key you wish to use in the bootstrap script exactly as presented in the key field.

## Connecting Clients

This section covers connecting your clients to Uyuni with the modified bootstrap script.

*Procedure: Running the Bootstrap Script*

1. From your Uyuni Server command line as root navigate to the following directory:

```
cd /srv/www/htdocs/pub/bootstrap/
```

2. Run the following command to execute the bootstrap script on the client:

```
cat MODIFIED-SCRIPT.SH | ssh root@example.com /bin/bash
```

3. The script will execute and proceed to download the required dependencies located in the repositories directory you created earlier. Once the script has finished running, log in to the Web UI and click **Systems › Overview** to see the new client listed.

## Package Locks

Package locks are used to prevent unauthorized installation or upgrades to software packages on traditional clients. When a package has been locked, it will show a padlock icon, indicating that it can not be installed. Any attempt to install a locked package will be reported as an error in the event log.

Locked packages can not be installed, upgraded, or removed, either through the Uyuni Web UI, or directly on the client machine using a package manager. Locked packages will also indirectly lock any dependent packages.

Package locks can only be used on traditional clients that use the Zypper package manager. The feature is not currently supported on Red Hat Enterprise Linux or Salt clients.

*Procedure: Using Package Locks*

1. On the client machine, install the `zypp-plugin-spacewalk` package:

```
# zypper in zypp-plugin-spacewalk
```

2. Navigate to the **Software › Packages › Lock** tab on the managed system to see a list of all available packages.
3. Select the packages to lock, and click **[ Request Lock ]**. You can also choose to enter a date and time for the lock to activate. Leave the date and time blank if you want the lock to activate as soon as possible. Note that the lock might not activate immediately.
4. To remove a package lock, select the packages to unlock and click **[ Request Unlock ]**. Leave the date and time blank if you want the lock to deactivate as soon as possible. Note that the lock might not deactivate immediately.

# Registering Salt Clients

There are three possible methods for registering Salt minions. You can use a bootstrap repository, a bootstrap script, or install using the Web UI.

This section describes using a bootstrap repository. Registering Salt clients with a bootstrap client is the same as registering traditional clients, which is described at Registering Traditional Clients. When using this method, ensure you enable the `Bootstrap using Salt` and activation key options in `Configuration File Deployment`, so that highstate is applied automatically. For information on using the Web UI, see Bootstrapping Salt.

You can also use these methods to change existing traditional clients into Salt minions.

*GPG Keys and Uyuni Client Tools*

The GPG key used by Uyuni Client Tools is not trusted by default. Either update your bootstrap repository to trust the key explicitly, or use the Web UI to manually trust the key from each client.

To register Salt clients with a bootstrap repository, you will need to have already set up a SUSE Manager tools repository, which is described in Create Tools Repository. You will also need to have fully synchronized a base channel for clients to obtain software packages (for example: `SLES12-SP4-Pool_for_x86_64`).

*Procedure: Registering Salt Minions*

1. On your minion as root enter the following command:

```
zypper ar http://FQDN.server.example.com/pub/repositories/sle/12/4/bootstrap/ \
   sles12-sp4
```

Do not use `HTTPS`. Use `HTTP` instead to avoid errors.

2. After adding the repository containing the necessary Salt packages execute:

```
zypper in salt-minion
```

3. Modify the minion configuration file to point to the fully qualified domain name (`FQDN`) of the Uyuni server (master):

```
vi /etc/salt/minion
```

Find and change the line:

```
master: salt
```

to:

```
master: FQDN.server.example.com
```

4. Restart the Salt minion with:

```
systemctl restart salt-minion
```

Your newly registered minion should now show up within the Web UI under **Salt › Keys**. Accept the `pending` key to begin management.

If you have used your hypervisor clone utility, and attempted to register the cloned Salt client, you might get this error:

```
We're sorry, but the system could not be found.
```

This is caused by the new, cloned, system having the same machine ID as an existing, registered, system. You can adjust this manually to correct the error and register the cloned system successfully.

For more information and instructions, see xref:FILENAME.adoc#bp.chapt.suma3.troubleshooting.registering.cloned.salt.systems[].

# Non-SUSE Clients

## Managing Red Hat Enterprise Linux Clients

The following sections provide guidance on managing Red Hat Expanded Support clients, this includes Salt minions and traditional systems.

### Server Configuration for Red Hat Enterprise Linux Channels

This section provides guidance on server configuration for Red Hat Enterprise Linux Channels provided by SUSE.

- Minimum of 8 GB RAM and at least two physical or virtual CPUs. Taskomatic will use one of these CPUs.
- Taskomatic requires of minimum of 3072 MB RAM. This should be set in `/etc/rhn/rhn.conf`:

```
taskomatic.java.maxmemory=3072
```

- Provision enough disk space. `/var/spacewalk` contains all mirrored RPMs. For example, Red Hat Enterprise Linux 6 x86_64 channels require 90 GB and more.
- LVM or an NFS mount is recommended.
- Access to RHEL 5/6/7 Subscription Media.

*Access to RHEL Media or Repositories*

Access to Red Hat base media repositories and RHEL installation media is the responsibility of the user. Ensure that all your RHEL systems obtain support from RHEL or all your RHEL systems obtain support from SUSE. If you do not follow these practices you may violate terms with Red Hat.

### Red Hat Enterprise Linux Channel Management Tips

This section provides tips on Red Hat Enterprise Linux channel management.

- The base parent distribution Red Hat Enterprise Linux channel per architecture contains zero packages. No base media is provided by SUSE. The RHEL media or installation ISOs should be added as child channels of the Red Hat Enterprise Linux parent channel.
- The Red Hat Enterprise Linux and tools channels are provided by SUSE Customer Center (SCC) using `mgr-sync`.
- It can take up to 24 hours for an initial channel synchronization to complete.
- When you have completed the initial synchronization process of any Red Hat Enterprise Linux channel it is recommended to clone the channel before working with it. This provides you with a backup of the original synchronization.

## Mirroring RHEL Media into a Channel

The following procedure guides you through setup of the RHEL media as a Uyuni channel. All packages on the RHEL media will be mirrored into a child channel located under RES 5/6/7 distribution per architecture.

*Procedure: Mirroring RHEL Media into a Channel*

1. Create a new Channel by log in to the Web UI and selecting **Channels › Manage Software Channels › Create Channel** .
2. Fill in basic channel details and add the channel as a child to the corresponding RES 5/6/7 distribution channel per architecture from SCC. The base parent channel should contain zero packages.
3. Modify the RES 5/6/7 activation key to include this new child channel.
4. As root on the Uyuni command line copy the ISO to the `/tmp` directory.
5. Create a directory to contain the media content:

```
{prompt.root}mkdir -p /srv/www/htdocs/pub/rhel
```

6. Mount the ISO:

```
{prompt.root}mount -o loop /tmp/name_of_iso /srv/www/htdocs/pub/rhel
```

7. Start `spacewalk-repo-sync` to synchronize Red Hat Enterprise Linux 7 packages:

```
{prompt.root}spacewalk-repo-sync -c channel_name -u https://127.0.0.1/pub/rhel/
Repo URL: https://127.0.0.1/pub/rhel/
Packages in repo:              [...]
Packages already synced:       [...]
Packages to sync:              [...]
[...]
```

   To synchronize RES 5/6 packages:

```
{prompt.root}spacewalk-repo-sync -c channel_name -u https://127.0.0.1/pub/rhel/Server/
Repo URL: https://127.0.0.1/pub/rhel/Server/
Packages in repo:              [...]
Packages already synced:       [...]
Packages to sync:              [...]
[...]
```

8. When the channel has completed the synchronization process you can use the channel as any normal Uyuni channel.

Attempting to synchronize the repository will sometimes fail with this error:

```
[Errno 256] No more mirrors to try.
```

To troubleshoot this error, look at the HTTP protocol to determine if `spacewalk-repo-sync` is running:

*procedure: Debug spacewalk-repo-sync*

1. Start debugging mode with `export URLGRABBER_DEBUG=DEBUG`
2. Check the output of `/usr/bin/spacewalk-repo-sync --channel <channel-label> --type yum`
3. If you want to disable debug mode, use `unset URLGRABBER_DEBUG`

## Registering RES Salt Minions with Uyuni

This section will guide you through registering RHEL minions with Uyuni.

This section assumes you have updated your server to the latest patch level.

### Synchronizing Appropriate Red Hat Enterprise Linux Channels

Ensure you have the corresponding Red Hat Enterprise Linux product enabled and required channels have been fully synchronized:

*RHEL 7.x*

- Product: Red Hat Enterprise Linux 7
- Mandatory channels: `rhel-x86_64-server-7` , `res7-suse-manager-tools-x86_64` , `res7-x86_64` systemitem>

*RHEL 6.x*

- Product: Red Hat Enterprise Linux 6
- Mandatory channels: `rhel-x86_64-server-6` , `res6-suse-manager-tools-x86_64` , `res6-x86_64`

*Checking Synchronization Progress*

To check if a channel has finished synchronizing you can do one of the following:

- From the UyuniWeb UI browse to **Admin** › **Setup Wizard** and select the **SUSE Products** tab. Here you will find a percent completion bar for each product.
- Alternatively, you may check the synchronization log file located under `/var/log/rhn/reposync/channel-label.log` using cat or the tailf command. Keep in mind that base channels can contain multiple child channels. Each of these child channels will generate its own log during the synchronization progress. Do not assume a channel has finished synchronizing until you have checked all relevant log files including base and child channels.

Create an activation key associated with the Red Hat Enterprise Linux channel.

### Creating a Bootstrap Repository

The following procedure demonstrate creating a bootstrap repository for RHEL:

1. On the server command line as root, create a bootstrap repo for RHEL with the following command:

```
mgr-create-bootstrap-repo RHEL_activation_channel_key
```

   If you use a dedicated channel per RHEL version, specify it with the `--with-custom-channel` option.

2. Rename `bootstrap.sh` to `resversion-boostrap.sh`:

```
{prompt.root}cp bootstrap.sh res7-bootstrap.sh
```

## Register a Salt Minion via Bootstrap

The following procedure will guide you through registering a Salt minion using the bootstrap script.

*Procedure: Registration Using the Bootstrap Script*

1. For your new minion download the bootstrap script from the Uyuni server:

```
wget --no-check-certificate https://`server`/pub/bootstrap/res7-bootstrap.sh
```

2. Add the appropriate res-gpg-pubkey--.key to the `ORG_GPG_KEY` key parameter, comma delimited in your `res7-bootstrap.sh` script. These are located on your Uyuni server at:

```
http://`server`/pub/
```

3. Make the `res7-bootstrap.sh` script executable and run it. This will install necessary Salt packages from the bootstrap repository and start the Salt minion service:

```
{prompt.root}chmod +x res7-bootstrap.sh{prompt.root}./res7-boostrap.sh
```

4. From the Uyuni Web UI select **Salt › Keys** and accept the new minion's key.

*Troubleshooting Bootstrap*

If bootstrapping a minion fails it is usually caused by missing packages. These missing packages are contained on the RHEL installation media. The RHEL installation media should be loop mounted and added as a child channel to the Red Hat Enterprise Linux channel. See the warning in [bp.expanded-support.resclients] on access to RHEL Media.

## Manual Salt Minion Registration

The following procedure will guide you through the registration of a Salt minion manually.

1. Add the bootstrap repository:

```
yum-config-manager --add-repo https://`server`/pub/repositories/res/7/bootstrap
```

2. Install the salt-minion package:

```
{prompt.root}yum install salt-minion
```

3. Edit the Salt minion configuration file to point to the Uyuni server:

```
{prompt.root}mkdir /etc/salt/minion.d{prompt.root}echo "master:`server_fqdn`" >
/etc/salt/minion.d/susemanager.conf
```

4. Start the minion service:

```
{prompt.root}systemctl start salt-minion
```

5. From the Uyuni Web UI select the **Salt › Keys** and accept the new minion's key.

# Preparing Channels and Repositories for CentOS Traditional Clients

This following section provides an example procedure for configuring CentOS channels and repositories and finally registering a CentOS client with Uyuni.

These steps will be identical for Scientific Linux and Fedora.

*Procedure: Preparing Channels and Repositories*

1. As root install spacewalk-utils on your Uyuni server:

```
zypper in spacewalk-utils
```

*Supported Tools*

The spacewalk-utils package contains a collection of upstream command line tools which provide assistance with spacewalk administrative operations. You will be using the `spacewalk-common-channels` tool. Keep in mind SUSE only provides support for `spacewalk-clone-by-date` and `spacewalk-manage-channel-lifecycle` tools.

2. Run the `spacewalk-common-channels` script to add the CentOS7 base, updates, and Spacewalk client channels.

```
{prompt.root}spacewalk-common-channels -u admin -p`secret`-a x86_64
'centos7'{prompt.root}spacewalk-common-channels -u admin -p`secret`-a x86_64 'centos7-
updates'{prompt.root}spacewalk-common-channels -u admin -p`secret`-a x86_64
'spacewalk26-client-centos7'
```

*Required Channel References*

The `/etc/rhn/spacewalk-common-channels.ini` must contain the channel references to be added. If a channel is not listed, check the latest version here for updates: https://github.com/spacewalkproject/spacewalk/tree/master/utils

3. From the Web UI select **Main Menu › Software › Manage Software Channels › Overview**. Select the base channel you want to synchronize, in this case `CentOS7 (x86_64)`. Select **Repositories › Sync**. Check the channels you want to synchronize and then click the **[ Sync Now ]** button or, optionally, schedule a regular synchronization time.
4. Copy all relevant GPG keys to `/srv/www/htdocs/pub`. Depending on what distribution you are interested in managing these could include an EPEL key, SUSE keys, Red Hat keys, and CentOS keys. After copying these you can reference them in a comma-delimited list within your bootstrap script (see Procedure: Preparing the Bootstrap Script).
    - CentOS7 key files: http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-7
    - EPEL key file: http://mirrors.kernel.org/fedora-epel/RPM-GPG-KEY-EPEL-7

- Spacewalk key: http://spacewalk.redhat.com/yum/RPM-GPG-KEY-spacewalk-2015
- Red Hat keys: http://www.redhat.com/contact/security-response-team/gpg-keys.html

5. Install and setup a CentOS 7 client with the default installation packages.
6. Ensure the client machine can resolve itself and your Uyuni server via DNS. Validate that there is an entry in `/etc/hosts` for the real IP address of the client.
7. Create an activation key (`centos7`) on the Uyuni server that points to the correct parent/child channels, including the CentOS base repo, updates, and Spacewalk client.

Now prepare the bootstrap script.

*Procedure: Preparing the Bootstrap Script*

1. Create/edit your bootstrap script to correctly reflect the following:

```
# can be edited, but probably correct (unless created during initial install):

# NOTE: ACTIVATION_KEYS *must* be used to bootstrap a client machine.

ACTIVATION_KEYS=1-centos7

ORG_GPG_KEY=res.key,RPM-GPG-KEY-CentOS-7,suse-307E3D54.key,suse-9C800ACA.key,RPM-GPG-
KEY-spacewalk-2015

FULLY_UPDATE_THIS_BOX=0

yum clean all
# Install the prerequisites
yum -y install yum-rhn-plugin rhn-setup
```

2. Add the following lines to the bottom of your script, (just before `echo "-bootstrap complete -"`):

```
# This section is for commands to be executed after registration
mv /etc/yum.repos.d/Cent* /root/
yum clean all
chkconfig rhnsd on
chkconfig osad on
service rhnsd restart
service osad restart
```

3. Continue by following normal bootstrap procedures to bootstrap the new client.

## Registering CentOS Salt Minions with Uyuni

The following procedure will guide you through registering a CentOS Minion.

*Support for CentOS Patches*

CentOS uses patches originating from CentOS is not officially supported by SUSE . See the matrix of Uyuni clients on the main page of the Uyuni wiki, linked from the *Quick Links* section: https://wiki.microfocus.com/index.php?title=SUSE_Manager

*Procedure: Register a CentOS 7 Minion*

1. Add the Open Build Service repo for Salt:

```
{prompt.root}yum-config-manager --add-repo
http://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products/RHEL_7/
```

2. Import the repo key:

```
{prompt.root}rpm --import
http://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products/RHEL_7/
repodata/repomd.xml.key
```

3. Check if there is a different repository that contains Salt. If there is more than one repository listed disable the repository that contains Salt apart from the OBS one.

```
{prompt.root}yum list --showduplicates salt
```

4. Install the Salt minion:

```
{prompt.root}yum install salt salt-minion
```

5. Change the Salt configuration to point to the Uyuni server:

```
{prompt.root}mkdir -p /etc/salt/minion.d{prompt.root}echo "master:`server_fqdn`" >
/etc/salt/minion.d/susemanager.conf
```

6. Restart the minion

```
{prompt.root}systemctl restart salt-minion
```

7. Proceed to **Main Menu › Salt › Keys** from the Web UI and accept the minion's key.

# Ubuntu Clients

## Managing Ubuntu Clients

Support for Ubuntu Clients was added in SUSE Manager 3.2. Currently, Salt minions running Ubuntu 16.04 LTS and 18.04 LTS are supported.

Ubuntu clients must be Salt minions. Traditional clients are not supported.

Bootstrapping is supported for starting Ubuntu clients and performing initial state runs such as setting repositories and performing profile updates. However, the root user on Ubuntu is disabled by default, so in order to use bootstrapping, you will require an existing user with `sudo` privileges for Python.

Other supported features:

- Synchronizing `.deb` channels
- Assigning `.deb` channels to minions
- GPG signing `.deb` repositories
- Minion cleanup on delete
- Information displayed in System details pages
- Package install, update, and remove
- Package install using `Package States`
- Configuration and state channels

Some actions are not yet supported:

- Patch and errata support
- Bare metal installations, PXE booting, and virtual host provisioning
- Live patching
- CVE Audit
- If you use are using a repository from storage media (`server.susemanager.fromdir = …` option in rhn.conf), Ubuntu Client Tools will not work.

This section describes how to manually prepare Ubuntu clients for registration to your SUSE Manager Server and synchronize the repositories.

*Procedure: Preparing an Ubuntu 16.04 Client for Registration*

1. On the client, open the `/etc/apt/sources.list.d/suma_client_tools.list` file, and add this line:

```
deb
https://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products:/debia
n/xUbuntu_16.04/ /
```

2. From the command line, import the appropriate release key and add it to the keyring:

```
curl
https://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products:/debia
n/xUbuntu_16.04/Release.key
sudo apt-key add -
```

3. Update the repository list in the package manager:

```
sudo apt update
```

4. Edit the `sudoers` file:

```
sudo visudo
```

Grant `sudo` access to the user by adding this line to the `sudoers` file. Replace `<user>` with the name of the user that will bootstrap the client in the Web UI:

```
<user>   ALL=NOPASSWD: /usr/bin/python, /usr/bin/python2, /usr/bin/python3
```

*Procedure: Preparing an Ubuntu 18.04 Client for Registration*

1. On the client, open the `/etc/apt/sources.list.d/suma_client_tools.list` file, and add this line:

```
deb
https://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products:/debia
n/xUbuntu_18.04/ /
```

2. From the command line, import the appropriate release key and add it to the keyring:

```
curl
https://download.opensuse.org/repositories/systemsmanagement:/saltstack:/products:/debia
n/xUbuntu_18.04/Release.key
sudo apt-key add -
```

3. Update the repository list in the package manager:

```
sudo apt update
```

4. Edit the `sudoers` file:

```
sudo visudo
```

Grant `sudo` access to the user by adding this line to the `sudoers` file. Replace `<user>` with the name of the user that will be used to bootsrap the client in the Web UI:

```
<user>   ALL=NOPASSWD: /usr/bin/python, /usr/bin/python2, /usr/bin/python3
```

*Procedure: Synchronizing Ubuntu Repositories*

1. In the SUSE Manager Web UI, navigate to **Software › Manage › Repositories** and click **[ Create Repository ]**.
2. In the `Create Repository` dialog, use these values to create a new repository:
   - In the `Repository URL` field, type the URL that contains the appropriate binary files, for example `http://ubuntumirror.example.com/ubuntu/dists/bionic/main/binary-amd64/`.
   - In the `Repository Type` field, select `deb`.
3. Click **[ Create Repository ]** to create the repository.
4. Navigate to **Software › Manage › Channels** and click **[ Create Channel ]**.
5. In the `Create Software Channel` dialog, create the channel as required for your environment. Ensure that in the `Architecture` field, you select `AMD64 Debian`.
6. Click **[ Create Channel ]** to create the software channel.
7. Navigate to **Software › Manage › Channels** and select your new channel from the channel list.
8. In the `Repositories` tab, click the `Add/Remove` tab, and select your new repository from the repository list.
9. Click the **[ Update Repositories ]** button to synchronize the repository.

You can check the progress of your synchronization from the command line using this command:

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

# Salt Minion Scalability

## Salt Minion Onboarding Rate

The rate at which SUSE Manager can on-board minions (accept Salt keys) is limited and depends on hardware resources. On-boarding minions at a faster rate than SUSE Manager is configured for will build up a backlog of unprocessed keys slowing the process and potentially exhausting resources. It is recommended to limit the acceptance key rate pro-grammatically. A safe starting point would be to on-board a minion every 15 seconds, which can be implemented via the following command:

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

## Minions Running with Unaccepted Salt Keys

Minions which have not been on-boarded, (minions running with unaccepted Salt keys) consume resources, in particular inbound network bandwidth for ~2.5 Kb/s per minion. 1000 idle minions will consume around ~2.5 Mb/s, and this number will drop to almost 0 once on-boarding has been completed. Limit non-onboarded systems for optimal performance.

Salt's official documentation suggests the maximum number of opened files should be set to at least 2 × the minion count. Current default is 16384, which is sufficient for ~8000 minions. For larger installations, this number may be increased by editing the following line in `/usr/lib/systemd/system/salt-master.service` :

```
LimitNOFILE=16384
```

## Salt Timeouts

### Background Information

Salt features two timeout parameters called `timeout` and `gather_job_timeout` that are relevant during the execution of Salt commands and jobs—it does not matter whether they are triggered using the command line interface or API. These two parameters are explained in the following article.

This is a normal workflow when all minions are well reachable:

- A salt command or job is executed:

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

- Salt master publishes the job with the targeted minions into the Salt PUB channel.
- Minions take that job and start working on it.
- Salt master is looking at the Salt RET channel to gather responses from the minions.
- If Salt master gets all responses from targeted minions, then everything is completed and Salt master will return a response containing all the minion responses.

If some of the minions are down during this process, the workflow continues as follows:

1. If `timeout` is reached before getting all expected responses from the minions, then Salt master would trigger an aditional job (a Salt `find_job` job) targeting only pending minions to check whether the job is already running on the minion.
2. Now `gather_job_timeout` is evaluated. A new counter is now triggered.
3. If this new `find_job` job responses that the original job is actually running on the minion, then Salt master will wait for that minion's response.

4. In case of reaching `gather_job_timeout` without having any response from the minion (neither for the initial `test.ping` nor for the `find_job` job), Salt master will return with only the gathered responses from the responding minions.

By default, Uyuni globally sets `timeout` and `gather_job_timeout` to 120 seconds. So, in the worst case, a Salt call targeting unreachable minions will end up *with 240 seconds of waiting* until getting a response.

### A Presence Ping Mechanism for Unreachable Salt Minions

In order to prevent waiting until timeouts are reached when some minions are down, SUSE introduced a so-called "presence mechanism" for Salt minions.

This presence mechanism checks for unreachable Salt minions when Uyuni is performing synchronous calls to these minions, and it excludes unreachable minions from that call. Synchronous calls are going to be displaced in favor of asynchronous calls but currently still being used during some workflows.

The presence mechanism triggers a Salt `test.ping` with a custom and fixed short Salt timeout values. Default Salt values for the presence ping are: `timeout = 4` and `gather_job_timeout = 1`. This way, we can quickly detect which targeted minions are unreachable, and then exclude them from the synchronous call.

### Overriding Salt Presence Timeout Values

Uyuni administrators can increase or decrease default presence ping timeout values by removing the comment markers (`\#`) and setting the desired values for `salt_presence_ping_timeout` and `salt_presence_ping_gather_job_timeout` options in `/etc/rhn/rhn.conf`:

```
# SUSE Manager presence timeouts for Salt minions
# salt_presence_ping_timeout = 4
# salt_presence_ping_gather_job_timeout = 1
```

### Salt SSH Minions (SSH Push)

Salt SSH minions are slightly different that regular minions (zeromq). Salt SSH minions do not use Salt PUB/RET channels but a wrapper Salt command inside of an SSH call. Salt `timeout` and `gather_job_timeout` are not playing a role here.

Uyuni defines a timeout for SSH connections in `/etc/rhn/rhn.conf`:

```
# salt_ssh_connect_timeout = 180
```

The presence ping mechanism is also working with SSH minions. In this case, Uyuni will use `salt_presence_ping_timeout` to override the default timeout value for SSH connections.

# Automating Installation

## Introduction

*Autoinstallation Types: AutoYaST and Kickstart*

In the following section, AutoYaST and AutoYaST features apply for SUSE Linux Enterprise client systems only.

For RHEL systems, use Kickstart and Kickstart features.

*Auto-Installing Salt Minions Currently Not Supported*

This procedure will work for traditionally managed systems (system type `management`).

Autoinstallation is not currently available for systems using Salt (system type `salt`).

AutoYaST and Kickstart configuration files allow administrators to create an environment for automating otherwise time-consuming system installations, such as multiple servers or workstations. AutoYaST files have to be uploaded to be managed with Uyuni. Kickstart files can be created, modified, and managed within the Uyuni Web interface.

Uyuni also features the Cobbler installation server. For more information on Cobbler, see:

xref:FILENAME.adoc#advanced.topics.cobbler[].

## AutoYaST

Using AutoYaST, a system administrator can create a single file containing the answers to all the questions that would normally be asked during a typical installation of a SUSE Linux Enterprise system.

AutoYaST files can be kept on a single server system and read by individual computers during the installation. This way the same AutoYaST file is used to install SUSE Linux Enterprise on multiple machines.

The *SUSE Linux Enterprise Server AutoYaST Guide* at (https://www.suse.com/documentation/sles-15/) will contain an in-depth discussion of “Automated Installation” using AutoYaST.

### AutoYaST Explained

When a machine is to receive a network-based AutoYaST installation, the following events must occur in this order:

1. After being connected to the network and turned on, the machine’s PXE logic broadcasts its MAC address and requests to be discovered.

2. If no static IP address is used, the DHCP server recognizes the discovery request and offers network information needed for the new machine to boot. This includes an IP address, the default gateway to be used, the netmask of the network, the IP address of the TFTP or HTTP server holding the bootloader program, and the full path and file name to that program (relative to the server's root).
3. The machine applies the networking information and initiates a session with the server to request the bootloader program.
4. The bootloader searches for its configuration file on the server from which it was loaded. This file dictates which Kernel and Kernel options, such as the initial RAM disk (initrd) image, should be executed on the booting machine. Assuming the bootloader program is SYSLINUX, this file is located in the `pxelinux.cfg` directory on the server and named the hexadecimal equivalent of the new machine's IP address. For example, a bootloader configuration file for SUSE Linux Enterprise Server should contain:

```
port 0
prompt 0
timeout 1
default autoyast
label autoyast
  kernel vmlinuz
  append autoyast=http://`my_susemanager_server`/`path`\
    install=http://`my_susemanager_server`/`repo_tree`
```

5. The machine accepts and uncompresses the initrd and kernel, boots the kernel, fetches the instsys from the install server and initiates the AutoYaST installation with the options supplied in the bootloader configuration file, including the server containing the AutoYaST configuration file.
6. The new machine is installed based on the parameters established within the AutoYaST configuration file.

## AutoYaST Prerequisites

Some preparation is required for your infrastructure to handle AutoYaST installations. For instance, before creating AutoYaST profiles, you may consider:

- A DHCP server is not required for AutoYaST, but it can make things easier. If you are using static IP addresses, you should select static IP while developing your AutoYaST profile.
- Host the AutoYaST distribution trees via HTTP, properly provided by Uyuni.
- If conducting a so-called bare-metal AutoYaST installation, provide the following settings:
  - Configure DHCP to assign the required networking parameters and the bootloader program location.
  - In the bootloader configuration file, specify the kernel and appropriate kernel options to be used.

## Building Bootable AutoYaST ISOs

While you can schedule a registered system to be installed by AutoYaST with a new operating system and package profile, you can also automatically install a system that is not registered with Uyuni, or does not

yet have an operating system installed. One common method of doing this is to create a bootable CD-ROM that is inserted into the target system. When the system is rebooted or switched on, it boots from the CD-ROM, loads the AutoYaST configuration from your Uyuni, and proceeds to install SUSE Linux Enterprise Server according to the AutoYaST profile you have created.

To use the CD-ROM, boot the system and type `autoyast` at the prompt (assuming you left the label for the AutoYaST boot as `autoyast`). When you press `Enter`, the AutoYaST installation begins.

For more information about image creation, refer to KIWI at http://doc.opensuse.org/projects/kiwi/doc/.

### Integrating AutoYaST with PXE

In addition to CD-ROM-based installations, AutoYaST installation through a Pre-Boot Execution Environment (PXE) is supported. This is less error-prone than CDs, enables AutoYaST installation from bare metal, and integrates with existing PXE/DHCP environments.

To use this method, make sure your systems have network interface cards (NIC) that support PXE, install and configure a PXE server, ensure DHCP is running, and place the installation repository on an HTTP server for deployment. Finally upload the AutoYaST profile via the Web interface to the Uyuni server. Once the AutoYaST profile has been created, use the URL from the **Autoinstallation Overview** page, as for CD-ROM-based installations.

To obtain specific instructions for conducting PXE AutoYaST installation, refer to the *Using PXE Boot* section of the *SUSE Linux Enterprise Deployment Guide*.

Starting with xref:FILENAME.adoc#ref.webui.systems.autoinst.profiles[], AutoYaST options available from **Systems › Kickstart** are described.

## Kickstart

Using Kickstart, a system administrator can create a single file containing the answers to all the questions that would normally be asked during a typical installation of Red Hat Enterprise Linux.

Kickstart files can be kept on a single server and read by individual computers during the installation. This method allows you to use one Kickstart file to install Red Hat Enterprise Linux on multiple machines.

The *Red Hat Enterprise Linux System Administration Guide* contains an in-depth description of Kickstart (https://access.redhat.com/documentation/en/red-hat-enterprise-linux/).

### Kickstart Explained

When a machine is to receive a network-based Kickstart, the following events must occur in this order:

1. After being connected to the network and turned on, the machine's PXE logic broadcasts its MAC address and requests to be discovered.
2. If no static IP address is used, the DHCP server recognizes the discovery request and offers network information needed for the new machine to boot. This information includes an IP address, the default

gateway to be used, the netmask of the network, the IP address of the TFTP or HTTP server holding the bootloader program, and the full path and file name of that program (relative to the server's root).

3. The machine applies the networking information and initiates a session with the server to request the bootloader program.
4. The bootloader searches for its configuration file on the server from which it was loaded. This file dictates which kernel and kernel options, such as the initial RAM disk (initrd) image, should be executed on the booting machine. Assuming the bootloader program is SYSLINUX, this file is located in the `pxelinux.cfg` directory on the server and named the hexadecimal equivalent of the new machine's IP address. For example, a bootloader configuration file for Red Hat Enterprise Linux AS 2.1 should contain:

```
port 0
prompt 0
timeout 1
default My_Label
label My_Label
      kernel vmlinuz
      append ks=http://`my_susemanager_server`/`path`\
          initrd=initrd.img network apic
```

5. The machine accepts and uncompresses the init image and kernel, boots the kernel, and initiates a Kickstart installation with the options supplied in the bootloader configuration file, including the server containing the Kickstart configuration file.
6. This Kickstart configuration file in turn directs the machine to the location of the installation files.
7. The new machine is built based on the parameters established within the Kickstart configuration file.

## Kickstart Prerequisites

Some preparation is required for your infrastructure to handle Kickstarts. For instance, before creating Kickstart profiles, you may consider:

- A DHCP server is not required for kickstarting, but it can make things easier. If you are using static IP addresses, select static IP while developing your Kickstart profile.
- An FTP server can be used instead of hosting the Kickstart distribution trees via HTTP.
- If conducting a bare metal Kickstart, you should configure DHCP to assign required networking parameters and the bootloader program location. Also, specify within the bootloader configuration file the kernel to be used and appropriate kernel options.

## Building Bootable Kickstart ISOs

While you can schedule a registered system to be kickstarted to a new operating system and package profile, you can also Kickstart a system that is not registered with Uyuni or does not yet have an operating system installed. One common method of doing this is to create a bootable CD-ROM that is inserted into the target system. When the system is rebooted, it boots from the CD-ROM, loads the Kickstart configuration from your Uyuni, and proceeds to install Red Hat Enterprise Linux according to the

Kickstart profile you have created.

To do this, copy the contents of `/isolinux` from the first CD-ROM of the target distribution. Then edit the `isolinux.cfg` file to default to 'ks'. Change the 'ks' section to the following template:

```
label ks
kernel vmlinuz
  append text ks=`url`initrd=initrd.img lang= devfs=nomount \
    ramdisk_size=16438`ksdevice`
```

IP address-based Kickstart URLs will look like this:

```
http://`my.manager.server`/kickstart/ks/mode/ip_range
```

The Kickstart distribution defined via the IP range should match the distribution from which you are building, or errors will occur. `ksdevice` is optional, but looks like:

```
ksdevice=eth0
```

It is possible to change the distribution for a Kickstart profile within a family, such as Red Hat Enterprise Linux AS 4 to Red Hat Enterprise Linux ES 4, by specifying the new distribution label. Note that you cannot move between versions (4 to 5) or between updates (U1 to U2).

Next, customize `isolinux.cfg` further for your needs by adding multiple Kickstart options, different boot messages, shorter timeout periods, etc.

Next, create the ISO as described in the *Making an Installation Boot CD-ROM* section of the *Red Hat Enterprise Linux Installation Guide*. Alternatively, issue the command:

```
mkisofs -o file.iso -b isolinux.bin -c boot.cat -no-emul-boot \
  -boot-load-size 4 -boot-info-table -R -J -v -T isolinux/
```

Note that `isolinux/` is the relative path to the directory containing the modified isolinux files copied from the distribution CD, while `file.iso` is the output ISO file, which is placed into the current directory.

Burn the ISO to CD-ROM and insert the disc. Boot the system and type "ks" at the prompt (assuming you left the label for the Kickstart boot as 'ks'). When you press `Enter`, Kickstart starts running.

## Integrating Kickstart with PXE

In addition to CD-ROM-based installs, Kickstart supports a Pre-Boot Execution Environment (PXE). This is less error-prone than CDs, enables kickstarting from bare metal, and integrates with existing PXE/DHCP environments.

To use this method, make sure your systems have network interface cards (NIC) that support PXE. Install and configure a PXE server and ensure DHCP is running. Then place the appropriate files on an HTTP server for deployment. Once the Kickstart profile has been created, use the URL from the **Kickstart Details** page, as for CD-ROM-based installs.

To obtain specific instructions for conducting PXE Kickstarts, refer to the *PXE Network Installations* chapter of the *Red Hat Enterprise Linux 4 System Administration Guide*.

Running the Network Booting Tool, as described in the Red Hat Enterprise Linux 4: System Administration Guide, select "HTTP" as the protocol and include the domain name of the Uyuni in the Server field if you intend to use it to distribute the installation files.

The following sections describe the autoinstallation options available from the **Systems › Autoinstallation** page.

# Cobbler

## Introduction

Uyuni features the Cobbler server, which allows administrators to centralize system installation and provisioning infrastructure. Cobbler is an installation server that provides various methods of performing unattended system installations. It can be used on server, workstation, or guest systems, in full or para-virtualized environments.

Cobbler offers several tools for pre-installation guidance, automated installation file management, installation environment management, and more. This section explains some of the supported features of Cobbler, including:

- Installation environment analysis using the `cobbler check` command
- Multi-site installation server configuration using the `cobbler replicate` command
- Virtual machine guest installation automation with the `koan` client-side tool
- Building installation ISOs with PXE-like menus using the `cobbler buildiso` command (for Uyuni systems with x86_64 architecture)

For more detailed Cobbler documentation, see http://cobbler.github.io/manuals/.

*Supported Cobbler Functions*

SUSE only support those Cobbler functions that are either listed within our formal documentation or available via the web interface and API.

## Cobbler Requirements

To use Cobbler for system installation with PXE, you will require a TFTP server. Uyuni installs a TFTP

server by default. To PXE boot systems, you will require a DHCP server, or have access to a network DHCP server. Edit the `/etc/dhcp.conf` configuration file to change `next-server` to the hostname or IP address of your Cobbler server.

Cobbler requires an open HTTP port to synchronize data between the Server and the Proxy. By default, Cobbler uses port 80, but you can configure it to use port 443 instead if that suits your environment.

*Correct Hostname Configuration*

Cobbler uses hostnames as a unique key for each system. If you are using the `pxe-default-image` to onboard bare metal systems, make sure every system has a unique hostname. Non-unique hostnames will cause all systems with the same hostname to have the configuration filess overwritten when a provisioning profile is assigned.

## Configuring Cobbler with /etc/cobbler/settings

Cobbler configuration is primarily managed using the `/etc/cobbler/settings` file. Cobbler will run with the default settings unchanged. All configurable settings are explained in detail in the `/etc/cobbler/settings` file, including information on each setting, and recommendations.

*Supported Languages*

If Uyuni complains that language `en` was not found within the list of supported languages available at `/usr/share/YaST2/data/languages`, remove the `lang` parameter in the `/etc/cobbler/settings` file, or add a valid parameter such as `en_US`.

For more on this topic, see https://www.suse.com/support/kb/doc?id=7018334.

## Cobbler and DHCP

Cobbler uses DHCP to automate bare metal installations from a PXE boot server. You must have administrative access to the network's DHCP server, or be able to configure DHCP directly on the the Cobbler server.

### Configuring an Existing DHCP Server

If you have existing DHCP server, you will need to edit the DHCP configuration file so that it points to the Cobbler server and PXE boot image.

As root on the DHCP server, edit the `/etc/dhcpd.conf` file and append a new class with options for performing PXE boot installation. For example:

```
allow booting;
allow bootp; ①
class "PXE" ②
{match if substring(option vendor-class-identifier, 0, 9) = "PXEClient"; ③
next-server 192.168.2.1; ④
filename "pxelinux.0";} ⑤
```

① Enable network booting with the `bootp` protocol.

② Create a class called `PXE`.

③ A system configured to have PXE first in its boot priority identifies itself as `PXEClient`.

④ As a result, the DHCP server directs the system to the Cobbler server at `192.168.2.1`.

⑤ The DHCP server retrieves the `pxelinux.0`bootloader file.

#### Setting up PXE Boot in KVM

It is possible to set up PXE booting in KVM, however we do not recommend you use this method for production systems. This method can replace the `next-server` setting on a DHCP server, as described in xref:FILENAME.adoc#advanced.topics.cobbler.reqs.dhcp.notmanaged[]. Edit the network XML description with `virsh`:

1. Produce an XML dump of the current description:

```
virsh net-dumpxml --inactive network > network.xml
```

2. Open the XML dump file at `network.xml` with a text editor and add a `bootp` parameter within the `<dhcp>` element:

```
<bootp file='/pxelinux.0' server='192.168.100.153'/>
```

3. Install the updated description:

```
virsh net-define network.xml
```

Alternatively, use the `net-edit` subcommand, which will also perform some error checking.

*Example 1. Minimal Network XML Description for KVM*

```
<network>
  <name>default</name>
  <uuid>1da84185-31b5-4c8b-9ee2-a7f5ba39a7ee</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='virbr0' stp='on' delay='0'/>
  <mac address='52:54:00:29:59:18'/>
  <domain name='default'/>
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254'/>
      <bootp file='/pxelinux.0' server='192.168.100.153'/> ①
</dhcp>
  </ip>
</network>
```

① `bootp` statement that directs to the PXE server.

### TFTP

Uyuni uses the `atftpd` daemon, but it can also use TFTP. The `atftpd` daemon is the recommended method for PXE serviices, and is installed by default. Usually, you do not have to change its configuration, but if you have to, use the YaST Services Manager.

Before TFTP can serve the `pxelinux.0` boot image, you must start the tftp service. Start YaST and use **System › Services Manager** to configure the `tftpd` daemon.

### Syncing TFTP Contents to Uyuni Proxies

It is possible to synchronize Cobbler-generated TFTP contents to Uyuni proxies to perform PXE booting using proxies.

#### Installation

On the Uyuni Server as the root user, install the `susemanager-tftpsync` package:

```
zypper install susemanager-tftpsync
```

On the SUSE Manager Proxy systems as the root user , install the `susemanager-tftpsync-recv` package:

```
zypper install susemanager-tftpsync-recv
```

#### Configuring SUSE Manager Proxy

Execute `configure-tftpsync.sh` on the SUSE Manager Proxy systems.

This setup script asks for hostnames and IP addresses of the Uyuni server and the proxy. Additionally, it asks for the `tftpboot` directory on the proxy. For more information, see the output of `configure-tftpsync.sh --help`.

#### Configuring Uyuni Server

As the root user, execute `configure-tftpsync.sh` on Uyuni Server:

```
configure-tftpsync.sh proxy1.example.com proxy2.example.com
```

Execute `cobbler sync` to initially push the files to the proxy systems. This will succeed if all listed proxies are properly configured.

*Changing the List of Proxy Systems*

You can call `configure-tftpsync.sh` to change the list of proxy systems. You must always provide the full list of proxy systems.

*Reinstalling a Configured Proxy*

If you reinstall an already configured proxy and want to push all the files again you must remove the cache file at `/var/lib/cobbler/pxe_cache.json` before you can call `cobbler sync` again.

#### Requirements

The Uyuni Server must be able to access the SUSE Manager Proxy systems directly. You cannot push using a proxy.

### Syncing and Starting the Cobbler Service

Before starting the Cobbler service, run a check to make sure that all the prerequisites are configured according to your requirements using the `cobbler check` command.

If configuration is correct, start the Uyuni server with this command:

```
/usr/sbin/spacewalk-service start
```

Do not start or stop the `cobblerd` service independent of the Uyuni service. Doing so may cause errors and other issues.

Always use `/usr/sbin/spacewalk-service` to start or stop Uyuni.

## Adding a Distribution to Cobbler

If all Cobbler prerequisites have been met and Cobbler is running, you can use the Cobbler server as an installation source for a distribution:

Make installation files such as the kernel image and the initrd image available on the Cobbler server. Then add a distribution to Cobbler, using either the Web interface or the command line tools.

For information about creating and configuring AutoYaST or Kickstart distributions from the Uyuni interface, refer to xref:FILENAME.adoc#ref.webui.systems.autoinst.distribution[].

To create a distribution from the command line, use the `cobbler` command as root:

```
cobbler distro add --name=`string`--kernel=`path`--initrd=`path`
```

`--name=string` **option**

A label used to differentiate one distribution choice from another (for example, `sles12server`).

`--kernel=path` **option**

Specifies the path to the kernel image file.

`--initrd=path` **option**

specifies the path to the initial ram disk (initrd) image file.

## Adding a Profile to Cobbler

Once you have added a distribution to Cobbler, you can add profiles.

Cobbler profiles associate a distribution with additional options like AutoYaST or Kickstart files. Profiles are the core unit of provisioning and there must be at least one Cobbler profile for every distribution added. For example, two profiles might be created for a Web server and a desktop configuration. While both profiles use the same distribution, the profiles are for different installation types.

For information about creating and configuring Kickstart and AutoYaST profiles in the Uyuni interface, refer to xref:FILENAME.adoc#ref.webui.systems.autoinst.profiles[].

Use the `cobbler` command as root to create profiles from the command line:

```
cobbler profile add --name=string --distro=string [--kickstart=url] \
  [--virt-file-size=gigabytes] [--virt-ram=megabytes]
```

`--name=string`

A unique label for the profile, such as `sles12webserver` or `sles12workstation`.

`--distro=string`

The distribution that will be used for this profile. For adding distributions, see xref:FILENAME.adoc#advanced.topics.cobbler.adddistro[].

`--kickstart=url`

The location of the Kickstart file (if available).

`--virt-file-size=gigabytes`

The size of the virtual guest file image (in gigabytes). The default is 5 GB.

`--virt-ram=megabytes`

The maximum amount of physical RAM a virtual guest can consume (in megabytes). The default is 512 MB.

## Adding a System to Cobbler

Once the distributions and profiles for Cobbler have been created, add systems to Cobbler. System records map a piece of hardware on a client with the Cobbler profile assigned to run on it.

If you are provisioning using `koan` and PXE menus alone, it is not required to create system records. They are useful when system-specific Kickstart templating is required or to establish that a specific system should always get specific content installed. If a client is intended for a certain role, system records should be created for it.

For information about creating and configuring automated installation from the Uyuni interface, refer to xref:FILENAME.adoc#s4-sm-system-details-kick[].

Run this command as the root user to add a system to the Cobbler configuration:

```
cobbler system add --name=string --profile=string \
  --mac-address=AA:BB:CC:DD:EE:FF
```

`--name=string`

A unique label for the system, such as `engineering_server` or `frontoffice_workstation`.

`--profile=string`

Specifies the name of one of the profiles added in xref:FILENAME.adoc#advanced.topics.cobbler.addprofile[].

`--mac-address=AA:BB:CC:DD:EE:FF`

Allows systems with the specified MAC address to automatically be provisioned to the profile associated with the system record when they are being installed.

For more options, such as setting hostname or IP addresses, refer to the Cobbler manpage (`man cobbler`).

## Using Cobbler Templates

The Uyuni web interface allows you to create variables for use with Kickstart distributions and profiles. For more information on creating Kickstart profile variables, refer to xref:FILENAME.adoc#s4-sm-system-kick-details-variables[].

Kickstart variables are part of an infrastructure change in Uyuni to support templating in Kickstart files. Kickstart templates are files that describe how to build Kickstart files, rather than creating specific Kickstarts. The templates are shared by various profiles and systems that have their own variables and corresponding values. These variables modify the templates and a template engine parses the template and variable data into a usable Kickstart file. Cobbler uses an advanced template engine called Cheetah that provides support for templates, variables, and snippets.

Advantages of using templates include:

- Robust features that allow administrators to create and manage large amounts of profiles or systems without duplication of effort or manually creating Kickstarts for every unique situation.
- While templates can become complex and involve loops, conditionals and other enhanced features and syntax, you can also create simpler Kickstart files without such complexity.

### Using Templates

Kickstart templates can have static values for certain common items such as PXE image file names, subnet addresses, and common paths such as `/etc/sysconfig/network-scripts/`. However, templates differ from standard Kickstart files in their use of variables.

For example, a standard Kickstart file may have a networking section similar to this:

```
network --device=eth0 --bootproto=static --ip=192.168.100.24 \
  --netmask=255.255.255.0 --gateway=192.168.100.1 --nameserver=192.168.100.2
```

In a Kickstart template file, the networking section would look like this instead:

```
network --device=$net_dev --bootproto=static --ip=$ip_addr \
  --netmask=255.255.255.0 --gateway=$my_gateway --nameserver=$my_nameserver
```

These variables are substituted with the values set in your Kickstart profile variables or in your system detail variables. If the same variable is defined in both the profile and the system detail, then the system detail variable takes precedence.

The template for the autoinstallation has syntax rules which relies on punctuation symbols. To avoid clashes, they need to be properly treated.

In case the autoinstallation scenario contains any shell script using variables like `$(example)`, its content should be escaped by using the backslash symbol: `\$(example)`.

If the variable named `example` is defined in the autoinstallation snippet, the templating engine will evaluate `$example` with its content. If there is no such variable, the content will be left unchanged. Escaping the `$` symbol will prevent the templating engine from evaluating the symbol as an internal variable. Long scripts or strings can be escaped by wrapping them with the `\#raw` and `\#end raw` directives. For example:

```
#raw
#!/bin/bash
for i in {0..2}; do
 echo "$i - Hello World!"
done
#end raw
```

Also, pay attention to scenarios like this:

```
#start some section (this is a comment)
echo "Hello, world"
#end some section (this is a comment)
```

Any line with a `#` symbol followed by a whitespace is treated as a comment and is therefore not evaluated.

For more information about Kickstart templates, refer to the Cobbler project page at:

https://fedorahosted.org/cobbler/wiki/KickstartTemplating

### Kickstart Snippets

If you have common configurations across all Kickstart templates and profiles, you can use the Snippets feature of Cobbler to take advantage of code reuse.

Kickstart snippets are sections of Kickstart code that can be called by a `$SNIPPET()` function that will be parsed by Cobbler and substituted with the contents of the snippet.

For example, you might have a common hard drive partition configuration for all servers, such as:

```
clearpart --all
part /boot --fstype ext3 --size=150 --asprimary
part / --fstype ext3 --size=40000 --asprimary
part swap --recommended

part pv.00 --size=1 --grow

volgroup vg00 pv.00
logvol /var --name=var vgname=vg00 --fstype ext3 --size=5000
```

Save this snippet of the configuration to a file like `my_partition` and place the file in

`/var/lib/cobbler/snippets/`, where Cobbler can access it.

Use the snippet by calling the `$SNIPPET()` function in your Kickstart templates. For example:

```
$SNIPPET('my_partition')
```

Wherever you invoke that function, the Cheetah parser will substitute the function with the snippet of code contained in the `my_partition` file.

## Using Koan

Whether you are provisioning guests on a virtual machine or reinstalling a new distribution on a running system, Koan works in conjunction with Cobbler to provision systems.

### Using Koan to Provision Virtual Systems

If you have created a virtual machine profile as documented in xref:FILENAME.adoc#advanced.topics.cobbler.addprofile[], you can use `koan` to initiate the installation of a virtual guest on a system. For example, create a Cobbler profile with the following command:

```
cobbler add profile --name=virtualfileserver \
  --distro=sles12-x86_64-server --virt-file-size=20 --virt-ram=1000
```

This profile is for a fileserver running SUSE Linux Enterprise Server 12 with a 20 GB guest image size and allocated 1 GB of system RAM. To find the name of the virtual guest system profile, use the `koan` command:

```
koan --server=hostname --list-profiles
```

This command lists all the available profiles created with `cobbler profile add`.

Create the image file, and begin installation of the virtual guest system:

```
koan --virt --server=cobbler-server.example.com \
  --profile=virtualfileserver --virtname=marketingfileserver
```

This command specifies that a virtual guest system be created from the Cobbler server (hostname `cobbler-server.example.com`) using the `virtualfileserver` profile. The `virtname` option specifies a label for the virtual guest, which by default is the system's MAC address.

Once the installation of the virtual guest is complete, it can be used as any other virtual guest system.

### Using Koan to Reinstall Running Systems

`koan` can replace a still running system with a new installation from the available Cobbler profiles by executing the following command *on the system to be reinstalled*:

```
koan --replace-self --server=hostname --profile=name
```

This command, running on the system to be replaced, will start the provisioning process and replace the system with the profile in `--profile=name` on the Cobbler server specified in `--server=hostname`.

## Building ISOs with Cobbler

Some environments might lack PXE support. The Cobbler `buildiso` command creates a ISO boot image containing a set of distributions and kernels, and a menu similar to PXE network installations. Define the name and output location of the boot ISO using the `--iso` option.

*ISO Build Directory*

Depending on Cobbler-related systemd settings (see `/usr/lib/systemd/system/cobblerd.service`) writing ISO images to public `tmp` directories will not work.

```
cobbler buildiso --iso=/path/to/boot.iso
```

The boot ISO includes all profiles and systems by default. Limit these profiles and systems using the `--profiles` and `--systems` options.

```
cobbler buildiso --systems="system1,system2,system3" \
  --profiles="profile1,profile2,profile3"
```

Building ISOs with the `cobbler buildiso` command is supported for all architectures except the z Systems architecture.

## Bare Metal Provisioning

Systems that have not yet been provisioned are called bare metal systems. You can provision bare metal systems using Cobbler. Once a bare metal system has been provisioned in this way, it will appear in the `Systems` list, where you can perform regular provisioning with autoinstallation, for a completely unattended installation.

### Bare Metal Provisioning System Requirements

To successfully provision a bare metal system, you will require a fully patched Uyuni server, version 2.1 or higher.

The system to be provisioned must have x86_64 architecture, with at least 2 GB RAM, and be capable of PXE booting.

The server uses TFTP to provision the bare metal client, so the appropriate port and networks must be configured correctly in order for provisioning to be successful. In particular, ensure that you have a DHCP server, and have set the `next-server` parameter to the Uyuni server IP address or hostname.

### Enabling Bare Metal Systems Management

Bare metal systems management can be enabled or disabled in the Web UI by clicking **Admin › SUSE Manager Configuration › Bare-metal systems**.

New systems are added to the organization of the administrator who enabled the bare metal systems management feature. To change the organization, log in as an Administrator of the required organization, and re-enable the feature.

Once the feature has been enabled, any bare metal system connected to the server network will be automatically added to the organization when it is powered on. The process can take a few minutes, and the system will automatically shut down once it is complete. After the reboot, the system will appear in the `Systems` list. Click on the name of the system to see basic information, or go to the `Properties`, `Notes`, and `Hardware` tabs for more details. You can migrate bare metal systems to other organizations if required, using the `Migrate` tab.

### Provisioning Bare Metal Systems

Provisioning bare metal systems is similar to provisioning other systems, and can be done using the `Provisioning` tab. However, you will not be able to schedule provisioning, it will happen automatically as soon as the system is configured and powered on.

*Bare Metal and System Set Manager*

System Set Manager can be used with bare metal systems, although not all features will be available, because bare metal systems do not have an operating system installed. This limitation also applies to mixed sets that contain bare metal systems; all features will be re-enabled if the bare metal systems are removed from the set.

### Troubleshooting Bare Metal Systems

If a bare metal system on the network is not automatically added to the `Systems` list, check these things first:

- You must have the `pxe-default-image` package installed.
- File paths and parameters must be configured correctly. Check that the `vmlinuz0` and `initrd0.img` files, which are provided by `pxe-default-image`, are in the locations specified in the `rhn.conf` configuration file.

- Ensure the networking equipment connecting the bare metal system to the Uyuni server is working correctly, and that you can reach the Uyuni server IP address from the server.
- The bare metal system to be provisioned must have PXE booting enabled in the boot sequence, and must not be attempting to boot an operating system.
- The DHCP server must be responding to DHCP requests during boot. Check the PXE boot messages to ensure that:
  - the DHCP server is assigning the expected IP address
  - the DHCP server is assigning the the Uyuni server IP address as `next-server` for booting.
- Ensure Cobbler is running, and that the Discovery feature is enabled.

If you see a blue Cobbler menu shortly after booting, discovery has started. If it does not complete successfully, temporarily disable automatic shutdown in order to help diagnose the problem. To disable automatic shutdown:

1. Select `pxe-default-profile` in the Cobbler menu with the arrow keys, and press the Tab key before the timer expires.
2. Add the kernel boot parameter `spacewalk-finally=running` using the integrated editor, and press Enter to continue booting.
3. Enter a shell with the username `root` and password `linux` to continue debugging.

*Duplicate profiles*

Due to a technical limitation, it is not possible to reliably distinguish a new bare metal system from a system that has previously been discovered. Therefore, we recommended that you do not power on bare metal systems multiple times, as this will result in duplicate profiles.

# Disconnected Setup with RMT or SMT (DMZ)

If it is not possible to connect Uyuni directly or via a proxy to the Internet, a disconnected setup in combination with RMT or SMT is the recommended solution. In this scenario, RMT or SMT stays in an “external” network with a connection to SUSE Customer Center and synchronizes the software channels and repositories on a removable storage medium. Then you separate the storage medium from RMT or SMT, and mount it locally on your Uyuni server to read the updated data.

*Offline Usage Scenario*

SMT and RMT are not made for server cascades. SUSE Manager always connects to SMT or RMT in an offline or disconnected scenario.

*RMT*

The successor of SMT and currently runs on the following systems:

- SUSE Linux Enterprise 15 (when available)
- Temporarily (for testing only): 12 SP2, and 12 SP3
- Not officially supported: openSUSE Leap 42.2, Leap 42.3, and openSUSE Tumbleweed

RMT allows you to provision updates for all of your devices running a product based on SUSE Linux Enterprise 12 SPx and later as well as openSUSE Leap.

*SMT*

The predecessor of RMT and is no longer actively developed. It runs on SUSE Linux Enterprise Server 12 SPx and allows you to provision updates for products based on SUSE Linux Enterprise 12 SPx and earlier. You will still need it, if you want to update SUSE Linux Enterprise 11 clients.

## Repository Management Tool (RMT) and Disconnected Setup (DMZ)

The following procedure will guide you through using RMT. It will work best with a dedicated RMT instance per Uyuni .

*Procedure: RMT: Fetching Repository Data from SUSE Customer Center*

1. Configure RMT in the external network with SCC. For details about configuring RMT, see the official guide (when available).
    a. Preparation work:

    Run `rmt-cli sync` to download available products and repositories data for your organization from SCC.

    Run `rmt-cli products list --all` to see the list of products that are available for your organization.

    Run `rmt-cli repos list --all` to see the list of all repositories available.

b. With `rmt-cli repos enable` enable repositories you want to mirror.

c. With `rmt-cli products enable`enable products. For example, to enable SLES _15:

```
rmt-cli product enable sles/15/x86_64
```

2. Using RMT, mirror all required repositories.
3. Get the required JSON responses from SCC and save them as files at the specified path (for example, `/mnt/usb` ).

*Write Permissions for RMT User*

The directory being written to must be writeable for the same user as the rmt service. The rmt user setting is defined in the `cli` section of `/etc/rmt.conf` .

Enter:

```
{prompt.root}rmt-cli export data /mnt/usb
```

4. Export settings about repositories to mirror to the specified path (in this case, `/mnt/usb` ); this command will create a `repos.json` file there:

```
{prompt.root}rmt-cli export settings /mnt/usb
```

5. Mirror the repositories according to the settings in the `repos.json` file to the specified path (in this case, `/mnt/usb` ).

```
{prompt.root}rmt-cli export repos /mnt/usb
```

6. Unmount the storage medium and carry it securely to your Uyuni server.

On the Uyuni server, continue with Updating Repositories on Uyuni From Storage Media.

## Repository Management Tool (SMT) and Disconnected Setup (DMZ)

The following procedure will guide you through using SMT.

*Procedure: SMT: Fetching Repository Data from SUSE Customer Center*

1. Configure SMT in the external network with SCC. For details about configuring SMT with SUSE Linux Enterprise 12, see https://www.suse.com/documentation/sles-12/book_smt/data/book_smt.html.

2. Using SMT, mirror all required repositories.
3. Create a “database replacement file” (for example, `/tmp/dbrepl.xml` ).

```
{prompt.root}smt-sync --createdbreplacementfile /tmp/dbrepl.xml
```

1. Mount a removable storage medium such as an external hard disk or USB flash drive.
2. Export the data to the mounted medium:

```
smt-sync --todir /media/disk/
smt-mirror --dbreplfile /tmp/dbrepl.xml --directory /media/disk \
           --fromlocalsmt -L /var/log/smt/smt-mirror-export.log
```

*Write Permissions for SMT User*

The directory being written to must be writeable for the same user as the smt daemon (user=smt). The smt user setting is defined in `/etc/smt.conf` . You can check if the correct user is specified via the following command:

```
{prompt.root}egrep '^smtUser' /etc/smt.conf
```

+

+ .Keeping the Disconnected Server Up-to-date NOTE: `smt-sync` also exports your subscription data. To keep Uyuni up-to-date with your subscriptions, you must frequently import and export this data.

+

1. Unmount the storage medium and carry it securely to your Uyuni server.

On the Uyuni server, continue with Updating Repositories on Uyuni From Storage Media.

## Updating Repositories on Uyuni From Storage Media

This procedure will show you how to update the repositories on the Uyuni server from the storage media.

*Procedure: Updating the UyuniServer from the Storage Medium*

1. Mount the storage medium on your Uyuni server (for example, at `/media/disk` ).
2. Specify the local path on the Uyuni server in `/etc/rhn/rhn.conf`:

```
server.susemanager.fromdir = /media/disk
```

This setting is mandatory for SUSE Customer Center and `mgr-sync`.

3. Restart Tomcat:

```
systemctl restart tomcat
```

1. Before performing another operation on the server execute a full sync:

```
mgr-sync refresh   # SCC (fromdir in rhn.conf required!)
```

2. `mgr-sync` can now be executed normally:

```
mgr-sync list channels
mgr-sync add channel channel-label
```

*Data Corruption*

The disk must always be available at the same mount point. To avoid data corruption, do not trigger a sync, if the storage medium is not mounted. If you have already added a channel from a local repository path, you will not be able to change its URL to point to a different path afterwards.

Up-to-date data is now available on your Uyuni server and is ready for updating client systems. According to your maintenance windows or update schedule refresh the data on the storage medium with RMT or SMT.

## Refreshing Data on the Storage Medium

*Procedure: Refreshing Data on the Storage Medium from RMT or SMT*

1. On your Uyuni server, unmount the storage medium and carry it to your RMT or SMT.
2. On your RMT or SMT system, continue with the synchronization step.

*Data Corruption*

The storage medium must always be available at the same mount point. To avoid data corruption, do not trigger a sync if the storage medium is not mounted.

This concludes using RMT or SMT with Uyuni .

# Managing Your Subscriptions

Coming Soon…