SUSE Consulting scripts for SUSE Manager Server for SUSE Consulting

SUSE Services

Jan 15, 2020

Contents

Copyright

SUSE Software Solutions GmbH

Maxfeldstrasse 5

90409 Nuremberg

Germany

www.suse.com

Disclaimer

SUSE LLC, makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Further, SUSE LLC, reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. Further, SUSE LLC, makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SUSE LLC, reserves the right to make changes to any and all parts of SUSE software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, reexport or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. SUSE assumes no responsibility for your failure to obtain any necessary export approvals.

1 SUSE Consulting Scripts

1.1 General

SUSE Consulting has written several (python) scripts that can be used for daily work. The scripts can be download from GitHub under the following link. Under the button **Clone or Download** all scripts can be downloaded. Please copy the zip-file to `/opt/susemanager` on the SUSE Manager server and extract. Following is a description of each script.

Note: These tools are only supported by SUSE Consulting. For bugs and questions please contact SUSE Consulting and **not** SUSE Support!!!!!!!!

1.1.1 configsm.yaml

This is the general configuration file. All scripts read information from this file. There are several sections in this file.

The first section is defining the SUSE Manager Server and the needed login credentials. Also the timeout how long an action can take before the script is stopped with a timeout error. The default is 600 seconds.

```
suman:
    server: <SUSE Manager Server FQDN>
    user: <user within SUSE Manager to perform the actions>
    password: <password from the user>
    timeout: 600
```

Each script can send a mail when there is a warning or a failure that causes the script to stop. If the option `sendmail` is true, a mail should be send for minor and major errors. False if no mail should be send. The mail will be send to the receivers listed at the option `receivers`. The option `sender` indentifies from whom the mail is send. And the option `server` sets the mail relay host. Often this can be the server itself.

```
smtp:
    sendmail: True
    receivers:
        - <receiver 1>
        - <receiver 2>
```

(continues on next page)

(continued from previous page)

```
    sender: <sender identification>
    server: 127.0.0.1
```

Every script will log information in log files. With the option `log_dir` the directory can be defined where the logs should reside. By default this is `/var/log`. It is advised to write the logs in a separate directory. Every script will write information into log files. The script `system_update.py` will create a sub-directory and create a separate log file for every system. The option `scripts_dir` defines the directory where the scripts are. By default this is `/opt/susemanager`. The option `update_script_dir` defines the location where the update scripts can be found that are used during maintenance.

```
dirs:
    log_dir: /var/log
    scripts_dir: /opt/susemanager
    update_script_dir: /opt/susemanager/update_scripts
```

The **maintenance** section is being used by the scripts `system_update.py` and `group_system_update.py`. The option `wait_between_systems` defines the time between each system when running `group_system_update.py`. The default is 5 seconds. Systems defined at the option `exclude_for_patch` will not be updated. In this example server `lx0001` will not be updated. Under the option `sp_migration` it can be defined to which SupportPack can be updated. The same for `sp_migration_project` but her define from which project to upgrade to which project. If none are defined, no SP-Migration will take place. If there is something defined, the OS defined will be updated to the parameter given. In the example given a SLES12 SP2 server will be update to SLES12 SP4. Exceptions can be defined. In the give example the server `lx0003` will not get a SP-migration.

```
maintenance:
    wait_between_systems: 5
    exclude_for_patch:
        - lx0001
    sp_migration:
        sles12-sp2: sles12-sp4
    sp_migration_project:
        s123: s125
        s124: s125
    exception_sp:
        sles12-sp4:
            - lx0003
```

The **channel_cloner** section is being used by the script `channel_cloner.py`. SUSE Consulting will provide more detailed information when and how this script can be used.

```
channel_cloner:
    sles-12-sp4-x86_64:
        base_channel: sles12-sp4-pool-x86_64
```

(continues on next page)

(continued from previous page)

```
        channels:
            - sle-sdk12-sp4-pool-x86_64,RELEASE-sle-sdk12-sp4-pool-x86_64
            - sle-sdk12-sp4-updates-x86_64,RELEASE-sle-sdk12-sp4-updates-x86_64
            - sles12-sp4-updates-x86_64,RELEASE-sles12-sp4-updates-x86_64
            - sle-manager-tools12-pool-x86_64-sp4,RELEASE-sle-manager-tools12-
↪pool-x86_64-sp4
            - sle-manager-tools12-updates-x86_64-sp4,RELEASE-sle-manager-
↪tools12-updates-x86_64-sp4
            - sle-module-adv-systems-management12-pool-x86_64-sp4,RELEASE-sle-
↪module-adv-systems-management12-pool-x86_64-sp4
            - sle-module-adv-systems-management12-updates-x86_64-sp4,RELEASE-
↪sle-module-adv-systems-management12-updates-x86_64-sp4
```

1.1.2 channel_cloner.py

This script is used to created clones that have a different structure then used. Please work together with SUSE Consulting when using this script.

The script has the following parameters:

Table 1.1: Parameters channel_cloner.py

Parameter	Description
-h	Display all available parameters
-r <release>	The <release> that should be used within the channel names. This is mandatory
-s <source>	The OS version. Eg sles12-sp4-x86_64. This is mandatory
-t <todate>	Patches until (and including) this date will be added. Format YYYY-MM-DD.
-o	If the channel already exists it will be overwritten. If this option is not given and the channel exists, the program will abort.

1.1.3 create_software_project.py

This script can be used for three scenarios

- to create a content lifecycle project. For this the following options are mandatory: `-p, -e, -b`. The following options are optional: `-a -d`.

  The following is an example to create a new project:

  ```
  <path to script>/create_software_project.py -p sles12sp4
  -e dev,tst,prd -b sles12-sp4-pool-x86_64 -a customchannel1,
  customchannel2
  ```

- to add channels to an existing project:

The following is an example to add a channel to an existing project:

```
<path to script>/create_software_project.py -p sles12sp4 -a
customchannel1,customchannel2
```

- to delete channels to an existing project:

  The following is an example to add a channel to an existing project:

  ```
  <path to script>/create_software_project.py -p sles12sp4 -d
  customchannel1,customchannel2
  ```

The cloned channels will only be created, the synchronization has to be started manually with the script `sync_stage.py`.

The script has the following parameters:

Table 1.2: Parameters `create_software_project.py`

Parameter	Description
-h	Display all available parameters
-p <project>	The name of the project. This is mandatory
-e <environment>	A comma-delimited list (without spaces) of the environments to be created. Eg `-e dev,tst,prd`. This is mandatory
-b <basechannel>	The basechannel label of which the channel should be created. Eg `-b sles12-sp4-pool-x86_64`. This is mandatory
-a <add channels>	A comma-delimited list (without spaces) of channels that should be added to the given project.
-d <delete channels>	A comma-delimited list (without spaces) of channels that should be removed from the given project.
-m <descriptopn>	Description of the project. Is optional.

1.1.4 `group_system_update.py`

This script will update via the script `system_update.py` all systems in the given system group.

The script has the following parameters:

Table 1.3: Parameters `group_system_update.py`

Parameter	Description
-h	Display all available parameters
-g <system group>	The name of the system group of which the systems should be updates. This is mandatory
-c	If this paramater is given, the configuration of the system will also be updated.
-u	If this paramater is given, the update scripts of the system will also be excuted.

1.1.5 smtools.py

This script is not executable. It is a library used within all other scripts.

1.1.6 sync_channel.py

This script will update the given channel from the channel it has been cloned from.

The script has the following parameters:

Table 1.4: Parameters sync_channel.py

Parameter	Description
-h	Display all available parameters
-g <channel label>	The label of the channel that should be updated. This is mandatory

1.1.7 sync_environment.py

This script will update the given environment in all projects. This is useful when there are multiple projects with the same environments and a specific environment has to be updated for all projects where the environment is present.

The script has the following parameters:

Table 1.5: Parameters sync_environment.py

Parameter	Description
-h	Display all available parameters
-e <environment>	The name of the environment that should be updated. This is mandatory

1.1.8 sync_stage.py

With this script there are two options:

- update a given base channel. For this use the option `-c`.
- update the given environment for the given project. For this the options `-p`, `-e` and `-m` are being used.

The script has the following parameters:

Table 1.6: Parameters `sync_stage.py`

Parameter	Description
-h	Display all available parameters
-p <project>	The name of the project.
-e <environment>	The name of the environments to be updated.
-m <descriptopn>	Description of the project. This is optional and when not given a default message will be used: `Created on <date>`.
-c <basechannel>	The basechannel label of which the channels should be updated.
-b	This will create a backup of the basechannel and all the child channels. The current date will be in the new labels. Optional

1.1.9 `system_update.py`

This script will perform several actions while updating a server. If the server is inactive, the script will stop immediately. First it will check if the given server is mentioned in the `configsm.yaml` in the under `maintenance.exclude_for_patch`. If this is the case the script will stop and the system will be updated. If this is not the case, it will check if there is a SupportPack migration defined for the running OS on this system. If this is the case and the system is not in the list of exclude systems (`maintenance.exception_sp`), the server will be migrated to the given SupportPack. But first it will be updated. If there is no Support Pack migration available or the system is excluded, it will receive a regular update. The update will be done in four steps. First the `salt-minion` will be updated. In the second step the update stack (`zypper` and `zypplib`) will be updated. When this is completed all available patches will be applied and the in the last step packages that are not part of a patch will be updated. When the option `-c` is given, the system will get a salt highstate before and after the packages are installed. Also the system will be rebooted, unless the option `-n` is given.

Also there is the option to execute scripts or state channels before and after the update. This is defined with the option `-u`. When this option is being set, the script will check the directory defined with the option `update_script_dir` in the `configsm.yaml` if there is a file called `general` and a file with the same name of the server (as defined in SUSE Manager). It will combine these two and execute when needed.

The script has the following parameters:

Table 1.7: Parameters `system_update.py`

Parameter	Description
-h	Display all available parameters
-s <server>	The name of the server. This has to be equal to the profile name in SUSE Manager.
-n	The server will not be rebooted after the patching is completed.
-c	Apply configuration after and before patching.
-u	Excute the server specific _start and _end scripts, or what is defined in general

1.1.10 `<update_script_dir>/general` and server specific

The files have the same format and should look like:

```
begin:
    timeout: 60
    commands:
    state:
end:
    timeout: 60
    commands:
    state:
```

The block under `begin` will be executed before the server will be patched or will get a SP-migration. The block under `end` will be executed after the server is rebooted. The `timeout` is the time the executing of a script may run. If the time is exceeded, the job will be reported as failed. So ensure that the timeout is high enough. Under `commands` list the commands to be executed. This should be preceded with a -. Under `state` define the state channels that should be executed. This should be preceded with a -.

An example:

```
begin:
    timeout: 60
    commands:
     - cd /tmp; rm -rf *
    state:
     - sap_hana_stop
end:
    timeout: 60
    commands:
     - /opt/something/something.sh
    state:
     - sap_hana_start
```