Hewlett Packard
Enterprise

HPE Insight Control Server Provisioning

How to Create an OS Build Plan for Installing Docker EE on SLES 12

Technical white paper

Contents

Summary

Insight Control server provisioning (ICsp) provides OS Build Plans, scripts, packages, and configuration files that are used to deploy operating systems, configure hardware, update firmware and perform scripted installations.

ICsp tasks, such as installing a server or updating firmware, are performed using OS Build Plans (OSBP). OS Build Plans are simply a collection of ordered steps and associated parameters that when placed together, in the proper order, can perform just about any action you require. Insight Control server provisioning comes ready to run, with sample build plans and build plan steps that are designed to work right out of the box. These sample build plans are very important, because they demonstrate the steps needed to perform the most common deployment-related operations.

Docker Enterprise Edition (Docker EE) is designed for enterprise development and IT teams who build, ship, and run business critical applications in production at scale. The Insight Control server provisioning appliances do not have an OSBP to perform scripted installation of Docker Enterprise Edition (EE). This document provides instructions and sample scripts to be used in conjunction with the existing OS Build Plans in ICsp to automate the provision of Docker ready servers on HPE ProLiant and Synergy Server.

Enhancing ICsp OS Build Plans to add Docker EE deployment automation has the following benefits:

- Accelerates Docker EE deployment time.
- Minimizes unintended user installation issues.
- Enables HPE to incorporate Docker installation best practices as part of the ICsp deployment plan.

Target audience: This document is intended for IT architects, IT administrators and system integrators. This document assumes that the reader is familiar with Insight Control server provisioning.

Docker Enterprise Edition overview

Docker EE is integrated, certified and supported to provide enterprises with the most secure container platform in the industry to modernize all applications. An application centric platform, Docker EE is designed to accelerate and secure the entire software supply chain, from development to production while running on any infrastructure.

Docker Enterprise Edition provides a container runtime, with integrated and multi-tenant orchestration, security and management in addition to an ecosystem of certified technologies. This gives enterprises an open container platform that ensures a simplified yet rich user experience. The new modular platform makes it easy to install, configure and upgrade Docker on certified infrastructure (operating systems and cloud providers).

Docker EE is a certified container platform for the CentOS Distribution, Red Hat Enterprise Linux (RHEL), Ubuntu, SUSE Linux Enterprise Server (SLES), Oracle Linux and Windows Server 2016, as well as cloud providers AWS and Azure. Docker and its partners provide cooperative support for certified containers and plugins so that customers can confidently use these products in production.

Docker EE is available in three tiers:

- Basic: The Docker platform for certified infrastructure, with support from Docker Inc. and Certified Containers and Plugins from Docker Store.
- Standard: Adds secure multi-tenancy with advanced image, container management, LDAP/AD user integration, secure software supply chain (Docker Datacenter).
- Advanced: Adds Docker Security Scanning and continuous vulnerability monitoring.

Docker EE secures the entire software supply chain from development to production, ensuring all applications are safer. Regardless of whether it is a traditional homegrown application, ISV application or micro services architecture, it will benefit from a modern security model with Docker EE.

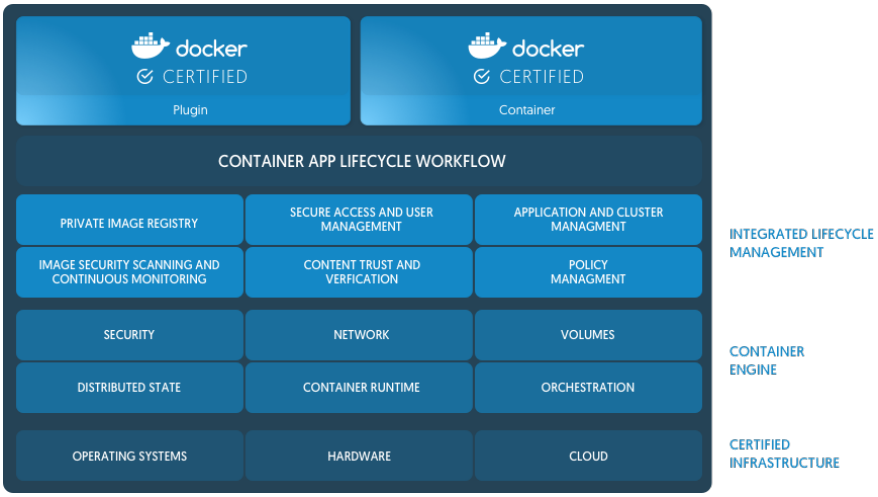

Figure 1. Docker Enterprise Edition

For more information on Docker EE refer to Docker Enterprise Edition.

HPE Insight Control server provisioning overview

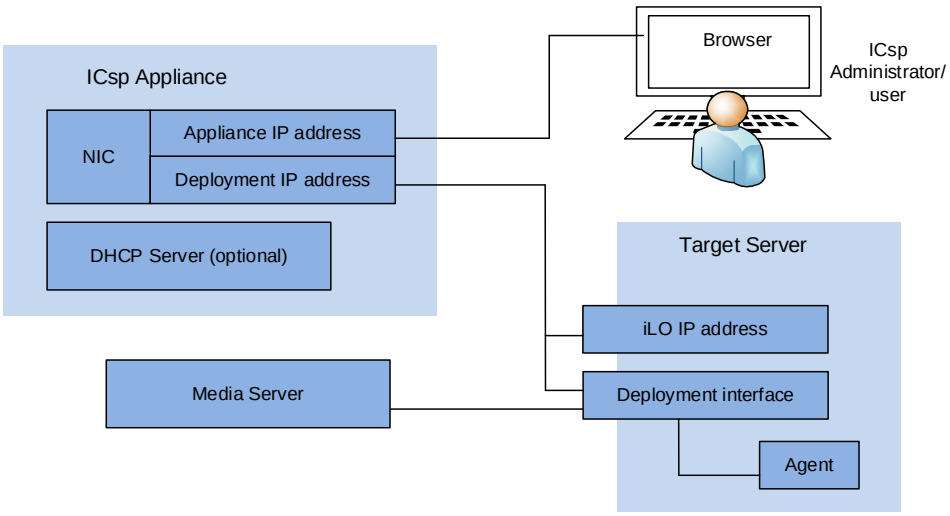

Figure 2. ICsp appliance architecture

The Insight Control (IC) server provisioning appliance is pre-packaged with OS Build Plans (OSBPs) that perform scripted installations. For detailed information on the Insight Control server provisioning appliance setup, see the HPE Insight Control server provisioning Installation Guide at http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=c05305712.

The Media Server is a virtual appliance, separate from Insight Control server provisioning, that holds deployment software. OS Build Plans use the software on the Media Server to provision managed servers. Software (media) on the Media Server can include vendor-supplied OS distribution files, captured images, and firmware and driver updates such as HPE Service Packs for ProLiant (HPE SPP). Having a separate Media Server allows you to easily manage the disk space and contents of the Media Server as opposed to being constrained by the appliance. You can also tailor the network bandwidth requirements of the Media Server to the deployment load.

For instructions on setting up a Media Server, see the "HPE Insight Control server provisioning Installation Guide," available at http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=c05305712.

For further instructions on managing and administering an ICsp appliance, please refer to the "HPE Insight Control Server Provisioning Administrator Guide" available at http://h20566.www2.hpe.com/hpsc/doc/public/display?docId=c05303955.

For extensive information on ICsp OS Build Plans, please refer to the "HPE Insight Control Build Plans Reference Guide" available at http://h20566.www2.hpe.com/hpsc/doc/public/display?docId=c05305518.

This section describes the challenges this solution is built to address and the benefits of using the solution.

OS Build Plans

An OS Build Plan is a sequence of steps that execute in a specific order to perform a task on a target server. OS Build Plan steps are autonomous operations, such as `run script` or `install package`. OS Build Plans are typically used for provisioning operating systems, but can be used to automate most tasks. OS Build Plans use the software on the Media Server to provision managed servers.

Four types of steps are available: Run Script, Deploy Package, Deploy Configuration File, and Capture Configuration File.

Creating Build Plans

You can use existing OS Build Plans as templates for creating your own. Hewlett Packard Enterprise supplies OS Build Plans with the Insight Control server provisioning product that work out of the box, but they are also designed to be used as templates. The HPE-provided OS Build Plans in ICsp are read-only and may not be edited, but you can save a copy as a starting point to work from.

Custom attributes

Custom attributes substitute specific values into scripts, configuration files, and package paths when an OS Build Plan is run. This is useful for configuring installation processes, including network and server configuration. Custom attributes are typically used for overriding default values.

Deploy Configuration File

Configuration files are text files stored on the appliance that are used for text-based data such as unattended installation files or hardware configuration files. The `Deploy Configuration File` step takes the specified configuration file and writes it to a user-specified location on the target server. These steps are often followed by a `run script` step that makes use of the configuration file. You can use one of the many sample configurations provided by HPE or you can create your own.

Scripts

Scripts in Insight Control server provisioning are used to accomplish provisioning tasks. To run a single script, insert it as a step in an OS Build Plan. Alternatively, you can run multiple scripts using a series of steps within a Build Plan. Insight Control server provisioning supports a variety of script types, for example, bash shell (sh), C shell (csh), KornShell (ksh), Python, Windows batch file (.BAT) and Opsware Global File System (OGFS) scripts.

Supported Configurations

This ICsp OS Build Plan has been tested using ICsp 7.6 on ProLiant DL/BL Servers and Synergy against SLES 12 SP1. The OSBP should work on any Proliant DL/BL Servers and Synergy Compute Modules that are supported by ICsp.

Location of required ICsp scripts

All the required scripts to create the OS Build Plan are available at https://github.com/HewlettPackard/ICsp-Docker-OSBP, under the **sles** folder.

OS Build Plan for Docker Enterprise Edition installation on SLES 12

Prerequisites

- ICsp is installed and configured to deploy SLES 12 OS.
- If the target system is behind a proxy, the proxy hostname and port must be provided.

Register Servers and Create Custom Build Plan

1. Register each of the servers that you will be deploying in ICsp by providing the required iLO information:

   Log in to the ICsp server and navigate to `Servers` → `Add server`

Populate the fields below:

a. iLO IP Address

b. Username

c. Password

Click Add.

2. Log in to the server configured as your HPE ICsp media server.
3. Upload SLES 12 SP1 to ICsp media server:

a. Create a directory on the media server share for SLES 12 SP1. Confirm these directories can be seen in a web browser by using the Media Server URL as shown in Fig. 3

Figure 3. ICsp Media Server with SLES 12 SP1 media

Note: for more information about creating and using a Media Server, please check the official ICsp Installation Guide at: http://h20565.www2.hpe.com/hpsc/doc/public/display?docId=c05305712

b. Copy the SLES 12 SP1 installation media to the Media server `sles12sp1-x64` directory created in the previous step. Once the copy is complete, confirm that the full contents of the ISO are visible from a web browser at the URL http://<media_server>/<share_name>, as shown in Figure 4.

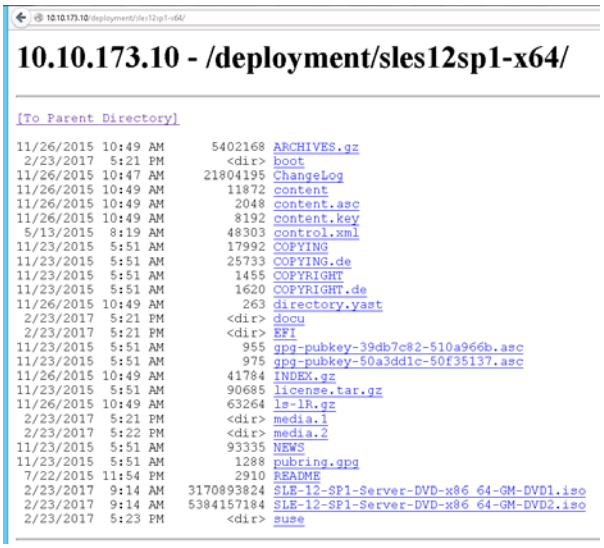

Figure 4. ICsp Media Server

4. Create a Custom Build Plan as follows:
   a. Select the ICsp provided “ProLiant OS - SLES 12 SP1 x64 Scripted Install“ OS Build Plan
   b. Save the OS Build Plan with a new name, for example “ProLiant OS – SLES12 SP1 x64 Scripted Install with Docker”

`Actions → Save as`

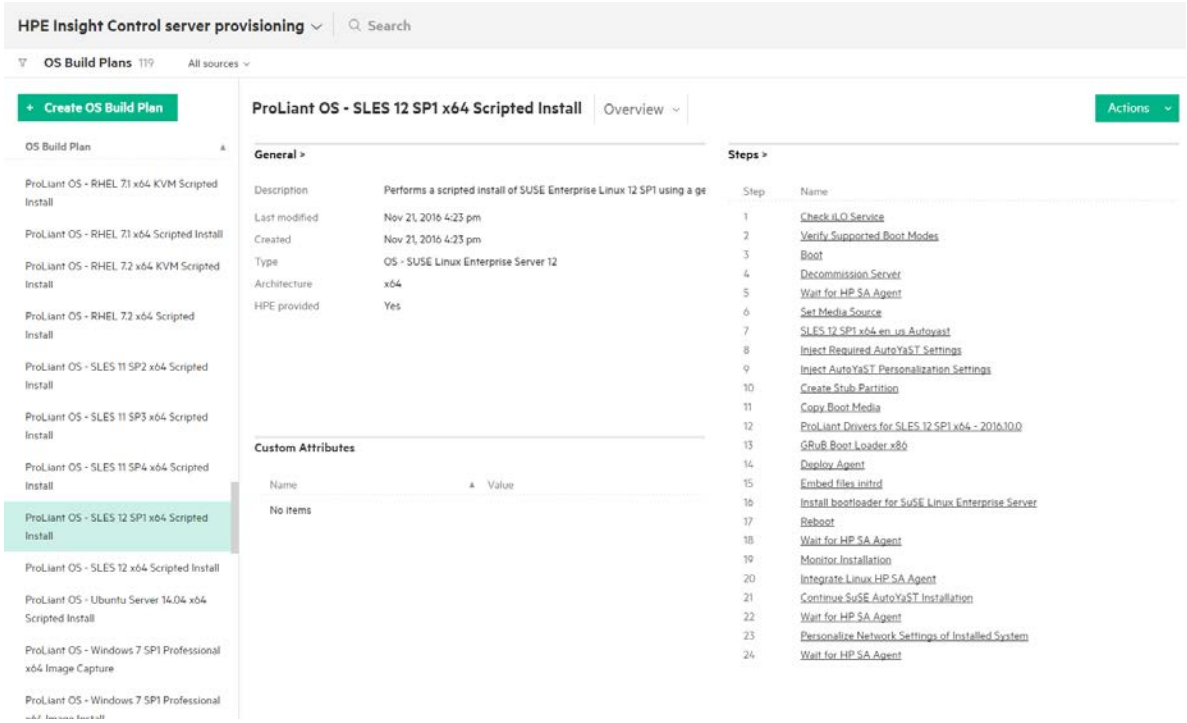

Figure 5. SLES 12 sP1 OS Build Plan

5. Add attributes to the OS Build Plan using `Custom Attributes` → `Edit`

   Custom attributes are user-defined name/value pairs that are used as a form of variable substitution in scripts and other appliance functions. To create custom attributes, select `OS Build Plans` → `Custom Attributes` → `Edit` → `Create Custom Attribute`.

   For Docker installation, the following attributes are created:

   a. **docker_repo (Mandatory):** Docker repository for Docker Enterprise Edition. This can be either the URL provided by Docker (external URL) or an internal URL accessible via http. When using an external URL, please note that if your license covers different Linux systems you will need to add the '`sles`' folder. For instance if the URL you've been provided with is '`https://storebits.docker.com/ee/linux/sub-36xxxxx3-dcc3-4ae8-b36d-06xxxxxa9/`', the custom attribute needs to be '`https://storebits.docker.com/ee/linux/sub-36xxxxx3-dcc3-4ae8-b36d-06xxxxxa9/sles`'. If your license only covers SLES then the provided URL should be enough. In case of doubt, please navigate to the URL using a browser to discover the correct URL. Please make sure to specify the protocol (e.g. `https://`) when specifying the URL.

   b. **docker_version (Optional):** version of Docker Enterprise Edition to be installed (i.e. `17.03`). If no version is specified then the latest found will be installed.

   c. **internal_sles_repo (Optional):** URL pointing to an internal SLES repository. For systems where the internet access is restricted, an internal repository can be used instead. When this custom attribute is specified e.g. http://<repository server>/<path to your repo>, the Docker EE repository URL associated with the subscription will be skipped. Please make sure to specify the protocol (e.g. `http://`) when specifying the URL.

   d. **nic_teaming (Optional):** the OSBP has the option to create one or more NIC teams to provide HA networking. This custom attribute defines the list of NIC teams we intend to create. Format is as follows::

```
<Team name1>, <MAC address 1>, <MAC address 2>
<Team name2>, <MAC address 3>, <MAC address 4>
...
```

   For instance:

```
Team0, 00:11:22:33:44:55, 00:11:22:33:44:66
Team1, 00:11:22:33:44:77, 00:11:22:33:44:88
...
```

This custom attribute can have any numbers of NIC pairs, but can also be left empty if NIC teaming is not required in the system. The IP address assigned to the NIC team will be chosen as follows:

   I. The static IP of the first NIC, if available, or

   II. The static IP of the second NIC, if available, or

   III. A DHCP provided IP if both NICs are set on DHCP.

   e. **proxy_hostname (Optional):** Proxy hostname.

   f. **proxy_port (Optional):** Proxy port.

   g. **no_proxy (Optional):** Comma-separated list of IP addresses or server names where the proxy should not be used for. Please make sure to include at least the media server IP address here, in order to avoid installation issues.

6. Create a new Custom Configuration file that will include a modified AutoYaST configuration file to replace the default one in the original OSBP. This new AutoYaST file contains the following changes:

   – Creation of a `docker` user.

   – Enablement of the SSH service (disabled in the default AutoYaST)

To create a Custom Configuration AutoYaST File:

Configuration Files → Select SLES 12 x64 en_us Autoyast → Actions → Save As << SLES 12 x64 en_us Autoyast for Docker >>

Edit the << SLES 12 x64 en_us Autoyast for Docker >> configuration file (AutoYaST)

Configuration Files → Select << SLES 12 x64 en_us Autoyast for Docker >> → Actions →Edit

Download the AutoYaST file named "autoyast_SLES12.txt" from GitHub, copy the contents of the file into the newly created Configuration File and select OK to save the configuration file. Replace the ICsp default AutoYaST configuration file in step 7 of the OS Build Plan with the newly created custom AutoYaST configuration file.

Refer to Appendix A for the script if you have trouble accessing it on GitHub.

7. Add a script to install Docker at the very end of the OSBP, which includes:
   - Download and installation of Docker EE packages.
   - Enablement and start-up of the Docker service.
   - Display of Docker info and run of a "hello world" container with the docker user to test that Docker is properly installed.

   To do this, follow the steps below:

   a. Download the script "install_docker_on_SLES12.sh" from GitHub to install Docker EE. Refer to Appendix B for the script if you have trouble accessing it on GitHub.

   b. Create a new Script in ICsp using

      Scripts → Actions → Create Script

   c. Copy the contents of the downloaded script and save.

   d. Add the script to the end of your OS Build Plan (step 25)

      OS BuildPlan → Actions → Edit → Add steps

8. Add a script to provide NIC teaming between one or more pairs of NICs, defined in one of the custom attributes specified above. If the custom attribute is left blank or if the values are incorrect, the script will finish without errors but won't attempt to create any NIC teams.

   a. Download the script named "nic_teaming_on_SLES12.sh" from GitHub to install NIC teaming. Refer to Appendix C for the script if you have trouble accessing it on GitHub.

   b. Choose Scripts → Actions → Create Script

   c. Copy the contents of downloaded script and save.

   d. Add the script at the end of the OS Build Plan (step 26)

The OSBP is now complete and ready to deploy Docker EE to a target server. To run the OSBP on the target server, select the server and click Actions → Run OS Build Plans. You should see the message stating that the job succeeded once the job OS Build Plan is successful.

You should be able to login via SSH to the brand new system using the docker account and the password ChangeMe123!. You can then switch to root if required using the same password, but you won't be allowed to connect directly with root via SSH. It is highly recommended that you change both passwords as soon as you log in for the first time.

Note: the docker user is not part of the sudoers by default, so you won't be able to run privileged commands or to switch to root by using the sudo command. You should instead switch to root by using the su command (with either "su -" or "su - root") and then entering the root password.

Appendix A – SLES 12 AutoYaST file for Docker

```
<?xml version="1.0"?>
<!DOCTYPE profile>
<profile xmlns="http://www.suse.com/1.0/yast2ns" xmlns:config="http://www.suse.com/1.0/configns">
<!--
Copyright 2017 Hewlett Packard Enterprise Development LP Licensed under the
Apache License, Version 2.0 (the "License"); you may not use this file except
in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
or agreed to in writing, software distributed under the License is distributed
on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied. See the License for the specific language governing
permissions and limitations under the License.
-->

<!-- The "bootloader" section has been commented out to make this AutoYaST
configuration as generic as possible, and let SLES automatically select
the appropriate boot loader (eLiLo or Grub) for the current boot mode
(UEFI or Legacy). To specify boot loader options, uncomment the "bootloader"
section and add the necessary customizations for the boot loader that is
 appropriate for your current boot mode. -->

<!--
  <bootloader>
    <loader_type>grub</loader_type>
    <sections config:type="list">
      <section>
        <append>@kernel_arguments: @</append>
        <type>image</type>
      </section>
    </sections>
  </bootloader>
-->

  <deploy_image>
    <image_installation config:type="boolean">false</image_installation>
  </deploy_image>

  <general>
    <ask-list config:type="list"/>
    <mode>
      <confirm config:type="boolean">false</confirm>
      <interactive config:type="boolean">false</interactive>
      <reboot config:type="boolean">false</reboot>
    </mode>
    <mouse>
      <id>probe</id>
    </mouse>
    <proposals config:type="list"/>
    <signature-handling>
      <accept_file_without_checksum config:type="boolean">true</accept_file_without_checksum>
      <accept_non_trusted_gpg_key config:type="boolean">true</accept_non_trusted_gpg_key>
      <accept_unknown_gpg_key config:type="boolean">true</accept_unknown_gpg_key>
      <accept_unsigned_file config:type="boolean">true</accept_unsigned_file>
      <accept_verification_failed config:type="boolean">true</accept_verification_failed>
      <import_gpg_key config:type="boolean">true</import_gpg_key>
    </signature-handling>
  </general>
```

```
  <groups config:type="list"/>

  <keyboard>
    <keymap>english-us</keymap>
  </keyboard>

  <language>
    <language>en_US</language>
    <languages>en_US</languages>
  </language>

  <login_settings/>

  <networking>
    <start_immediately   config:type="boolean">true</start_immediately>
  </networking>

<!-- To make this installation as generic as possible, this installation
will use all the available disk for installing SLES OS. SLES Installer
creates the default partitions and file systems based on current boot mode
(UEFI or Legacy), to specify partition options, add the "partition" section
and the necessary customizations for the drivers and partitions that is
appropriate for your current boot mode and storage configuration.-->

  <partitioning config:type="list">
    <drive>
      <use>all</use>
    </drive>
  </partitioning>

  <report>
    <errors>
      <timeout config:type="integer">10</timeout>
    </errors>
    <messages>
      <timeout config:type="integer">10</timeout>
    </messages>
    <warnings>
      <timeout config:type="integer">10</timeout>
    </warnings>
  </report>

  <runlevel>
      <default>3</default>
      <services config:type="list">
       <service>
         <service_name>snmpd</service_name>
         <service_status>enable</service_status>
       </service>
      </services>
   </runlevel>

<!-- Packages needed for successfull installation of HP SPP
-->
   <software>
      <patterns config:type="list">
        <pattern>base</pattern>
        <pattern>x11</pattern>
        <pattern>Basis-Devel</pattern>
        <pattern>apparmor</pattern>
```

```
    </patterns>
    <packages config:type="list">
      <package>autoyast2-installation</package>
      <package>net-snmp</package>
      <package>perl-SNMP</package>
      <package>libstdc++6-32bit</package>
      <package>kernel-syms</package>
             <package>patterns-sles-printing</package>
             <package>libHBAAPI2</package>
    </packages>
  </software>

<services-manager>
   <default_target>multi-user</default_target>
      <services>
        <enable config:type="list">
           <service>sshd</service>
        </enable>
      </services>
</services-manager>

<timezone>
  <hwclock>localtime</hwclock>
  <timezone>America/Chicago</timezone>
</timezone>

<user_defaults>
  <expire></expire>
  <group>100</group>
  <groups>video,dialout</groups>
  <home>/home</home>
  <inactive>-1</inactive>
  <shell>/bin/bash</shell>
  <skel>/etc/skel</skel>
</user_defaults>

<users config:type="list">
   <user>
      <fullname>root</fullname>
      <gid>0</gid>
      <home>/root</home>
      <password_settings>
        <inact>99999</inact>
        <max>99999</max>
        <min>99999</min>
        <warn>1</warn>
      </password_settings>
      <shell>/bin/bash</shell>
      <uid>0</uid>
      <encrypted config:type="boolean">true</encrypted>
      <user_password>@encrypted_root_password:$1$7z4m7f1z$wliShMhVv2HuCAPmuiQzV1@</user_password>
      <username>root</username>
    </user>
    <user>
      <fullname>docker</fullname>
      <gid>500</gid>
      <home>/home/docker</home>
      <password_settings>
        <inact>99999</inact>
        <max>99999</max>
```

```
          <min>99999</min>
          <warn>1</warn>
        </password_settings>
        <shell>/bin/bash</shell>
        <uid>500</uid>
        <encrypted config:type="boolean">true</encrypted>
        <user_password>@encrypted_root_password:$1$7z4m7f1z$wliShMhVv2HuCAPmuiQzV1@</user_password>
        <username>docker</username>
      </user>
  </users>
 <firewall>
      <enable_firewall config:type="boolean">false</enable_firewall>
      <start_firewall config:type="boolean">false</start_firewall>
    </firewall>

 <scripts>
       <pre-scripts>
      <script>
        <filename>startfw.sh</filename>
        <interpreter>shell</interpreter>
        <source>
                <![CDATA[### Loading Dynamic Smart Array hpvsa and hpdsa drivers###
#!/bin/sh
modprobe hpvsa
modprobe hpdsa
]]>
       </source>
      </script>
    </pre-scripts>

<!-- To Enable the Firewall comment the above firewall element lines and remove this comment. Additional port
numbers can be added to variable port

    <init-scripts config:type="list">
      <script>
        <filename>startfw.sh</filename>
        <interpreter>shell</interpreter>
        <source>
               <![CDATA[### Enabling Firewall ###

#!/bin/sh

port="1002"
echo "Init-script startfirewall: Starting the firewall"
service SuSEfirewall2_init start
service SuSEfirewall2_setup start
echo "Inject firewall settings"
if [ /sbin/SuSEfirewall2 status &>/dev/null ] ; then
                if [ ! grep -q "FW_SERVICES_EXT_TCP=.*$port" /etc/sysconfig/SuSEfirewall2] ; then
                                echo "- opening port $port"
                                sed -i "s|\(FW_SERVICES_EXT_TCP=\".*\)\"|\1 $port\"|;s|\" |\"|"
/etc/sysconfig/SuSEfirewall2
                                /sbin/SuSEfirewall2 start
                fi
                if [ ! grep -q "FW_SERVICES_EXT_IP=.icmp*" /etc/sysconfig/SuSEfirewall2]; then
                                echo "Setting FW_SERVICES_EXT_IP "
                                sed -i "s/FW_SERVICES_EXT_IP=\"\"/FW_SERVICES_EXT_IP=\"icmp\"/g"
/etc/sysconfig/SuSEfirewall2
                                /sbin/SuSEfirewall2 start
                fi
```

```
                if [ ! grep -q "FW_DEV_EXT=.any*" /etc/sysconfig/SuSEfirewall2] ; then
                                echo "Setting FW_DEV_EXT"
                                sed -i "s/FW_DEV_EXT=\"\"/FW_DEV_EXT=\"any\"/g" /etc/sysconfig/SuSEfirewall2
                                /sbin/SuSEfirewall2 start
                fi
fi
echo "Init-script startfirewall: Permanently enabling the firewall"
chkconfig SuSEfirewall2_init on
chkconfig SuSEfirewall2_setup on
]]>
       </source>
      </script>
    </init-scripts> -->
  </scripts>
</profile>
```

Appendix B – Docker Enterprise Edition installation script for SUSE 12

```
#!/bin/bash

# Copyright 2017 Hewlett Packard Enterprise Development LP Licensed under the
# Apache License, Version 2.0 (the "License"); you may not use this file except
# in compliance with the License. You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
# or agreed to in writing, software distributed under the License is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

## Logging functions

# 0 - no output, 1 - error|warn messages, 2 - info|error|warn messages, 3 - debug|info|error|warn messages
LOG_FILE_LEVEL=${LOG_FILE_LEVEL:-3}
LOG_STDIO_LEVEL=${LOG_STDIO_LEVEL:-2}
LOG_FILE_DRIVER=${LOG_FILE_DRIVER:-1}
LOG_STDIO_DRIVER=${LOG_STDIO_DRIVER:-1}

SCRIPT_LOG=$HOME/bootstrap.log
touch $SCRIPT_LOG

function EMIT_FILE_DRIVER() {
    if [ 1 -eq $LOG_FILE_DRIVER ]; then
        echo -e $2 >> $SCRIPT_LOG
    fi
}

function EMIT_STDIO_DRIVER() {
    if [ 1 -eq $LOG_STDIO_DRIVER ]; then
        echo -e $2
    fi
}

function EMIT_LOG(){
    # 0 - no output, 1 - error|warn messages, 2 - info|error|warn messages, 3 - debug|info|error|warn messages
    case $1 in
        ERROR|WARN) [ $LOG_STDIO_LEVEL -gt 0 ] && EMIT_STDIO_DRIVER "$@" ;;
        INFO)  [ $LOG_STDIO_LEVEL -gt 1 ] && EMIT_STDIO_DRIVER "$@" ;;
        DEBUG) [ $LOG_STDIO_LEVEL -gt 2 ] && EMIT_STDIO_DRIVER "$@" ;;
    esac
    case $1 in
        ERROR|WARN) [ $LOG_FILE_LEVEL -gt 0 ] && EMIT_FILE_DRIVER "$@" ;;
        INFO)  [ $LOG_FILE_LEVEL -gt 1 ] && EMIT_FILE_DRIVER "$@" ;;
        DEBUG) [ $LOG_FILE_LEVEL -gt 2 ] && EMIT_FILE_DRIVER "$@" ;;
    esac
    r=$? #mute the error level for loglevel checks
}

function SCRIPTENTRY(){
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    script_name=`basename "$0"`
    script_name="${script_name%.*}"
    EMIT_LOG DEBUG "[$timeAndDate] [DEBUG] [$ln] > $script_name $FUNCNAME"
}
```

```
function SCRIPTEXIT(){
    local ln="${BASH_LINENO[0]}"
    script_name=`basename "$0"`
    script_name="${script_name%.*}"
    EMIT_LOG DEBUG "[$timeAndDate] [DEBUG] [$ln] < $script_name $FUNCNAME"
}

function ENTRY(){
    local cfn="${FUNCNAME[1]}"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG DEBUG "[$timeAndDate] [DEBUG] [$ln] > $cfn $FUNCNAME"
}

function EXIT(){
    local cfn="${FUNCNAME[1]}"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG DEBUG "[$timeAndDate] [DEBUG] [$ln] < $cfn $FUNCNAME"
}


function INFO(){
    local function_name="${FUNCNAME[1]}"
    local msg="$1"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG INFO "[$timeAndDate] [INFO] [$ln] $msg"
}


function DEBUG(){
    local function_name="${FUNCNAME[1]}"
    local msg="$1"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG DEBUG "[$timeAndDate] [DEBUG] [$ln] $msg"
}

function ERROR(){
    local function_name="${FUNCNAME[1]}"
    local msg="$1"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG ERROR "[$timeAndDate] [ERROR] [$ln] $msg"
}

function WARN(){
    local function_name="${FUNCNAME[1]}"
    local msg="$1"
    local ln="${BASH_LINENO[0]}"
    timeAndDate=`date`
    EMIT_LOG WARN "[$timeAndDate] [WARN] [$ln] $msg"
}

#
#
## Variables
docker_service_dir='/etc/systemd/system/docker.service.d'
docker_proxy_conf='http-proxy.conf'
```

```
docker_user=docker
docker_repo="@docker_repo@"
docker_version="@docker_version@"
internal_sles_repo="@internal_sles_repo@"
sles_version="12.3" # Only supported version

#
#
## Proxy settings
export proxy_hostname=@proxy_hostname@
export proxy_port=@proxy_port@
export http_proxy=http://@proxy_hostname@:@proxy_port@
export https_proxy=https://@proxy_hostname@:@proxy_port@
export no_proxy=@no_proxy@

#
#
## Script functions
function install_docker() {
    INFO 'Removing previous repo (if any)...'
    zypper --non-interactive rr docker-ee-stable >> /dev/null 2>&1

    INFO 'Installing the Docker repository...'
    if [ ${#docker_version} -gt 0 ]; then
        zypper addrepo ${docker_repo}/${sles_version}/x86_64/stable-${docker_version} docker-ee-stable && zypper
--no-gpg-checks refresh
    else
        zypper addrepo ${docker_repo}/${sles_version}/x86_64/stable docker-ee-stable && zypper --no-gpg-checks
refresh
    fi
    [ $? -ne 0 ] && ERROR 'There was a problem setting the Docker repository!' && exit 1

    # We rebuild the DB to avoid occasional failures when caching the rpm database
    rpmdb --rebuilddb

    INFO 'Installing Docker...'
    zypper --non-interactive --no-gpg-checks install docker-ee
    [ $? -ne 0 ] && ERROR 'There was a problem installing Docker EE!' && exit 1

    ## Adding the proxy configuration to the docker daemon env vars
    if [ ! -z "${http_proxy}" ] || [ ! -z "${https_proxy}" ]; then
        INFO 'Adding proxy settings to daemon configuration...'
        env="Environment="
        [ ! -z "${http_proxy}" ]  && env="${env}\"HTTP_PROXY=${http_proxy}\" "
        [ ! -z "${https_proxy}" ] && env="${env}\"HTTPS_PROXY=${https_proxy}\" "
        [ ! -z "${no_proxy}" ]    && env="${env}\"NO_PROXY=${no_proxy}\""
        mkdir ${docker_service_dir}
        echo -e "[Service]\n${env}" > ${docker_service_dir}/${docker_proxy_conf}
    fi
}

function enable_and_start() {
    INFO 'Enabling and starting service...'
    systemctl daemon-reload && \
    systemctl enable docker.service && \
    systemctl start docker.service
    [ $? -ne 0 ] && ERROR 'There was a problem enabling or starting the docker service!' && exit 1
    # Display Docker info
    INFO 'Checking Docker info...'
    docker info
```

```
    [ $(docker info | grep 'Storage Driver: btrfs' | wc -l) -eq 1 ] && INFO 'Docker info appears to be as
expected. Default storage Btrfs is correct!'
    # Add docker user to the docker group
    /usr/sbin/usermod -aG docker docker
    su - docker -c 'docker run hello-world'
}

function configure_internal_sles_repo() {
    # Disable SLES default repos first
    SLES_repos=$(zypper repos | grep "SLES12" | grep -v '#' | cut -f1 -d' ')
    for repo in ${SLES_repos}; do
        zypper modifyrepo -dr ${repo}
    done
    # Add internal repo
    zypper addrepo ${internal_sles_repo} Internal_repo
}

function additional_config() {
    ## Allow root logins using SSH
    # Replace any entry of "PermitRootLogin..." or "#PermitRootLogin..." with "PermitRootLogin no"
    sed -i  's/^[\#]\?PermitRootLogin.*/PermitRootLogin no/g' /etc/ssh/sshd_config
    # If nothing was found in the step above then just add the line to the sshd config file
    [ $(grep "PermitRootLogin no" /etc/ssh/sshd_config | wc -l) -eq 0 ] && echo "PermitRootLogin no" >>
/etc/ssh/sshd_config
    # Restart the service
    service sshd restart
}

#
#
## Main
if [ ${#internal_sles_repo} -gt 0 ]; then
    configure_internal_sles_repo
fi
install_docker
enable_and_start
additional_config
exit 0
```

Appendix C – NIC Teaming script for SUSE 12

```
#!/bin/bash

# Copyright 2017 Hewlett Packard Enterprise Development LP Licensed under the
# Apache License, Version 2.0 (the "License"); you may not use this file except
# in compliance with the License. You may obtain a copy of the License at
# http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law
# or agreed to in writing, software distributed under the License is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

function initialize() {
   skip_team=0
}

function validate_macs() {
    if [ `ip link | grep -i "$1" | wc -l` -ne 1 ]; then
        echo "ERROR : MAC address $1 not found, skipping this team"
        skip_team=1
    fi
}

function create_team() {
    # Get interface names
    int_name1=$(echo `ip link | grep -i -B 1 "${mac1}" | head -1 | cut -d':' -f2`)
    int_name2=$(echo `ip link | grep -i -B 1 "${mac2}" | head -1 | cut -d':' -f2`)

    # Define configuration files
    conf_path="/etc/sysconfig/network"
    conf_int1="${conf_path}/ifcfg-${int_name1}"
    conf_int2="${conf_path}/ifcfg-${int_name2}"
    conf_int1_bak="/tmp/ifcfg-${int_name1}.bak"
    conf_int2_bak="/tmp/ifcfg-${int_name2}.bak"

    # Define configuration file for NIC team
    cfg_team="${conf_path}/ifcfg-${team_name}"

    # Find out how these interfaces boot (none, static, dhcp) and if they have an ip
    if [ -f "${conf_int1}" ]; then
        bootproto1=$(grep -i BOOTPROTO "${conf_int1}" | cut -d'=' -f2 | tr '[:upper:]' '[:lower:]')
        ip1=$(echo `ip a | grep -i -A 2 ${int_name1} | grep "inet " | awk -F' ' '{ print $2 }'`)
        team_ip=${ip1}
        #main_int=${int_name1}   # to be used in the team config file as DEVICE 0
        #second_int=${int_name2} # to be used in the team config file as DEVICE 1
    elif [ -f "${conf_int2}" ]; then
        bootproto2=$(grep -i BOOTPROTO "${conf_int2}" | cut -d'=' -f2 | tr '[:upper:]' '[:lower:]')
        ip2=$(echo `ip a | grep -i -A 2 ${int_name2} | grep "inet " | awk -F' ' '{ print $2 }'`)
        team_ip=${ip2}
        #main_int=${int_name2}   # to be used in the team config file as DEVICE 0
        #second_int=${int_name1} # to be used in the team config file as DEVICE 1
    #else
        #main_int=${int_name1}   # to be used in the team config file as DEVICE 0
        #second_int=${int_name2} # to be used in the team config file as DEVICE 1
    fi

    # create the team network file. Default is DHCP but if a static IP is found, use static
    if [[ ! -z "${ip1}" && "${bootproto1}" == "static" ]]; then
        team_bootproto="static"
```

```
        team_ip=${ip1}
    elif [[ ! -z "${ip2}" && "${bootproto2}" == "static" ]]; then
        team_bootproto="static"
        team_ip=${ip2}
    else
        team_bootproto="dhcp"
    fi

    # Stop slave interfaces
    wicked ifdown ${int_name1}
    wicked ifdown ${int_name2}

    # If configuration files exist for the slave interfaces, then remove them (or move to /tmp/<filename>.bak)
    [ -f "${conf_int1}" ] && mv "${conf_int1}" "${conf_int1_bak}"
    [ -f "${conf_int2}" ] && mv "${conf_int2}" "${conf_int2_bak}"

cat > ${cfg_team} << EOF
STARTMODE=auto
BOOTPROTO=${team_bootproto}
TEAM_RUNNER="roundrobin"
TEAM_PORT_DEVICE_0=${int_name1}
TEAM_PORT_DEVICE_1=${int_name2}
TEAM_LW_NAME="ethtool"
TEAM_LW_ETHTOOL_DELAY_UP="10"
TEAM_LW_ETHTOOL_DELAY_DOWN="10"
EOF

if [ ! -z ${team_ip} ]; then
cat >> ${cfg_team} << EOF
IPADDRESS=${team_ip}
EOF
fi


# Details of Wicked configuration file
#wicked show-config ## commented out -> too much unneeded output at this stage

# Start Network Teaming device
wicked ifup all ${team_name}

# Restart the network to start up cleanly
service network restart

# Check the status of Network Teaming device
wicked ifstatus --verbose ${team_name}
teamdctl ${team_name} state
}

## Main

# Check if the nic_teaming variable is empty
nic_teaming="@nic_teaming@"
[ ${#nic_teaming} -eq 0 ] && exit 0

# Read the interfaces into an array
readarray ifaces_list < <(echo "${nic_teaming}")

# Install teamd packages
zypper --non-interactive --no-gpg-checks install libteam-tools libteamdctl0 libteamdctl0 python-libteam
```

```
# Go throgh the list of teams to be created
for t in "${ifaces_list[@]}"; do
    initialize
    team_name=$(echo ${t} | cut -d',' -f1)
    mac1=$(echo ${t} | cut -d',' -f2)
    mac2=$(echo ${t} | cut -d',' -f3)
    validate_macs ${mac1} ${mac2}
    [ ${skip_team} -eq 0 ] && create_team
done

exit 0
```

Appendix D – Troubleshooting

If the ICsp Job fails, please check the following for more information:

- Check the job log by clicking the Logs link in the Jobs details and check for the step that failed.
- Check the console on the target server for errors during the installation.
- Make sure you have included the Media Server hostname/IP address in the `no_proxy` custom attribute.
- Make sure that the DHCP server on the deployment network is reachable. Refer to HPE Insight Control Server Provisioning Installation Guide.
- Make sure you are using the amended AutoYaST file provided either on GitHub or Appendix A

If the Docker Installation fails:

- If you are behind a proxy, make sure that the proxy host and port are set and that the values are correct.
- Check the Docker URL in the custom attributes. You can test it in a browser to make sure it is correct and reachable.

If the NICs team is not created:

- Check that the MAC addresses provided in the custom attributes match the ones in the server.
- Please note that the deployment network cannot be teamed. Only secondary NICs can be teamed.

Resources and additional links

HPE Servers
hpe.com/servers

HPE Storage
hpe.com/storage

HPE Networking
hpe.com/networking

HPE Insight Control Server provisioning
https://www.hpe.com/us/en/software/servers-insight-control.html

HPE Insight Control Server Provisioning documentation
http://h17007.www1.hpe.com/us/en/enterprise/servers/solutions/info-library/index.aspx?cat=insightmanagement&subcat=ic

Docker EE
https://docs.docker.com/enterprise/

To help us improve our documents, please provide feedback at hpe.com/contact/feedback.

Sign up for updates

Hewlett Packard Enterprise