Hewlett Packard
Enterprise

HPE Insight Control Server Provisioning

How to Create an OS Build Plan for Installing Docker EE on RHEL 7

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.

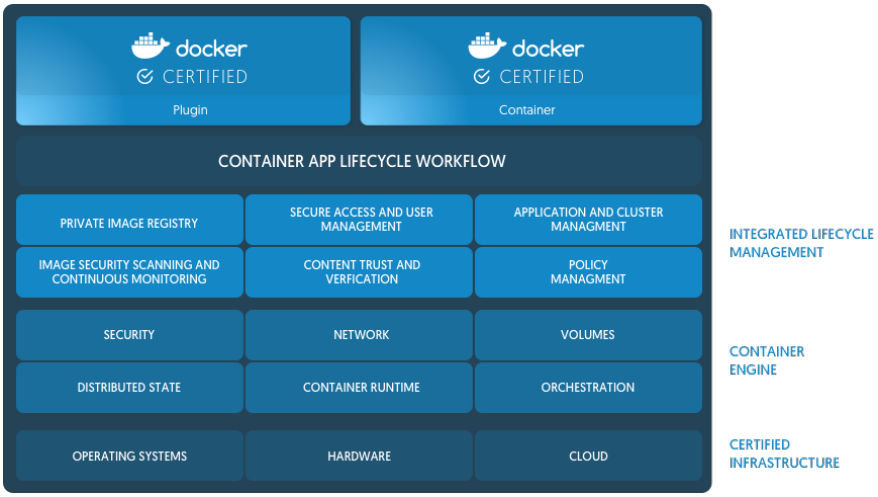

Figure 1. Docker Enterprise Edition

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

HPE Insight Control server provisioning overview

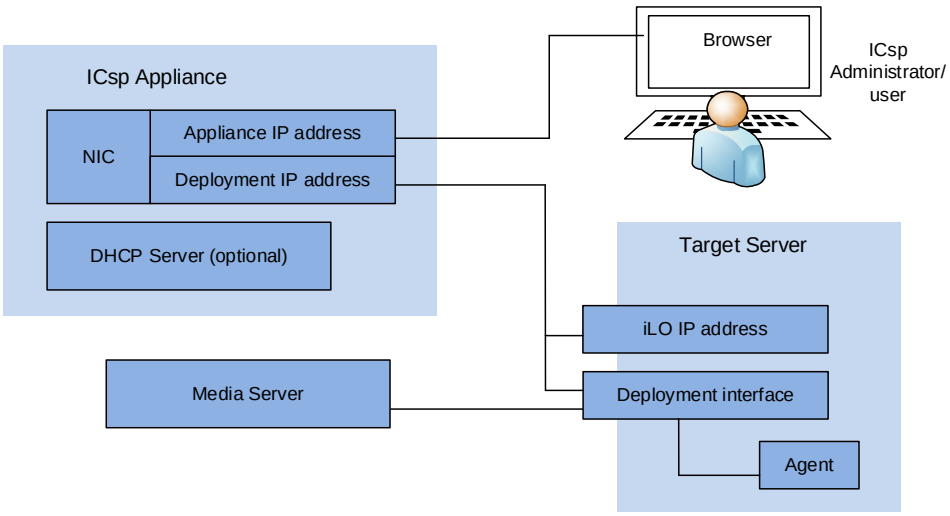

Figure 2. ICsp appliance architecture

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 RHEL 7.2. 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 **rhel** folder.

OS Build Plan for Docker Enterprise Edition installation on RHEL 7

Prerequisites

- ICsp is installed and configured to deploy RHEL 7 OS.
- This build plan requires either a Red Hat Network account to download files from the Internet or access to an internal RHEL repository.
- 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 RHEL 7.2 to ICsp media server:

a. Create a directory on the media server share for RHEL 7.2. 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 RHEL 7.2 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 RHEL 7.2 installation media to the Media server `rhel72-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.

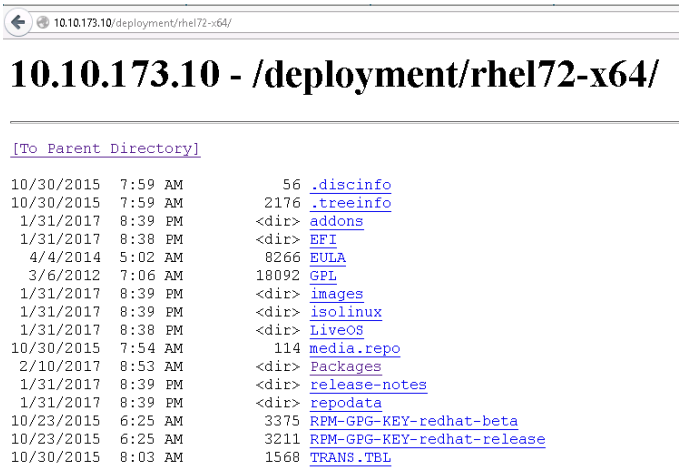

Figure 4. ICsp Media Server

4. Create a Custom Build Plan as follows:
    a. Select the ICsp provided "ProLiant OS - RHEL 7.2 x64 Scripted Install" OS Build Plan
    b. Save the OS Build Plan with a new name, for example "ProLiant OS - RHEL 7.2 x64 Scripted Install with Docker"

`Actions` → `Save as`

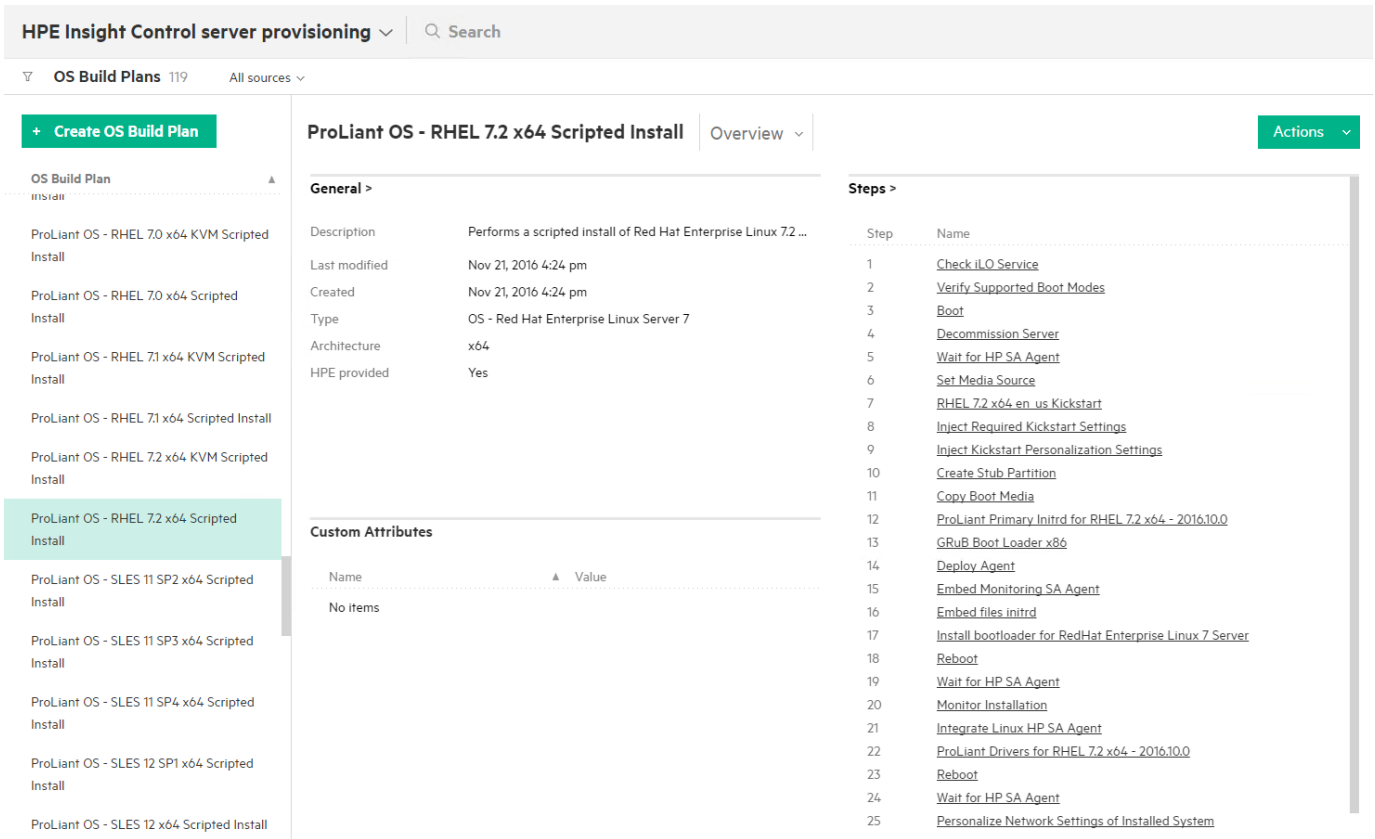

Figure 5. RHEL 7.2 OS Build Plan

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 cover different Linux systems you will need to add the 'rhel' 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/rhel'. If your license only covers RHEL 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. **rhn_user (Optional):** Red Hat Network user (needed to register the system with Red Hat). Required if no `internal_rhel_repo` is specified.

   d. **rhn_pw (Optional):** Red Hat Network password (needed to register the system with Red Hat). Required if no `internal_rhel_repo` is specified.

   e. **internal_rhel_repo (Optional):** URL pointing to an internal RHEL 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 RHN registration will be skipped. Required if no `rhn_user/rhn_password` are provided. Please make sure to specify the protocol (e.g. `http://`) when specifying the URL.

   f. **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.

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

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

   i. **no_proxy (Optional):** Comma-separated list of IP addresses or server names where the proxy should not be used for.

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

- Creation of a `docker` user belonging to the `wheel` group, able to run Docker commands.
- Definition of `/boot` and `/boot/efi` to support UEFI-based AMD64 and Intel 64 systems.
- Creation of two physical volumes and two volume groups (one for the system and one for Docker) on the server under deployment. The relevant logical volumes for the Docker volume group will be created at a later stage during the Docker installation step.

To create a Custom Configuration Kickstart File:

`Configuration Files → Select RHEL 7.2 x64 en_us Kickstart → Actions → Save As <<RHEL 7.2 x64 en_us for Docker Kickstart>>`

Edit the `<<RHEL 7.2 x64 en_us for Docker Kickstart>>` configuration file (kickstart)

`Configuration Files → Select <<RHEL 7.2 x64 en_us for Docker Kickstart>> → Actions →Edit`

Download the kickstart file named "`kickstart_RHEL7.ks`" 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 kickstart configuration file in step 7 of the OS Build Plan with the newly created custom kickstart 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:
   - Registration of the system with the Red Hat Network (required to download dependencies).
   - Download and installation of Docker EE packages.
   - Configuration of the storage driver for Docker (LVM devicemapper), including the creation of the logical volumes.
   - 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 named "`install_docker_on_RHEL7.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 at the end of the OS Build Plan (step 27)

   `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_RHEL7.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 28)

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 – RHEL7 Kickstart file for Docker

```
# 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.
lang en_US.UTF-8
keyboard us
timezone --utc America/Chicago
text
install
skipx
network  --bootproto=dhcp

authconfig --enableshadow --enablemd5
rootpw --iscrypted "@encrypted_root_password:$1$7z4m7f1z$wliShMhVv2HuCAPmuiQzV1@"
user --groups=wheel --homedir=/home/docker --name=docker --
password="@encrypted_root_password:$1$7z4m7f1z$wliShMhVv2HuCAPmuiQzV1@" --iscrypted --gecos="docker"

zerombr
clearpart --all --initlabel
part /boot --fstype xfs --size=300
part /boot/efi --fstype efi --size=300
part swap --size=1024
part pv.01 --size=15000 --grow
part pv.02 --size=10000 --grow
volgroup vg_root pv.01
logvol  /  --vgname=vg_root --size=10000 --name=lv_root
volgroup vg_docker pv.02

bootloader --append="@kernel_arguments: @" --location=mbr

# Disable firewall and selinux for SPP components
firewall --disabled
# Port 1002 is needed for agent communication if the firewall is enabled
# firewall --enable --port=1002:tcp
selinux --disabled

%pre
# Set FCOEwait Custom Attribute to 120 seconds when deploying to FCOE SAN through
# Broadcom CNA to allow FCOE driver to load correctly
sleep @FCOEwait:0@
%end

%packages
@Base
# Needed to ensure Mellanox driver installed when required
#kmod-mlnx-ofa_kernel

# Components listed below are needed for mount to media server for HPSUM installation
keyutils
libtalloc
cifs-utils

# Components listed below are needed to run HPSUM and SPP components
expat.i686
```

```
expect
fontconfig.i686
freetype.i686
libICE.i686
libSM.i686
libuuid.i686
libXi.i686
libX11.i686
libXau.i686
libxcb.i686
libXcursor.i686
libXext.i686
libXfixes.i686
libXi.i686
libXinerama.i686
libXrandr.i686
libXrender.i686
zlib.i686
libgcc.i686
libstdc++.i686
libhbaapi
make
net-snmp
net-snmp-libs
%end
```

Appendix B – Docker Enterprise Edition installation script for RHEL 7

```
#!/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@"
```

```
RHEL_version_string=7 # only supported version
internal_rhel_repo="@internal_rhel_repo@"
docker_version="@docker_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 rhn_registration() {
    export TMPPYTHONPATH=$PYTHONPATH
    unset PYTHONPATH # RHN subscription-manager won't work with this one set up from ICsp
    subscription-manager config --server.proxy_hostname=${proxy_hostname} \
                                --server.proxy_port=${proxy_port}
    subscription-manager register --auto-attach --username=@rhn_user@ --password=@rhn_pw@
    if [ $? -ne 0 ]; then
        # Double checking the RHN registration just in case
        if [ $(subscription-manager status | grep Current | wc -l) -ne 1 ] || [ $(subscription-manager list |
grep Subscribed | wc -l) -ne 1 ]; then
            ERROR 'There was a problem registering this system with the Red Hat Network!'
            exit 1
        fi
    fi
    export PYTHONPATH=$TMPYTHONPATH
}

function install_docker() {
    INFO "Storing Docker EE yum variables"
    echo "${docker_repo}" > /etc/yum/vars/dockerurl
    echo "${RHEL_version_string}" > /etc/yum/vars/dockerosversion

    INFO 'Cleaning up yum cached data...'
    yum -y -q -e 0 clean all
    [ $? -ne 0 ] && ERROR 'There was a problem running yum clean all!' && exit 1

    INFO 'Installing yum utils and setting up the stable repo...'
    yum  -y -q -e 0 install yum-utils deltarpm && \
    yum-config-manager -y -q -e 0 --add-repo ${docker_repo}/docker-ee.repo
    [ $? -ne 0 ] && ERROR 'There was a problem setting the Docker repository!' && exit 1

    INFO 'Installing Docker...'
    yum -y -q -e 0 makecache fast && \
    yum -y -q -e 0 install "docker-ee-${docker_version}*"
    [ $? -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 config_storage() {
# This function assumes that the VG has already been created via the kickstart file

INFO 'Creating PV and LVs...'
# Create volume group and logical volumes
lvcreate --wipesignatures y -n lv_thinpool vg_docker -l 95%VG
lvcreate --wipesignatures y -n lv_thinpoolmeta vg_docker -l 1%VG
lvconvert -y --zero n -c 512K --thinpool vg_docker/lv_thinpool --poolmetadata vg_docker/lv_thinpoolmeta

# Configure autoextend settings
INFO 'Configuring autoextend settings...'
cat > '/etc/lvm/profile/docker-thinpool.profile' << EOF
activation {
thin_pool_autoextend_threshold=80
thin_pool_autoextend_percent=20
}
EOF

# Apply lvm profile
INFO 'Applying LVM profile...'
lvchange --metadataprofile docker-thinpool vg_docker/lv_thinpool
# Verify lv is monitored
if [ $(lvs -o+seg_monitor | grep 'monitored' | wc -l) -eq 1 ]; then
    INFO 'Monitoring is set up OK...'
else
    ERROR 'There was an error setting the monitoring...'
    exit 1
fi

# Configure Docker daemon
INFO 'Configuring the Docker daemon'
mkdir /etc/docker/
cat > '/etc/docker/daemon.json' << EOF
{
  "storage-driver": "devicemapper",
   "storage-opts": [
     "dm.thinpooldev=/dev/mapper/vg_docker-lv_thinpool",
     "dm.use_deferred_removal=true",
     "dm.use_deferred_deletion=true"
   ]
}
EOF
}

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: devicemapper' | wc -l) -eq 1 ] && [ $(docker info | grep 'Pool Name:
vg_docker-lv_thinpool' | wc -l) -eq 1 ] && INFO 'Docker info appears to be as expected. Storage configured
correctly!'
```

```
    su - docker -c 'docker run hello-world'
}

function configure_internal_RHEL_repo() {
# Disable RedHat managed repo
subscription-manager config --rhsm.manage_repos=0
# Add internal repo
repo="/etc/yum.repos.d/internal.repo"
cat > ${repo} << EOF
[Internal_RHEL_repo]
name=Internal_RHEL_repo
gpgcheck=0
enabled=1
baseurl=${internal_rhel_repo}
EOF
}

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 using an internal repo we assume the internet access is restricted and don't register the system with RHN
if [ ${#internal_rhel_repo} -gt 0 ]; then
    configure_internal_RHEL_repo
else
    rhn_registration
fi
install_docker
config_storage
enable_and_start
additional_config
exit 0
```

Appendix C – NIC Teaming script for RHEL 7

```
#!/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.

#
## Functions

function create_team() {
    echo "Creating team: ${team_name}"
    # Get interface names
    int_name1=$(echo `ip link | grep -i -B 1 "${mac1}" | head -1 | cut -d':' -f2`)
    int_name2=$(echo `ip link | grep -i -B 1 "${mac2}" | head -1 | cut -d':' -f2`)
    # Get configuration files
    cfg_int1="/etc/sysconfig/network-scripts/ifcfg-${int_name1}"
    cfg_int2="/etc/sysconfig/network-scripts/ifcfg-${int_name2}"
    cfg_team="/etc/sysconfig/network-scripts/ifcfg-team-${team_name}"
    # Find out how these interfaces boot (none, static, dhcp)
    bootproto1=$(grep -i BOOTPROTO "${cfg_int1}" | cut -d'=' -f2 | tr '[:upper:]' '[:lower:]')
    bootproto2=$(grep -i BOOTPROTO "${cfg_int2}" | cut -d'=' -f2 | tr '[:upper:]' '[:lower:]')
    # Check if they have an IP address configured
    ip1=$(echo `ip a | grep -i -A 2 ${int_name1} | grep "inet " | awk -F' ' '{ print $2 }' | cut -d'/' -f1`)
    ip2=$(echo `ip a | grep -i -A 2 ${int_name2} | grep "inet "| awk -F' ' '{ print $2 }' | cut -d'/' -f1`)
    # Create team using nmcli
    nmcli connection add type team ifname ${team_name}
    nmcli con add type team-slave con-name ${team_name}-port1 ifname ${int_name1} master ${team_name}
    nmcli con add type team-slave con-name ${team_name}-port2 ifname ${int_name2} master ${team_name}
    # Default is DHCP but if we find a static IP we'll use this one instead
    if [[ ! -z "${ip1}" && "${bootproto1}" == "static" ]]; then
        echo "Using static IP $ip1 for the team NIC."
        grep -Ei '^DNS|^IPADDR|^NETMASK|^GATEWAY|^BOOTPROTO' ${cfg_int1} >> ${cfg_team}
        sed -i '/BOOTPROTO=dhcp/d' ${cfg_team}
    elif [[ ! -z "${ip2}" && "${bootproto2}" == "static" ]]; then
        echo "Using static IP $ip2 for the team NIC."
        grep -Ei '^DNS|^IPADDR|^NETMASK|^GATEWAY|^BOOTPROTO' ${cfg_int2} >> ${cfg_team}
        sed -i '/BOOTPROTO=dhcp/d' ${cfg_team}
    else
        echo "Using DHCP for the team NIC."
    fi
    # Activate the team and restart the network
    nmcli connection up ${team_name}-port1
    nmcli connection up ${team_name}-port2
    nmcli connection up team-${team_name}
    nmcli connection show
    systemctl restart network
}

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 initialize() {
    skip_team=0
}

#
## 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 package in case it's missing
yum -y install teamd

# Go through 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 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 kickstart file provided either on GitHub or Appendix A

If the RHEL Registration fails:

- Check that the RHEL username and password specified in your custom attributes are correct.
- If a local repository exists, verify the path to the repository in a browser as the local repository gets precedence when installing Docker EE

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