The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Presenters

Christoph Doerbeck	Principal Solutions Architect, RHCA
Matt St. Onge	Senior Partner Solutions Architect, RHCE
Eddie Chen	Senior Solutions Architect, RHCA
Joe Hackett	Alliance Solutions Architect, RHCSA
Rob Wilmoth	Principal Solutions Architect, RHCDS

and guests...

Introduction & Prerequisites

This lab is geared towards Red Hat Enterprise Linux administrators.

During this session attendees will learn:

- Introduction to automation with **ansible** and **system roles**
- Service startup and management with `systemd`
- Performance tuning & monitoring with performance **co-pilot**, **tuned** and **numad**
- Storage management and optimization with with **ssm** and **VDO**
- Network interface management with **nmcli** (Network Manager CLI)
- Dynamic firewall with `firewalld`
- System administration with **Cockpit**
- System Security Audit with **oscap-workbench** (OpenSCAP)
- System Backup and Recovery with **rear** (Relax and Recover)

To accomplish this, attendees should have a Red Hat Certified Systems Administrator (RHCSA) certification or equivalent experience.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Environment Overview

Before you begin, please take note of the lab environment.

Each station has a laptop which will provide you with access to Virtual Machines running the public cloud. You can use SSH to connect to the workstation for the majority of the exercises, however exercises that use a GUI will require connectivity to the VM console via the link provided.

Important Credentials

The student password is 'student' (all lower case)
The root password is 'redhat' (all lower case)

Last Minutes Changes & Preparations

1) Get a copy of this document @

```
# sudo -i
# git clone https://github.com/mattstonge/RHEL7lab2018.git
# cd RHEL7lab2018
```

2) As root, run the ansible playbook in root's home directory on VM workstation

```
# ansible-playbook -b ~/RHEL7lab2018/RHEL7-Lab-Setup.yml
```

Bash Skills Refresher

```
ls /dev/vd{a,b,c}          ...  ls /dev/vda /dev/vdb /dev/vdc

ls /dev/vda{1..3}          ...  ls /dev/vda1 /dev/vda2 /dev/vda3

Ctrl-R                     ...  Search command line history
```

Other Handy Tools

```
screen                     ... ctl+a (c)reate (n)ext (r)esume
```

Introduction to Ansible and System Roles

Ansible is an open source automation platform that uses a simple *automation language* that describes an IT application infrastructure within Ansible Playbooks. Ansible is also an *automation engine* that runs Ansible Playbooks.

You can utilize Ansible in many ways:

- Device configuration management
- Application deployment
- Provisioning
- Continuous delivery
- Security and compliance
- Orchestration

`ansible` is the primary command line tool to initiate a ad hoc command. `ansible-playbook` is the command line tool to run Ansible Playbooks. The syntax used for configuring Ansible Playbooks is written in YAML.

The Ansible Engine product channel is provided as a convenience with the Red Hat Enterprise Linux subscription as a *unsupported dependency* for the implementation of RHEL System Roles. However, support of Ansible Engine product by the RHEL subscription is limited to the context of, and modules used by, RHEL System Roles. A full support subscription is available for Ansible Engine and Ansible Tower which are also able to use the RHEL System Roles.

Ansible Concepts

There are two types of machines in the Ansible architecture: *control nodes* and *managed hosts*. Ansible is installed and run from a control node(s), which also has copies of Ansible configuration files.

Managed hosts are listed in an *inventory* file, which can also organize those systems into groups for easier management. The inventory can be defined in a static text file (INI), or via a dynamic file determined by scripts that get information from external files.

Ansible users create high-level *plays* to ensure a host or group of hosts are in a particular state. A play performs several *tasks*, in order, on a host or set of hosts. A file that contains one or more plays is called a *playbook*.

Each task runs a *module*, a small piece of code that is written in Python, Powershell, or some other language with specific arguments. Each module is a tool in your toolkit. By default, Ansible ships with hundreds of these modules that perform a wide variety of tasks. For example, modules can install software, make API calls, or act on system files.

RHEL System Roles Concepts

Red Hat Enterprise Linux (RHEL) 7.4 introduced RHEL System Roles as a Technology Preview. The RHEL System Roles are a collection of Ansible roles and modules that provide a stable and consistent configuration interface to remotely manage RHEL 6.9 and later versions of Red Hat Enterprise Linux (including RHEL 7.5 as a Technology Preview). This effort is based on development of the Linux System Roles upstream project.

RHEL System Roles will provide the user a consistent user interface to provide settings to a given subsystem that is abstract from any one particular implementation or configuration. Thereby, making generic system configurations easy to maintain with consistent automation techniques while maintaining modularity.

The initial set of roles includes:

- kdump
- postfix
- network
- selinux
- timesync

The RHEL System Roles are provided in the RHEL Extras channel which provides customers access to select, and quickly evolving system roles.

Typically Ansible Engine and the RHEL System Roles only need to be installed on a single, or few, control node(s) which can then be used to manage or configure client nodes. While the roles will likely work with earlier versions, compatibility is only tested against RHEL 6.9 and later clients.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Getting Started

The `rhel-system-roles` and `ansible` RPM packages are provided in the RHEL Extras channel. The Extras channel should be enabled by default.

To install Ansible and RHEL System Roles, log into the `workstation` as `root` and run the following commands:
Note: `ansible` RPM may already have been installed.

```
# yum install rhel-system-roles ansible -y
…<output truncated>...
Package ansible-2.4.2.0-2.el7.noarch already installed and latest
version
Resolving Dependencies
--> Running transaction check
---> Package rhel-system-roles.noarch 0:0.6-3.el7 will be
installed
--> Finished Dependency Resolution

Installed:
  rhel-system-roles.noarch 0:0.6-3.el7
…<output truncated>...
```

RHEL System Roles documentation is located in
`/usr/share/doc/rhel-system-roles/SUBSYSTEM/`

Ansible Roles documentation is located in
`/usr/share/ansible/roles/rhel-system-roles.SUBSYSTEM/`

Examples

Ad Hoc Commands

Now that we have installed Ansible and RHEL System Roles and a general understanding of what Ansible is, let's create and modify the initial configuration file to run a few modules against our *managed hosts* using our *inventory* file.

1. Create a Ansible configuration file in your working directory to override the default configuration file.
   ```
   # cd ~
   # vi .ansible.cfg
   ```

   The .cfg file should contain the following:
   ```
   [defaults]
   inventory = /root/inventory
   remote_user = root
   ask_pass = false

   [privilege_escalation]
   become = true
   become_method = sudo
   become_user = root
   become_ask_pass = false
   ```

   Next, create an inventory file for our managed hosts:
   ```
   # vi inventory
   ```

   The `inventory` file should contain the following:
   ```
   [myself]
   workstation.example.com

   [servers]
   server1.example.com
   server2.example.com
   ```

2. Let's send the **ping** module to our **servers** group to verify basic connectivity:
   ```
   # ansible servers -m ping
   ```

   You should get the following output:

   ```
   server2.example.com | SUCCESS => {
       "changed": false,
       "ping": "pong"
   }
   server1.example.com | SUCCESS => {
       "changed": false,
       "ping": "pong"
   ```

```
    }
```

3. Let's now send the **setup** module with arguments to gather facts about our network interface (eth0) on server1.example.com:

```
# ansible -m setup -a 'filter=ansible_eth0' server1.example.com
…<output truncated>...
server1.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "features": {
                "busy_poll": "off [fixed]",
                "fcoe_mtu": "off [fixed]",
…<output truncated>...
"ipv4": {
            "address": "10.0.0.11",
            "broadcast": "10.0.0.255",
            "netmask": "255.255.255.0",
            "network": "10.0.0.0"
              },
              "ipv6": [
                  {
                    "address": "fe80::2ec2:60ff:fe02:d1d5",
                    "prefix": "64",
                    "scope": "link"
                  }
```

Writing A Playbook With Roles

Let's now define the role content for use with Ansible Playbooks that introduces RHEL System Roles. In this case we'll create a role called "motd".

1. Create the directory structure for the "motd" role:

```
# mkdir -p ~/roles/{motd,defaults,files,handlers,tasks,templates}
# touch roles/motd/defaults/main.yml
# touch roles/motd/tasks/main.yml
# touch roles/motd/templates/motd.j2
```

2. **roles/motd/defaults/main.yml** file should include:

```
---
# Tasks file for MOTD

- name: Deliver MOTD file
  template:
    src: templates/motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

3. **roles/motd/templates/motd.j2** file should include:

```
This is the system {{ ansible_hostname }}.

Today's date is: {{ ansible_date_time.date }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

4. **roles/motd/defaults/main.yml** file should include:

```
---
system_owner: root@workstation.example.com
```

5. Create the playbook "use-motd-role.yml" to include the "motd" role reference:

   # vi use-motd-role.yml

```
---
- name: Use MOTD role playbook
  hosts: server1.example.com
  user: student
  become: true
  roles:
    - motd
```

6. Run the playbook and you will see the following:

```
PLAY [Use MOTD role playbook]
**********************************************************************

TASK [Gathering Facts]
**********************************************************************
ok: [server1.example.com]

TASK [motd : Deliver MOTD file]
**********************************************************************
changed: [server1.example.com]

PLAY RECAP
**********************************************************************
server1.example.com : ok=2    changed=1    unreachable=0
failed=0
```

When the playbook was executed the tasks continued to perform because the role itself can be identified by the `roles` configuration within the playbook.

7. For example, you should also see the following MOTD when logging into server1.example.com:

```
…<output truncated>...
This is the system server1.example.com.

Today's date is: 2018-04-22.

Only use this system with permission.
You can ask root@workstation.example.com for access.
…<output truncated>...
```

Introduction to systemd

`systemd` is the "`init`" system in RHEL 7 and replaces Upstart (the SysV "init" system used in prior releases of RHEL). `systemd` is more than a facility to bring up user space, it is a system manager that offers:

- service parallelization,
- socket and D-Bus activation,
- on-demand starting of services,
- tracks services and child processes via cgroups,
- and much more.

`systemctl` is the primary command line tool for interacting with `systemd`. Although the `service` and `chkconfig` commands still function, it is highly recommended to use `systemctl` to take full advantage of `systemd`'s capabilities.

Getting started

Log into the `workstation` as `root`, and collect some data about the boot process.

```
# systemd-analyze
Startup finished in 329ms (kernel) + 619ms (initrd) + 2.816s (userspace) =
3.766s
```

```
# systemd-analyze blame
631ms kdump.service
605ms firewalld.service
478ms postfix.service
```
…<output truncated>...

To streamline the boot process for something like a cloud image, we can easily learn the "cost" of some of the default services. Using standard disk partitions (not LVM) and disabling services like `postfix`, `kdump`, and `rsyslog` will easily get the boot process to complete in under two seconds.

Note: Doing this would probably be a bad idea for a traditional production server where services like kernel crash dumps and logging may be important.

Core Concepts

Units

The fundamental building block that systemd manages is called a "unit". A "unit" can describe different types of objects, but the most common type is a "service".

A "unit file" is the configuration file that describes a unit and tells systemd what dependencies exist and how to start, stop and monitor the object.

"unit files" are stored in 2 different directories. One location is reserved for the default configurations as shipped by Red Hat and the other is for customization by the local administrators.

Red Hat unit files: /usr/lib/systemd/system/...
Customizations: /etc/systemd/system/...

Targets

systemd has a concept similar to SysV init runlevels, called targets. systemd will boot to the "default target" which can be configured using the systemctl set-default command. Some common targets and their equivalent SysV runlevels are:

multi-user.target == runlevel 3
graphical.target == runlevel 5

Let's view the current default target.

```
# systemctl get-default
graphical.target
```

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Services

As mentioned above, systemd has another concept called a service. A service is a type of unit which defines the traditional daemon or process. Now let us look at what services are running on the system.

```
# systemctl -t service
UNIT              LOAD   ACTIVE SUB     DESCRIPTION
abrt-ccpp.service loaded active exited  Install ABRT coredump
hook
abrt-oops.service loaded active running ABRT kernel log watcher
...<output truncated>...
```

Next let's view all of the services available (ie: everything installed, running or not) on the system. The following command is similar to the older chkconfig --list as it will show both enabled and disabled services.

```
# systemctl list-unit-files -t service
UNIT FILE                           STATE
abrt-ccpp.service                   enabled
abrt-oops.service                   enabled
...<output truncated>...
arp-ethers.service                  disabled
atd.service                         enabled
auditd.service                      enabled
autovt@.service                     disabled
bluetooth.service                   enabled
...<output truncated>...
```

The state will be **enabled**, **disabled**, **static**, or **masked**. Static indicates that the unit file does not contain an "install" section used to enable the unit. In this case, the unit typically performs a one-off action or is used as a dependency of another unit and should not be run by itself.

13

Examples

Installing a LAMP stack

Now that we have a good idea of what's installed on our system, let's get a basic lamp stack up and running.

1. Install the following packages: `httpd`, `mariadb-server`, `mariadb`, `php`, `php-mysql`

   ```
   # yum install -y httpd mariadb-server mariadb php php-mysql
   ```

2. Enable these services to start on boot: `httpd mariadb`

   `systemctl` allows us to "glob" units, so enable `httpd` and `mariadb` in a single command.

   ```
   # systemctl enable httpd mariadb
   ln -s '/usr/lib/systemd/system/httpd.service'
   '/etc/systemd/system/multi-user.target.wants/httpd.service'
   ln -s '/usr/lib/systemd/system/mariadb.service'
   '/etc/systemd/system/multi-user.target.wants/mariadb.service'
   ```

3. Start the services

   ```
   # systemctl start httpd mariadb
   ```

4. View the status.

   ```
   # systemctl status httpd mariadb
   httpd.service - The Apache HTTP Server
      Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled)
      Active: active (running) since Tue 2016-05-31 21:26:35 EDT; 5s ago
     Process: 28102 ExecStop=/bin/kill -WINCH ${MAINPID}
   (code=exited, status=0/SUCCESS)
    Main PID: 28124 (httpd)
   ```
   ...<output truncated>...

   Take a moment to review the output of these commands.

Customizing Unit Files

systemd controls more than daemons or services. For this lab, we will primarily be working with service units but it's important to know that systemd is handling the dependencies between other types: sockets, timers, mounts, swap, slices, etc.
Unit files that ship with the RHEL are stored under /usr/lib/systemd/system.

Custom unit files, changes or extensions are stored under /etc/systemd/system
(or /run/systemd/system for runtime changes that won't persist).

While the defaults for unit files won't need to be altered most of the time, there will be circumstances where changing the defaults is quite beneficial. These could include hardware or software watchdog monitoring, tunings, resource management, or many other reasons.

1. Create a drop-in configuration file to extend the default `httpd.service` unit.

```
# mkdir /etc/systemd/system/httpd.service.d
# cd /etc/systemd/system/httpd.service.d
# vim 50-httpd.conf

### add the following to the config file

[Service]
Restart=always
CPUShares=2048
OOMScoreAdjust=-1000
```

2. Save the config file, exit the editor, and notify `systemd` of the changes:

```
# systemctl daemon-reload
# systemctl status httpd
```

httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Drop-In: /etc/systemd/system/httpd.service.d
        └─50-httpd.conf
...output truncated...

Notice that `systemctl` status displays that the unit has been extended with a **drop-in** file.

OOMScoreAdjust is use by the Out Of Memory killer and is an integer between -1000 (to disable OOM killing for this process) and 1000 (to make killing of this process under memory pressure very likely).

Using what you learned in the last step, extend the **mariadb.service** unit with **Restart=always**. Verify that systemd recognizes the settings and test the policy using killall mysqld.

CGroup Resource Management

`systemd` will setup a single root cgroup hierarchy under `/sys/fs/cgroup/systemd` and use cgroups to track services and their child processes. We need to understand the following unit types:

- **Slice:** A unit used to build the cgroup hierarchy. This is essentially how the system is "carved up."
- **Scope:** A transient, organizational unit that groups processes that have registered with `systemd`. User sessions, VMs, and containers are exposed as scopes for resource management.
- **Service:** A daemon or group of process that systemd controls and monitors.

By default, the system will have two slices: **system** and **user.**

The **system.slice** is the default location for service units. The services that ship with RHEL (ie: httpd and mariadb) will be located in this slice by default.

The **user.slice** contains all user sessions and processes.

Also, a **machine.slice** will be created when VMs and containers are started.

Each of these slices are directly under the root slice and get equal scheduler time while the system is under contention. The default configuration prevents processes from any slice from monopolizing the entire system. It also goes a step further and prevents the same thing within each slice and scope. Previously, scheduling time was done per thread, and services with a disproportional number of threads/processes received a disproportionate amount of CPU time.

By default, slices, scopes, and services default to `CPUShares=1024`. This will work perfectly for most workloads out of the box.

Note: The VM Infrastructure in this lab **may** have different values than 1024. All further commands should still work correctly.

View the default cgroup hierarchy on our workstation.

 # systemd-cgls
```
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 23  ├─user.slice
└─user-0.slice
├─session-2.scope
│ ├─2045 sshd: root@pts/1
│ └─2049 -bash
└─session-1.scope
├─2009 sshd: root@pts/0
├─2013 -bash
├─2132 systemd-cgls
└─2133 less
└─system.slice
├─polkit.service
.....................
```

Raising or lowering the `CPUShares` for a service will control the relative amount of CPU time. It works much like nice values, but a lot better.

Double the default value of `CPUShares` and verify the change:

 # systemctl set-property httpd CPUShares=2048
 # systemctl show -p CPUShares httpd
CPUShares=2048

Attention: bash-completion works on these commands. Type `systemctl [tab] [tab]` and `systemctl set-property httpd [tab] [tab]`. How great is that!? This is possible since the `bash-completion` rpm is installed (for more info: **yum info bash-completion**)
Additional systemd tools

`systemd` provides a tool, **systemd-cgtop**, to view the cgroup usage in a top-like format. This is useful for tuning systems & services. For metrics to display, at least one of the following subsystems will need "accounting" enabled. `CPUAccounting=1`, `MemoryAccounting=1`, and/or `BlockIOAccounting=1`. These can be configured with either drop-ins or with `systemctl set-property`. While tuning a system it's a good idea to use the `--runtime` option so that changes are temporary and reset upon reboot.

Performance Analysis and Tuning

There are many tools provided in RHEL 7 to help optimize and monitor performance. In this section, we are going to take a look at:

- Performance Co-Pilot (PCP),
- tuned, and
- Numad

While this lab focuses on RHEL 7, these tools are also available in more recent minor releases of RHEL 6 and can have a huge impact on system performance.

Performance Co-Pilot

Performance Co-Pilot (PCP) is a suite of tools, services and libraries for acquiring, storing and analyzing system-level performance measurements. PCP's light-weight, distributed architecture makes it particularly well suited to centralized analysis of complex systems.

PCP is broken down into two primary components:

- **PCP Collectors**: These are the parts of PCP that collect and extract performance data from various sources, e.g. the operating system kernel.

- **PCP Monitors**: These are the parts of PCP that display data collected from hosts (or archives) that have the PCP Collector installed. Many monitor tools are available as part of the core PCP release, while other (typically graphical) monitoring tools are available separately in the PCP GUI package.

Getting Started

Log into the workstation as root, install the necessary packages and start the services

```
# yum install pcp pcp-gui
# systemctl enable pmcd pmlogger
# systemctl start pmcd pmlogger
# systemctl status pmcd pmlogger
```

The default logging interval used by the pmlogger service is 60 seconds. For the purposes of this lab, we will modify the configuration to capture data at 10 seconds. Edit the configuration as described here:

 # vim /etc/pcp/pmlogger/control

duplicate or modify the following line to match

#LOCALHOSTNAME n n PCP_LOG_DIR/pmlogger/mysummary -r -T24h10m -c config.Summary

workstation y n PCP_LOG_DIR/pmlogger/workstation -r -T24h10m -c config.default -t 10s

Having made this configuration change, you will need to restart the pmlogger service

 # systemctl restart pmlogger

pmval - Memory Utilization

Next, we are going to use the **pmval** command to look at memory utilization for a specific time range: <hr>:<min>:00 to <hr>:<min>:59 iin the latest performance archive log.

pmval prints current or archived value for the nominated performance metric.

Note: Additional examples and command options are given by running man pmval on the command line

 # cd /var/log/pcp/pmlogger/workstation

 # ls -lart

Use the most recent log file (ends in .0) at the bottom of the list for the next command

 # pmval -a 20160606.00.32.0 mem.util.used

Based on the output, select a relevant start and end time range to analyze 1min of performance data.

 # pmval -a 20160606.00.32.0 -S @00:33:00 -T @00:33:55 mem.util.used

Per the pmval output, we were able to go back in time and look at memory utilization history on workstation during a recent 60 second interval

pmval - Additional Metrics

There is an extensive list of `pmval` metrics like `mem.util.used`. To list all available metrics and their descriptions, run the following command:

 # pminfo -t
 # pminfo -t | grep mem.util.used
mem.util.used [used memory metric from /proc/meminfo]

Explore the metric list and see what else interests you and test it out with `pmval`
pmstat - Overall System Performance

We can also review the overall system performance using `pmstat` command. The following command used logfile 20160606.00.32.0 to display 7 statistic entries starting from 00:33:00.

To test the same command, please change the date-stamped filename and start time accordingly

pmstat -a 20160606.00.32.0 -S @00:33:00 -s 7

@ Sat Jun 6 00:33:00 2016

Loadavg				memory	swap		io		system		cpu		
1 min	swpd	free	buff	cache	pi	po	bi	bo	in	cs	us	sy	id
0.04	0	1556m	764	145864	0	0	0	8	34	61	0	0	100
0.04	0	1556m	764	145864	0	0	0	18	35	60	0	0	100
0.03	0	1556m	764	145868	0	0	0	10	34	59	0	0	100
0.03	0	1556m	764	145868	0	0	0	0	33	57	0	0	100
0.03	0	1556m	764	145876	0	0	0	3	33	58	0	0	100
0.03	0	1556m	764	145876	0	0	0	6	33	59	0	0	100
0.02	0	1556m	764	145884	0	0	0	8	36	63	0	0	100

Additional PCP Monitors

PCP includes other monitors to view and analyze collector data:

- pmstat: Displays data similar to vmstat
- pmatop: Provides a top-like view
- pmchart: Graphical display ⇐ this is missing
- pmie: Automate actions based on performance metrics

Note: Remember that these tools can work across remote systems to analyze more complicated systems (think 3-tier applications)

Note: If you are using the graphical console of the **workstation** VM, you should be able to run pmchart from the command line.

pmchart

- Click the second icon from the left to "Open View"
- Look through the available views and select Overview
- Next, click File → New Chart and view how granular the available metrics are
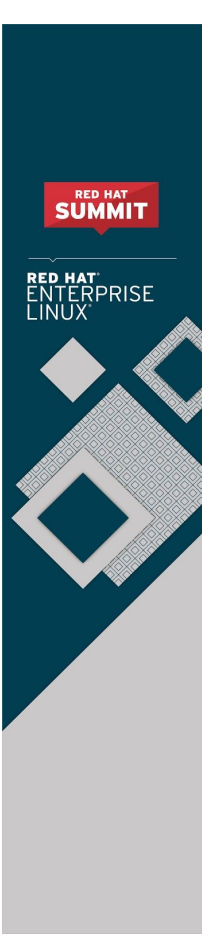- Select cgroups → cpuacct → usage   (cgroup.groups.cpuacct.usage)

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

- Explore other metrics and chart their live performance

This tool can be used to "playback" collector data to help find root cause analysis.

Tuned - Standard Tuning Profiles

tuned is a tuning daemon that can adapt the operating system for better performance. Red Hat provides tuning profiles to enhance the most commonly used workloads. In RHEL 7, **tuned** is enabled by default and also makes an intelligent decision about which profile to run out of the box. The concept of configuration inheritance has also been added in this release to make the profiles easier to customize.

For a full list of current tuning profiles:

 # man tuned-profiles

Getting Started

Log into the workstation as root and identify the running profile.

 # tuned-adm active
Current active profile: virtual-guest

`tuned-adm` can assess your system and make a tuning profile recommendation. This also sets the default profile for your system at install time

 # tuned-adm recommend
virtual-guest

Next, list the available profiles on your system

 # tuned-adm list

Available profiles:
- balanced
- desktop
...
- virtual-host

Current active profile: virtual-guest

The details of the profiles can be found in the man page

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

 # man tuned-profiles

Change the Current Tuning Profile

To switch to another existing `tuned` profile (ex: `powersave`), use the **tuned-adm** command.

 # tuned-adm profile powersave

Now use **tuned-adm** again to verify that your system tuning profile is now set to **powersave.**

 # tuned-adm active
Current active profile: powersave

Customizing a Tuning Profile

Let us say our system is running an application that works well with the **virtual-guest** profile but not with Transparent Hugepages (THP). Examples of workloads where THP are not optimal include: SAP HANA, DB2, Datastage, Ambari, etc.

Note: Red Hat includes specific bare metal and virt profiles for SAP HANA with RHEL.

Begin by checking the current status of THP

 # cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never

Now let us create a directory for our custom configuration and then create a config which inherits virtual-guest and then modifies THP.

 # cd /usr/lib/tuned

 # mkdir virtual-guest-no-thp
 # cd virtual-guest-no-thp

 # vim tuned.conf

add the following contents

```
[main]
include=virtual-guest

[vm]
transparent_hugepages=never
```

Now save the file and load the new tuning profile. Finally, check your work.

 # tuned-adm profile virtual-guest-no-thp
 # tuned-adm active
Current active profile: virtual-guest-no-thp

 # cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]

By looking at other `tuned.conf` files in `/usr/lib/tuned`, you will notice that other profile's `tuned.conf` contains a `[sysctl]` section. It is common practice to place `sysctl` tunings in /etc/sysctl.conf so they are set on boot, however `tuned` provides a mechanism for maintaining these types of tunables as well as others like disk scheduling and power settings for workload profiles.

Disabling Tuned

`tuned` is simple to disable if you choose not to run it.

 # tuned-adm off
 # tuned-adm active
No current active profile.

 # systemctl stop tuned.service
 # systemctl disable tuned.service

Optimizing NUMA

Historically, NUMA has been one of the most important items to tune and account for on larger systems. The RHEL 7 kernel implements automatic NUMA balancing for hardware with NUMA properties. Both following conditions are required:

- **numactl**: hardware shows multiple nodes
- **NUMA flags**: NUMA options can be set in
  `/sys/kernel/debug/sched_features`

This is the first release of RHEL where, out of the box, NUMA will require little to no tuning considerations for most workloads. The kernel is NUMA aware and in most cases will simply "do the right thing". That said, there are still edge cases where certain workloads will perform better with manual pinning (`numctl`) or from running `numad`.

Getting Started

Red Hat introduced **numad** (an automatic NUMA affinity management daemon) in RHEL 6.3. It is a userspace tool that aims to improve out-of-the-box NUMA system performance for any long running, significant resource consuming processes (ex: KVM processes, HPC applications, etc…). It is not likely to help with processes that run only a few minutes, don't do much processing or don't use much memory.

By default, `numad` is not installed on a RHEL 7 host. The following steps will walk you through installing and enabling `numad` on Red Hat Enterprise Linux 7.

```
# yum install numad numactl


# systemctl enable numad.service
# systemctl start numad.service
# systemctl status numad.service
```

numactl lets administrators run a process with a specified scheduling or memory placement policy. It can also set a persistent policy for shared memory segments or files, and set the processor affinity and memory affinity of a process. Granted this is not too exciting on our small lab VM, but let's look at the current resources on your VM using numactl:

```
# numactl --hardware
available: 1 nodes (0)
node 0 cpus: 0 1
node 0 size: 4095 MB
node 0 free: 2294 MB
node distances:
node  0
 0:  10
```

Now run lscpu to gather CPU architecture information from sysfs and /proc/cpuinfo

```
# lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           2
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               15
Model name:          Intel(R) Xeon(R) CPU @ 2.50GHz
Stepping:            11
CPU MHz:             2499.998
BogoMIPS:            4999.99
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            4096K
```

```
NUMA node0 CPU(s):     0,1
```

Let's interpret the output from the previous commands. Based on the `lscpu` output, it shows that this VM has 1 NUMA node, 2 CPU sockets, and 2 CPU cores. `numactl` is also reporting that our single NUMA node host has total of 4095 MB of memory with 2294 MB free currently.

Note: Your output may differ due the the lab environment

Sample numactl Output From A Larger Host

In a multi-CPU server environment, `numactl` is able to display additional information about the CPU placements on the motherboard. Here is the `numactl` output of a multi-CPU server:

Note: this output comes from a different physical host and provided as an example.

```
# numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36 node 0 size: 65415 MB
node 0 free: 43971 MB

node 1 cpus: 2 6 10 14 18 22 26 30 34 38 node 1 size: 65536 MB
node 1 free: 44321 MB

node 2 cpus: 1 5 9 13 17 21 25 29 33 37 node 2 size: 65536 MB
node 2 free: 44304 MB

node 3 cpus: 3 7 11 15 19 23 27 31 35 39 node 3 size: 65536 MB
node 3 free: 44329   MB

node   distances:
node  0     1      2      3
```

```
0:   10   21   21   21
1:   21   10   21   21
2:   21   21   10   21
3:   21   21   21   10
```

Sample lscpu Output From A Larger Host

Here is `lscpu` output of a multi-CPU server

Note: this output comes from a different physical host and provided as an example.
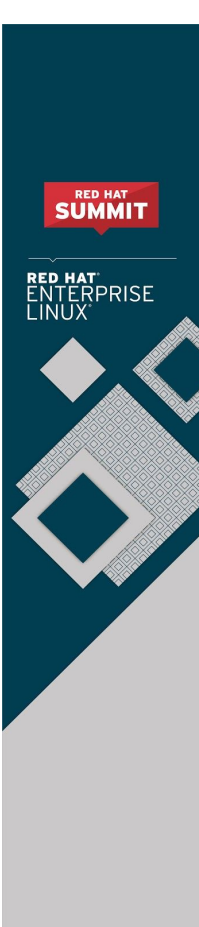
```
# lscpu
Architecture:   x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order:     Little Endian
CPU(s):     40
On-line CPU(s) list: 0-39
Thread(s) per core:   1
Core(s) per socket:   10
Socket(s): 4
NUMA node(s):    4
Vendor ID: GenuineIntel
CPU family:      6
Model:      47
Model name:      Intel(R) Xeon(R) CPU E7- 4870  @ 2.40GHz
Stepping:  2
CPU MHz:   2394.204
BogoMIPS:  4787.85
Virtualization: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 30720K
NUMA node0 CPU(s):   0,4,8,12,16,20,24,28,32,36
NUMA node1 CPU(s):   2,6,10,14,18,22,26,30,34,38
NUMA node2 CPU(s):   1,5,9,13,17,21,25,29,33,37
NUMA node3 CPU(s):   3,7,11,15,19,23,27,31,35,39
```

Based on previous outputs of our larger host, `numactl` is able to display current free and total memory that is local to each NUMA node. Also the relative distance between 2 CPU sockets on the motherboard. Based on node distance information from 'numactl --hardware', we know that any given CPU has direct connection to another CPU. CPU 0's distance to CPU 0 is 10 (the shortest), to CPU 1 is 21, to CPU 2 is 21 and to CPU 3 is 21. (i.e. same distance from CPU 0 to CPU 1, 2, and 3)

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

NUMA Statistics

The **numastat** tool displays per-NUMA node memory statistics for processes and the operating system. It shows administrators whether process memory is spread throughout a system or centralized on specific nodes.

```
# numastat -v
Per-node numastat info (in Mbs):
                 Node 0          Total
                 --------------- ---------------
Numa_Hit         11718.43        11718.43
Numa_Miss        0.00            0.00
Numa_Foreign     0.00            0.00
Interleave_Hit   46.96           46.96
Local_Node       11718.43        11718.43
Other_Node       0.00            0.00
```

To find a description of each value displayed above or other numastat options, review the man page for numastat.

```
# man numastat
```

Most importantly to look out for are: **numa_miss**, **numa_foreign** and **other_node** values. A high value indicates a process has attempted to get a page from its local NUMA node, but it was out of free pages and the system had to allocate free pages from another NUMA node.

Below is an example of a RHEL 6 hypervisor running without `numad`. Notice the VMs are split almost evenly across the sockets.

```
File  Edit  View  Search  Terminal  Help          Vms are split between nodes
[root@rhel-h2.rdu ~]$ numastat -n qemu

Per-node process memory usage (in MBs)
PID                        Node 0            Node 1            Total
-----------------     ----------------  ----------------  ----------------
3568  (qemu-kvm)              583.24            577.30           1160.54
3807  (qemu-kvm)             5486.69           6716.62          12203.31
4488  (qemu-kvm)              659.03           1643.70           2302.73
7672  (qemu-kvm)              711.23           4984.16           5695.38
14250 (qemu-kvm)             2159.01           2019.75           4178.77
14823 (qemu-kvm)              484.36            695.58           1179.95
-----------------     ----------------  ----------------  ----------------
Total                      10083.56          16637.11          26720.67

Per-node numastat info (in MBs):
                           Node 0            Node 1            Total
                     ----------------  ----------------  ----------------
Numa_Hit                21081785.76       18074538.80       39156324.56
Numa_Miss                   2268.10            134.90           2403.00
Numa_Foreign                 134.90           2268.10           2403.00
Interleave_Hit                54.49             54.68            109.17
Local_Node              21081574.10       18074490.33       39156064.43
Other_Node                  2479.76            183.36           2663.12
[root@rhel-h2.rdu ~]$ 
```

Next is the same hardware running `numad`. Notice the NUMA alignment is almost perfect and the Numa_Miss count dropped from ~2300 to ~7.

Enable and Disable NUMA Balancing

Disable To disable/enable system-wide automatic NUMA balancing, use the following commands

To disable NUMA balancing:
```
# echo 0 > /proc/sys/kernel/numa_balancing
```

To enable NUMA balancing:
```
# echo 1 > /proc/sys/kernel/numa_balancing
```

LOGICAL VOLUME MANAGER (LVM)

With the introduction of RHEL 7, two new storage management interfaces are available:

- System Storage Manager (SSM), and
- libStorageMgmt

System Storage Manager (**ssm**) is a new command line interface for consolidated storage operations which integrates logical volume management, device mapper (DM), multiple-device (MD) and file-system operations into one unified user interface.

The libStorageMgmt package is a storage array independent Application Programming Interface (API). It provides a stable and consistent API that allows developers the ability to programmatically manage different storage arrays and leverage the hardware accelerated features provided. This library is used as a building block for other higher level management tools and applications. End system administrators can also use it as a tool to manually manage storage and automate storage management tasks with the use of scripts.

With the release of RHEL 7.5, Red Hat added new technology into storage stack allowing for compression and deduplication. The technology came by way of the acquisition of a company called Permabit and it is called the Virtual Data Optimizer... or VDO.

System Storage Manager (SSM)

Traditional manual steps to construct a working file-system from the ground up went like this:

- Partition a block device (`fdisk` or `parted`)
- Create a physical volume (`pvcreate`)
- Create a logical volume group (`vgcreate`)
- Create a logical volume (`lvcreate`)
- Create a filesystem (`mkfs`)
- Mount the filesystem (`mount`)
- Create entry in `/etc/fstab` for persistence

With SSM, the workflow is as follows:

- Partition a block device (`fdisk` or `parted`)
- Create pv, vg, lv, filesystem and mount (`ssm`)
- Add entry in `/etc/fstab` for persistence

Getting Started

First, we need to install system-storage-manager utility.

```
# yum install system-storage-manager
```

Notice that there are NO physical storage devices. We are using files mapped as block devices via device-mapper.
Create a Simple Filesystem

 # ssm list devices

```
-------------------------------------------------------------------
Device          Free    Used     Total  Pool          Mount point
-------------------------------------------------------------------
/dev/fd0                          4.00 KB
/dev/loop0                      100.00 MB
/dev/loop1                      100.00 MB
/dev/loop2                      100.00 MB
/dev/loop3                      100.00 MB
/dev/loop4                      100.00 MB
/dev/loop5                      100.00 MB
/dev/loop6                      100.00 MB
/dev/vda                         10.00 GB                PARTITIONED
/dev/vda1                       500.00 MB                /boot
/dev/vda2   0.00 KB  9.51 GB    9.51 GB  rhel_pwob-r7
-------------------------------------------------------------------
```

** IMPORTANT **

If you don't see loop0-loop6, then stop and rerun the ansible playbook as described at the top of this lab.

 # ssm list volumes

```
---------------------------------------------------------------------------------------
Volume                 Pool          Volume size  FS      FS size     Free  Type   Mount point
---------------------------------------------------------------------------------------
/dev/rhel_pwob-r7/swap  rhel_pwob-r7    1.00 GB                             linear
/dev/rhel_pwob-r7/root  rhel_pwob-r7    8.51 GB  xfs    8.50 GB   2.72 GB  linear /
/dev/vda1                             500.00 MB  xfs  496.67 MB  284.79 MB  part   /boot
---------------------------------------------------------------------------------------
```

Traditional filesystem

Creation

```
# ssm -f create --fstype ext4 /dev/mapper/loop0p1 /mnt/exercise1


  Physical volume "/dev/dm-2" successfully created.
  Volume group "lvm_pool" successfully created
  Logical volume "lvol001" created.
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: done
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=0 blocks, Stripe width=0 blocks
24576 inodes, 98304 blocks
4915 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=33685504
12 block groups
8192 blocks per group, 8192 fragments per group
2048 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729

Allocating group tables: done
Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

Directory '/mnt/exercise' does not exist! Create (Y/n/q) ? Y
You have new mail in /var/spool/mail/root
```

```
# df /mnt/exercise1


Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/mapper/lvm_pool-lvol001 91099   1550    82668   2% /mnt/exercise1
```

Removal and Cleanup

```
# ssm -f remove /mnt/exercise1 lvm_pool

Device '/dev/lvm_pool/lvol001' is mounted on '/mnt/exercise1' Unmount (N/y/q) ? Y
  Logical volume "lvol001" successfully removed
  Volume group "lvm_pool" successfully removed


# pvremove /dev/mapper/loop0p1

Labels on physical volume "/dev/mapper/loop0p1" successfully wiped
```

RAID filesystems

Creating RAID-1 (Mirrored) Filesystem

```
# ssm -f create \
        --fstype ext4 \
        --size 50M \
        -p summitvg \
        -n exercise2 \
        -r 1 \
        /dev/mapper/loop{0,1}p1 /mnt/exercise2
```

```
Physical volume "/dev/mapper/loop0p1" successfully created
Physical volume "/dev/mapper/loop1p1" successfully created
Volume group "summitvg" successfully created
Rounding up size to full physical extent 52.00 MiB
Logical volume "exercise2" created.
meta-data=/dev/summitvg/exercise2 isize=256 agcount=2, agsize=6656 blks
        =                sectsz=512    attr=2, projid32bit=1
        =                crc=0         finobt=0
data    =                bsize=4096    blocks=13312, imaxpct=25
        =                sunit=0 swidth=0 blks
naming  =version 2       bsize=4096    ascii-ci=0 ftype=0
log     =internal log bsize=4096    blocks=853, version=2
        =                sectsz=512    sunit=0 blks, lazy-count=1
realtime =none           extsz=4096    blocks=0, rtextents=0
```

```
# df -T /mnt/exercise2

Filesystem              Type 1K-blocks Used Available Use% Mounted on
/dev/mapper/summitvg-exercise2 ext4 47463   1038    42699   3% /mnt/exercise2

# lvs summitvg

LV      VG      Attr    LSize  Pool Origin Data%   Meta%  Move Log Cpy%Sync Convert
exercise2 summitvg rwi-aor--- 52.00m                      100.00
```

Creating RAID-10 (Mirrored and Striped) Filesystem

```
# ssm list volumes

--------------------------------------------------------------------------------

Volume Pool    Volume size  FS        FS size Free  Type     Mount point
--------------------------------------------------------------------------------

/dev/rhel_workstation/root   rhel_workstation    8.47   GB     xfs    8.46   GB
4.87   GB     linear  /
/dev/rhel_workstation/swap   rhel_workstation    1.00   GB
             linear
/dev/summitvg/exercise2      summitvg    52.00  MB     ext4   52.00 MB     42.74 MB
     raid1   /mnt/exercise2
/dev/sda1            500.00 MB     xfs    496.67 MB     397.81 MB     part   /boot
--------------------------------------------------------------------------------

# ssm -f create \
    --size 100M \
    --fstype xfs \
    --pool summitvg \
    --name exercise3 \
    --raid 10 \
    /dev/mapper/loop{2..5}p1 /mnt/exercise3
```

```
Physical       volume "/dev/mapper/loop2p1" successfully created
Physical       volume "/dev/mapper/loop3p1" successfully created
Physical       volume "/dev/mapper/loop4p1" successfully created
Physical       volume "/dev/mapper/loop5p1" successfully created
Volume group "summitvg" successfully extended
Rounding       size (25 extents) up to stripe boundary size (26 extents).
Logical volume "exercise3" created.
meta-data=/dev/summitvg/exercise3 isize=256 agcount=4, agsize=6640 blks
        =                     sectsz=512    attr=2, projid32bit=1
        =                     crc=0         finobt=0
data    =                     bsize=4096    blocks=26560, imaxpct=25
        =                     sunit=16      swidth=64 blks
naming =version 2            bsize=4096    ascii-ci=0 ftype=0
log    =internal log bsize=4096    blocks=768, version=2
        =                     sectsz=512    sunit=16 blks, lazy-count=1
realtime =none         extsz=4096    blocks=0, rtextents=0
```

df /mnt/exercise3

```
Filesystem               Type 1K-blocks Used Available Use% Mounted on
/dev/mapper/summitvg-exercise3 xfs   103168 5472   97696   6% /mnt/exercise3
```

lvs summitvg

```
LV          VG      Attr    LSize  Pool Origin Data%  Meta%      Move Log Cpy%Sync
Convert
exercise2 summitvg rwi-aor---52.00m           100.00
exercise3 summitvg rwi-aor--- 104.00m              100.00
```

lvs -o +segtype summitvg

```
LV      VG      Attr    LSize   Pool Origin Data%    Meta%  Move Log Cpy%Sync Convert Type
exercise2 summitvg rwi-aor---52.00m 100.00 raid1
exercise3 summitvg rwi-aor--- 104.00m       100.00 raid10
```

Virtual Data Optimizer (VDO)

First, we need to ensure the vdo and kmod-kvdo packages are installed.

```
# yum install kmod-kvdo vdo
```

Notice the the packages were already installed and you are ready to create your optimized filesystems.

Again, recall that there are NO physical storage devices involved in this exercise. We are using files mapped as block devices via device-mapper. You would NOT do this in a real world scenario.

The workflow will resemble the traditional manual approach with one extra step
- Partition a block device (`fdisk` or `parted`)
- Create a physical volume (`pvcreate`)
- Create a logical volume group (`vgcreate`)
- Create a logical volume (`lvcreate`)
- **Configure VDO**
- Create a filesystem (`mkfs`)
- Mount the filesystem (`mount`)
- Create entry in `/etc/fstab` for persistence

Create an Optimized Filesystem

The disk has already be partitioned, so we will being by creating the physical volume, the logical volume group and finally the logical volume.

```
# pvcreate /dev/mapper/loop6p1
# vgcreate vdovg /dev/mapper/loop6p1
# lvcreate -n exercise4 -l 100%VG vdovg
```

NETWORKMANAGER CLI

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

RHEL 7 brings new functionality to the command line for NetworkManager, the default network configuration utility. While you can still edit these interfaces by hand, the command line interface, `nmcli`, is able to provide a consistent, scriptable, frame-work for you to work with.

In this part of the lab, we will discuss the general operation and command structure, as well as how to configure a bonded interface, along with the differences between bonding and teaming of NIC devices.

Getting Started

1. Let's get started by logging into the `workstation` VM as `root` (if not already logged in) and simply perform the following command to get a quick status of what you have access to:

```
# nmcli device status
DEVICE      TYPE       STATE CONNECTION
eth0        ethernet   connected  eth0
eth1        ethernet   disconnected   -
eth2        ethernet   disconnected   -
eth3        ethernet   disconnected   --
```

At this point, you should see output similar to the above information. On this system, we have one device that is connected, eth0, and eth[1:3] are disconnected. We will primarily be working with the currently disconnected devices.

(WARNING) it is likely that the device names are different (ie: ensX vs. ethX)

Notice that the command structure is:

```
nmcli [OPTIONS] OBJECT { COMMAND | help }
```

It's also important to mention that the bash-completion package will support `nmcli` commands, so while some of the commands below will look a bit long, they can be typed more quickly than it may appear.

2. From here, we're going to use nmcli to create a new connection on eth1 with a static IP configuration. We'll be using a very generic ipv4 network, however ipv6 is supported, and uses similar command structures (ip6/gw6, etc). Not only will this create a static connection for eth1, but it will create a NetworkManager profile, called lab-eth1. To do this, issue the following command:

```
# nmcli con add con-name lab-eth1 ifname eth1 type ethernet ip4
192.168.122.100/24 gw4 192.168.122.1
```

Connection `lab-eth1` (xxxxxx) successfully added.

3. Now that we've done this, let's take a look at `/etc/sysconfig/network-scripts/ifcfg-lab-eth1` to see what configuration variables have been populated.

```
# cat /etc/sysconfig/network-scripts/ifcfg-lab-eth1
```

As you can see, more than the standard set of options have been applied and the interface is UP.

```
# nmcli device status; ip a
```

4. Let's now DOWN the interface
```
# nmcli con down "lab-eth1"
# nmcli device status; ip a
```

5. You can begin using this interface by bringing up the network with this command:

```
# nmcli con up "lab-eth1" ifname eth1
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/Activate Connection/2)
```

In the above command, notice that we still explicitly call out the interface name that we're applying that connection profile to. Because we can see from the `ifcfg-lab-eth1` script that it's already assigned to that interface, we don't necessarily need that. (e.g. `nmcli con up lab-eth1`) We could, however, apply a wild card to that interface name within the profile, which would allow us to apply that connection string to any interface we wish.

6. Let's examine the `nmcli` device status again, this time looking at the newly brought up `lab-eth1` connection, assigned to the `eth1` device:

```
# nmcli device status
DEVICE      TYPE         STATE CONNECTION
eth0        ethernet     connected  eth0
eth1        ethernet     connected  lab-eth1
eth2        ethernet     disconnected    -
eth3        ethernet     disconnected    --
```

7. Next, we're going to create a bonded (active/passive; Mode 1) interface, with two slaves. As we can see from above, `eth2` and `eth3` are available for the bonding. Get started by using `nmcli` to create a `bond0` profile for us to work with:

```
# nmcli con add type bond ifname bond0 mode active-backup
Connection 'bond-bond0' (xxxxxxxx) successfully added.
```

Note: If you can't remember the bonding mode try: `nmcli con add type bond ifname bond0 mode [tab] [tab]` to see the available options.

8. Examining the `/etc/sysconfig/network-scripts/ifcfg-bond-bond0` profile, we can see that it's a very basic configuration, and cannot work until we add some slave interfaces, so next let's use `nmcli` to add a pair of slaves:

```
# cat /etc/sysconfig/network-scripts/ifcfg-bond-bond0
```


```
# nmcli con add type bond-slave ifname eth2 master bond0
Connection 'bond-slave-eth2' (xxxxxxxx) successfully added.
```

```
# nmcli con add type bond-slave ifname eth3 master bond0
Connection 'bond-slave-eth3' (xxxxxxxx) successfully added.
```


9. In order to bring up the bond using nmcli, bring up the slaves first, then the bond:

```
# nmcli con up bond-slave-eth2
Connection successfully activated...
```

```
# nmcli con up bond-slave-eth3
Connection successfully activated...
```

```
# nmcli con up bond-bond0
Connection successfully activated...
```

10. Now that our bond is configured let's set another static IPv4 address using the modify command:

```
# nmcli con mod bond-bond0 ipv4.addresses 192.168.122.101/24
# nmcli con mod bond-bond0 ipv4.method manual
# nmcli con up bond-bond0
# ip ad sh bond0
```

Network Teaming

The next new concept to introduce is Network Teaming. The two main benefits of NIC Teaming are data aggregation and failover, above and beyond the capabilities of network bonding. Another upside is that you can actually convert a bond to a team, which is our next exercise with `nmcli`. In order to simplify the environment, we will not be changing the name of the bond, as the conversion tool will not update things such as `firewalld`, or any other scripts or programs outside of the `ifcfg` files.

1. Network teaming uses `libteam` to control one instance of the team driver. Ensure that `teamd` is already installed on your local workstation. We can also see that `teamd.service` is static, and doesn't need to be started or enabled.

```
# yum install teamd
# systemctl list-unit-files | grep teamd
teamd@.service   static
```

2. Next, we need to clean up a few pieces of the former section – run the following to bring down the bond and the slave interfaces, and then remove the configurations, all within `nmcli`:

```
# nmcli con show
# nmcli con down bond-slave-eth2
Connection 'bond-slave-eth2' successfully deactivated.

# nmcli con down bond-slave-eth3
Connection 'bond-slave-eth3' successfully deactivated.

# nmcli con down bond-bond0
Connection 'bond-bond0' successfully deactivated.

# nmcli con delete bond-slave-eth2
# nmcli con delete bond-slave-eth3
# nmcli con delete bond-bond0
# nmcli con show
```

3. Now, to create the new teaming interface(s), perform the following:

```
# nmcli con add type team ifname team0 config \
      '{"runner":{"name": "activebackup"}}'

Connection 'team-team0' successfully added.
```

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

```
# nmcli con show team-team0
<output properties for team-team0>

# nmcli con add type team-slave con-name team0-port1 ifname eth2 master
team-team0
Connection 'team0-port1' successfully added.

# nmcli con add type team-slave con-name team0-port2 ifname eth3 master
team-team0
Connection 'team0-port2' successfully added.
```

4. Notice that the information above seems very similar in nature to the bonding interfaces that we created earlier in the lab. Now, use `nmcli` to bring up the interfaces, and examine the properties of the connection:

```
# nmcli con up team0-port1
# nmcli con up team0-port2
# nmcli con up team-team0
# nmcli con sh team-team0
# nmcli device status; ip a
```

Examine the network properties in the output of the last two commands. We see what IP is assigned, which port actually has the address, and how traffic would be flowing (should any be heading across the interface). It also shows the mode (active/backup).

5. If we take down one of the interfaces (the one holding traffic), we can then do another 'show details', and we see that the IP address has moved over to the other interface.

```
# nmcli con down team0-port1
# nmcli con sh team-team0
# nmcli device status; ip a
```

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

DYNAMIC FIREWALL WITH FIREWALLD

The dynamic firewall daemon, `firewalld`, provides a dynamically managed firewall with support for network "zones" to assign a level of trust to a network and its associated connections and interfaces. It has support for IPv4 and IPv6 firewall settings. It supports Ethernet bridges and has a separation of runtime and permanent configuration options. It also has an interface for services or applications to add firewall rules directly.

The `iptables` service stores configuration in `/etc/sysconfig/iptables` while `firewalld` stores it in various XML files in `/usr/lib/firewalld/` and `/etc/firewalld/`.
Note: The `/etc/sysconfig/iptables` file does not exist as `firewalld` is installed by default on Red Hat Enterprise Linux.

With the `iptables` service, every single change means flushing all the old rules and reading all the new rules from `/etc/sysconfig/iptables` while with `firewalld` there is no re-creating of all the rules; only the differences are applied. Consequently, `firewalld` can change the settings during runtime without existing connections being lost.

Both `firewalld` and the legacy `iptables` command use `iptables` tool to talk to the kernel packet filter. Using the `iptables` service interferes with running the `firewalld` service - do not use them both.
Note: Making use of the `systemctl mask` command will ensure that only one or the other service may be started, even if both are available on the system.

Get started

1. Log into `workstation` VM as `root`.

2. `firewalld` is already installed and running on RHEL7. These steps will restore `firewalld` if it has been disabled. The below steps are not necessary to run in this lab

```
#      systemctl stop iptables ip6tables
#      systemctl disable iptables ip6tables
#      yum install firewalld firewall-config
#      systemctl enable firewalld
#      systemctl start firewalld
#      systemctl status -l firewalld
```

3. `firewalld` uses several different, completely adaptable, XML config files to store configurations. These files also ensure run-time and persistent configuration separation. The default configuration files are in **/usr/lib/firewalld**, while user customized files (which take precedence if they exist) are in **/etc/firewalld**.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Examine the default files - both the zones and services files

```
# cd /usr/lib/firewalld/zones
# cd /usr/lib/firewalld/services
```

4. `firewald` uses **zones** to assign a level of trust to a network and its associated connections and interfaces. The zone files include definitions of Services, Ports (ranges) with protocols, Rich rules, Internet Control Message Protocol (ICMP) blocks , Masquerading and Port/packet forwarding. There are several built-in zones: `block`, `dmz`, `drop`, `external`, `home`, `internal`, `public`, `trusted`, `work` - and you can also create new zones

Each zone is similar to a complete firewall and there is a single zone selection per environment or connection. The **public** zone is the Initial default. Zones are identified in ifcfg files or NetworkManager configuration using: `ZONE=<name>`

Here is an example of the public zone file from `/usr/lib/firewalld/zones`

```
public
<?xml version="1.0" encoding="utf-8"?>
<zone>
     <service name="ssh"/>
     <service name="dhcpv6-client"/>
</zone>


drop

<?xml version="1.0" encoding="utf-8"?> <zone target="DROP">
</zone>


custom
<?xml version="1.0" encoding="utf-8"?>
<zone>
     <interface name="em2"/>
     <source address="10.0.1.0/24"/>
     <service name="ssh"/>
     <service name="ipp-client"/>
     <service name="dhcpv6-client"/>
     <rule><protocol value="ah"/><accept/></rule>
</zone>
```

5. There are nearly 50 built-in services files shipped with `firewalld`. Services files include Port (ranges) with protocol, Netfilter helper modules, and destination address (range) for IPv4 and/or IPv6. You can also create your own services files or customize the default files. To customize a file, simply copy it from `/usr/lib/firewalld/services` to `/etc/firewalld/services`

then make any required changes.

Here is an example of the default `http.xml` services file from
`/usr/lib/firewalld/services`

```xml
<?xml version="1.0" encoding="utf-8"?>
<service>
     <short>WWW (HTTP)</short>
     <description>HTTP is the protocol used to serve Web pages. If you
plan to make your Web server publicly available, enable this option.
This option is not required for viewing pages locally or developing Web
pages.</description>
     <port protocol="tcp" port="80"/>
</service>
```

Using the `firewalld` GUI

1. Start `firewall-config` - either by running `firewall-config` from command line or by hitting the super key and typing `firewall-config` (or until the icon shows) or using the menu bar: `Applications → Sundry → Firewall`

2. To identify the current status of the firewall from the GUI
    ● Notice Connected in lower left
    ● Also notice Runtime vs Permanent – upper left Configuration:
    ● Runtime – changes applied immediately
    ● Permanent – changes applied at next system start or firewalld reload
    ● Notice that there is a Runtime to Permanent command under Options

3. In the Lower Right, there are several indicators – Default Zone, Lockdown, Panic Mode

4. Under the Views menus - select all the available options - ICMP Types, Direct Configuration and Lockdown Whitelist

5. Look at items under Options – hover over each to get a description

6. Lets look at the settings for the Default Zone
    ● Select Runtime
    ● Select Public Zone
    ● Under Services tab - look at enabled services
    ● Check http and https and ssh
    ● Under the upper Services Tab - examine the services you have selected – note the ports

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

- Under Zones - Ports tab - examine ports

Ports 80, 443, 22 should be enabled, based on the Services you selected

7. Add a new port and enable Port Forwarding ◦ Still examining the Public Zone
- In Zone Ports - Click Add and add TCP 8080
- Examine Masquerading and Port Forwarding – there should be nothing there
- If the local web server is listening on 8080 but we want people to access it via port 80 – we can forward port 80 to 8080
- select the Port Forwarding Tab - click add and select Protocol TCP, Port 80, Local Forwarding, 8080

8. Look at the remaining Public Zone tabs
- Examine ICMP Filter
- Examine Rich Rules
- Click the right arrow (>) twice to Interfaces and Source show
- Examine Interfaces and Sources
- Select the interface and click `edit` - it allows you to change it's zone

9. Any changes you make to the Runtime take effect immediately. However, they will not persist if I reboot or restart the firewall. To make them persistent, you can change from Runtime to Permanent and make all the same changes - or the easier way is to use the Runtime to Permanent command under options.

`Options->Runtime to Permanent` – saves the Runtime set up to the Permanent files so it will be this way when we start up – don't do this so that we can revert back to the initial state after the lab

10. First let's examine the predefined services
- Click the `Services` tab and examine the services

11. Now we'll create a custom service
Note: that there are no custom services

```
# ls /etc/firewalld/services/
# nothing there
```

Services can not be modified while in the run time mode - so we must change to Permanent from Runtime
- Select a service you are not using, such as Samba
- Under `Ports and Prococols`, click `Add`
- Enter a new Port, `446 tcp`

Now we have a custom samba service with a file in `/etc/firewalld/services`

```
# ls /etc/firewalld/services/
```

Let's look at it

```
# cat /etc/firewalld/services/samba.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
     <short>Samba</short>
     <description>This option allows you to access and participate in
Windows file and printer sharing networks. You need the samba package
installed for this option to be useful.</description>
     <port protocol="tcp" port="446"/>
     <port protocol="udp" port="137"/>
     <port protocol="udp" port="138"/>
     <port protocol="tcp" port="139"/>
     <port protocol="tcp" port="445"/>
     <module name="nf_conntrack_netbios_ns"/>
</service>
```

12. To revert back to using the default Samba service file - we'll simply remove the file in `/etc/firewalld/services` ◦ change back to Runtime

```
# rm /etc/firewalld/services/samba.xml
```

Change back to Permanent and view the Samba Ports.

Verify defaults are back.

Using the `firewalld` CLI

1. The `firewalld` command line is `firewall-cmd`

```
# firewall-cmd
```

2. First let's check the state and status
Note: All commands are applied to Runtime by default. Add `--permanent` to commands to make them apply to permanent environment. if **zone** isn't specified, then the default zone, in this case `public`, is assumed. To make commands apply to another zone add `--zone=<zone>` to the command.

```
# firewall-cmd --state
```

3. To reload the `firewalld` files use `--reload`.

```
# firewall-cmd --reload
```

4. Use the following commands to get information about the firewall:

```
# firewall-cmd  --get-active-zones
```

```
# firewall-cmd --zone=public --list-interfaces
```

```
# firewall-cmd --zone=public --list-services
```

```
# firewall-cmd --zone=public --list-all
```

5. Let's add some services and ports to the `public` zone (since `public` is the default - we don't have to explicitly specify it.) First we'll add the `http` and `https` services

```
# firewall-cmd --add-service=http --add-service=https
```

```
# firewall-cmd --list-services
```

Next we'll add port 8080:

```
# firewall-cmd --add-port=8080/tcp
```

Finally, we'll enable port-forwarding:

```
# firewall-cmd --add-forward-port=port=80:proto=tcp:toport=8080
```

6. Panic mode allows you turn off all network traffic. The panic mode commands are:

```
# firewall-cmd --query-panic
# firewall-cmd --panic-on
# firewall-cmd --panic-off
```

To see panic mode in action open two terminals. In terminal one, `ping localhost`:

```
# ping localhost
```

In terminal two, first query panic mode status:

```
# firewall-cmd --query-panic
```

Next, turn on panic mode from terminal two.

```
# firewall-cmd --panic-on
```

Note the impact on the `ping` command in terminal one.

Turn panic mode back off.

firewall-cmd --panic-off

7. A useful option is the timeout option. Add `--timeout=<interval><s/m/h>` to make a command revert back after so many seconds. This is a great way to test commands without causing lasting damage if you make a mistake. To see the `timeout` option at work, either use the same to terminals from the last exercise or open two new ones.

In terminal one run the following command to see the services enabled for the `public` zone:

watch firewall-cmd --zone=public --list-services

In terminal two use the following command to add the `mysql` service to the `public` zone - but make it only last for 10 seconds:

firewall-cmd --add-service=mysql --timeout=10s

Verify that `mysql` shows as a service in first window - and that it disappears in 10 seconds.

Clean Up

1. Revert back to default `firewalld` configuration either using the GUI (`Options->Reload Firewall`) or the CLI
firewall-cmd --reload

2. Close any unneeded terminals

System Administration with Cockpit

Cockpit is a system administration tool that provides a user interface for monitoring and administering servers through a web browser. It allows you to monitor current values and adjust limits on system resources, control life cycle on container instances, and manipulate container images.

Here are a few important facts about Cockpit:

- Cockpit does not add a layer of other functionalities that are not present on your systems. It exposes user interface elements that enable you to interact with the system.

- Cockpit does not take control over your servers, in a way that when you configure something from Cockpit, you can only manage it from there. You can effectively move away from Cockpit to the command-line and come back to it at any point.

- Cockpit does not require configuration or infrastructure, and once you install it, it is ready for use. You could, however, configure it to make use of the authentication infrastructure that is available to you, for example a single sign-on system like Kerberos.

- Cockpit has zero memory and process footprint on the server when not in use.

- Cockpit does not store data or policy. This also means it does not have its own users. The users from the systems can authenticate in Cockpit using their system credentials and they keep the same permissions.

- Cockpit dynamically updates itself to reflect the current state of the server, within a time frame of a few seconds.

- Cockpit is not intended for configuration management. This means that Cockpit itself does not have a predefined template or state for the server that it then imposes on the server. Cockpit can interact with other configuration management systems or custom tools that are manipulating server configuration.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Getting Started

To install Cockpit on workstation

1. Log into workstation VM as root

2. Verifying that yum repos are enabled to install Cockpit

```
# yum repolist
```

Make sure rhel-7-server-rpms, rhel-7-server-optional-rpms and rhel-7-server-extras-rpms repos are enabled.

3. Installing *Cockpit* on `workstation`

```
# yum install -y cockpit cockpit-dashboard
```

4. Enabling and starting Cockpit service

```
# systemctl enable cockpit
# systemctl start cockpit
# systemctl status cockpit
```

You should see cockpit.service is now active and running

5. *Optional:* if firewall is running on your server

```
# firewall-cmd --add-port=9090/tcp
# firewall-cmd --permanent --add-port=9090/tcp
```

6. Log into *Cockpit* user interface

On your workstation, click on Applications (top left corner) -> Firefox Web Browser

Enter URL: "https://localhost:9090/ and login as root:<same password>
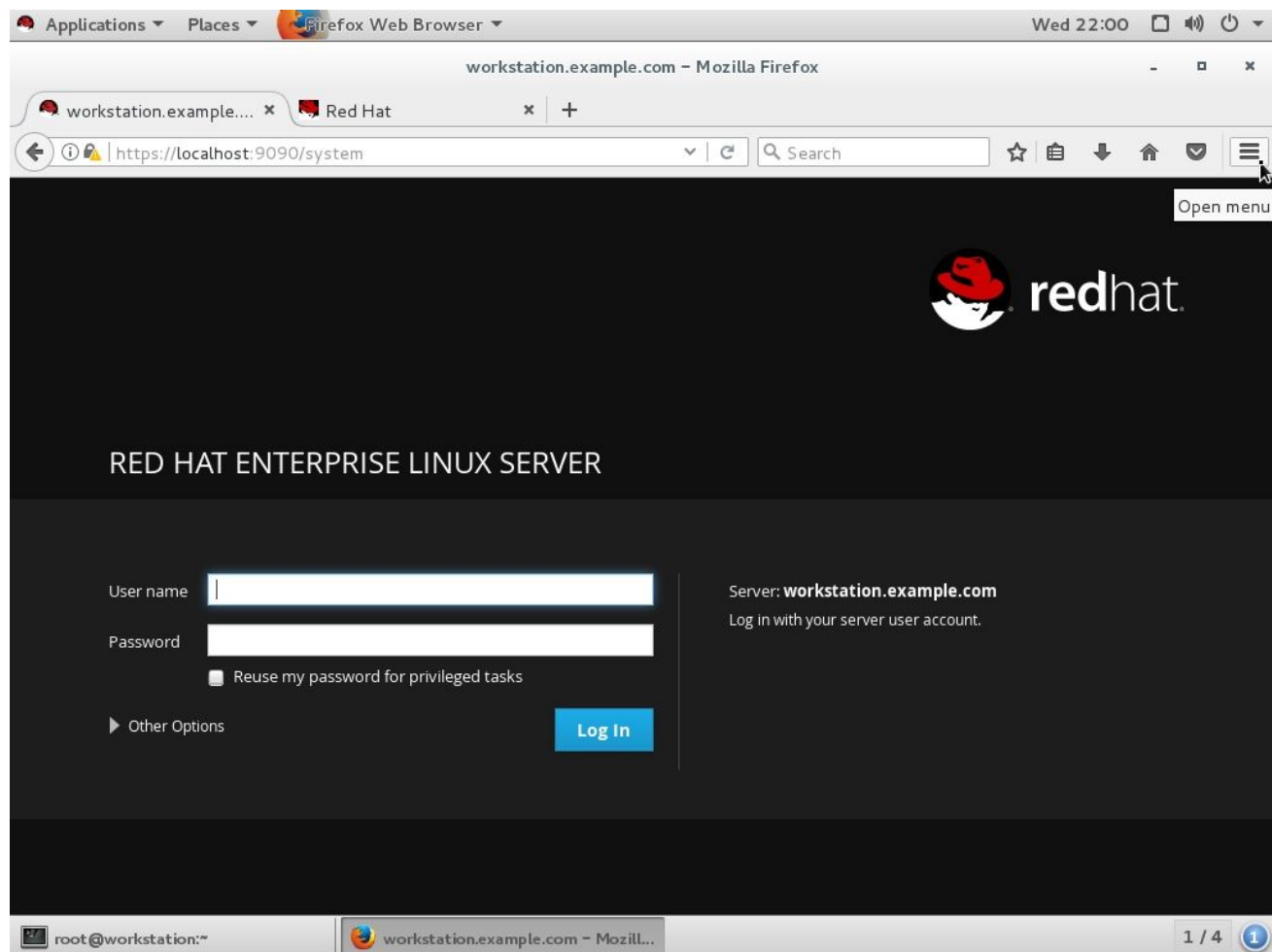
7. Explore system resource

Basic system information is displayed here. Real-Time system resource consumption of CPU, Memory, Disk I/O, and Network Traffic.

**You can also change system performance profile by clicking on "Performance profile: virtual-guest". You can change the performance profile to fit the system purpose.
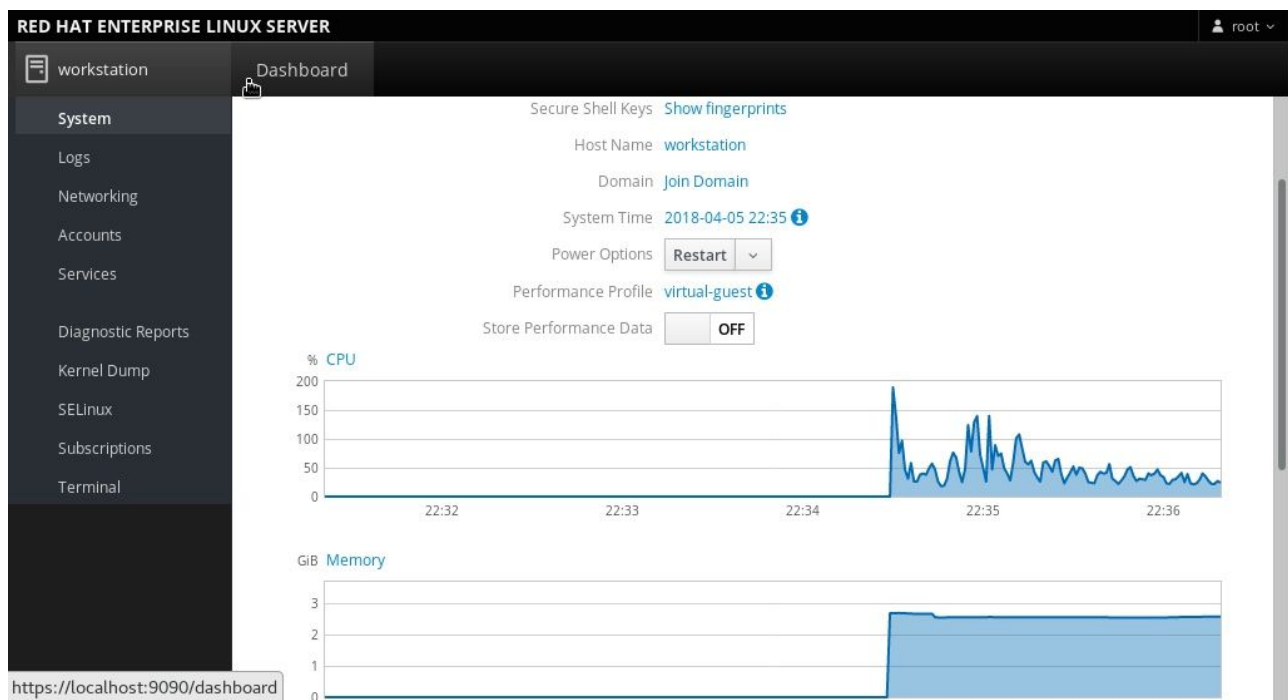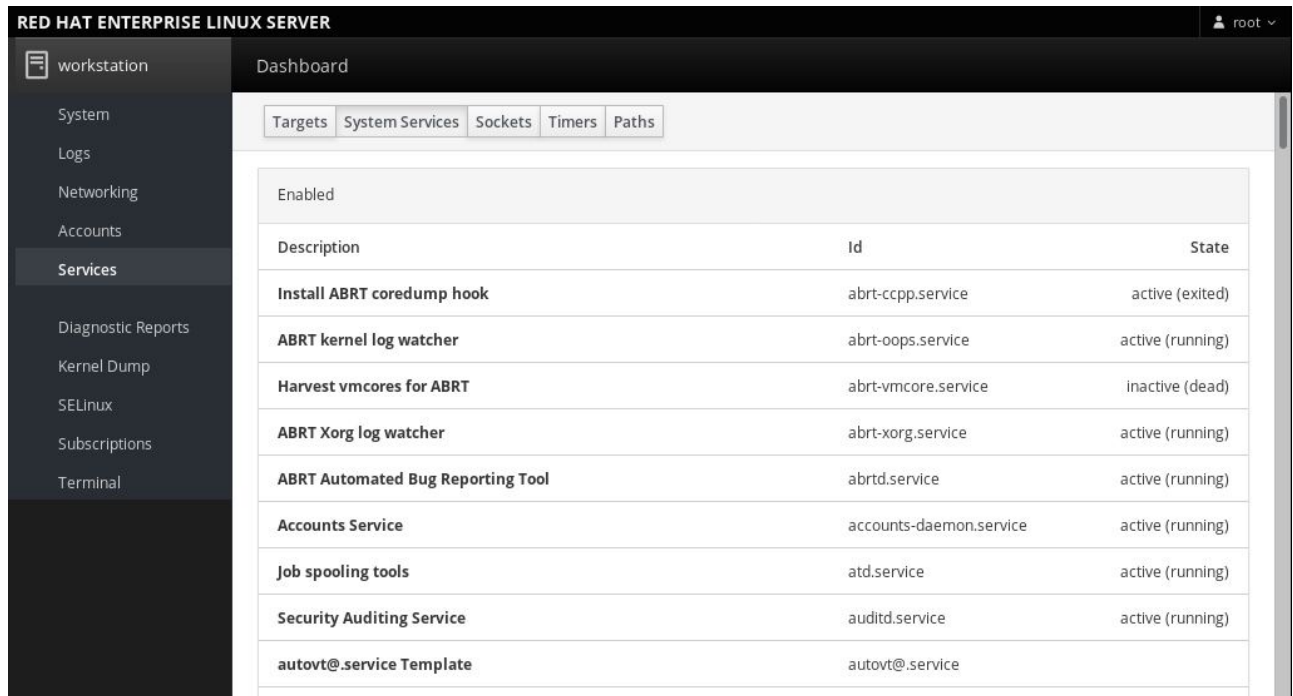
8. Explore system services

Click on *Services* on the left side of the web browser, Cockpit will display all available system services and their current state. You can explore individual services by clicking on them. Current version only displays service logs for that service. Later version of Cockpit will give you the ability to start, stop, or restart services.

9. Explore system logs

Cockpit allows you to review logs more efficiently. Click on *Logs* on the left side of the web browser. Logs can be filtered to display Errors, Warnings, Notices, or All. Additional details of the error can be viewed by clicking on the error. Please try a few.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

10. Explore storage volumes

Click on the *Storage* tab to review the latest storage logs and partition info of the system. Cockpit also gives you the ability to create partition table, delete, and format partitions, but we will not exercise them in this lab.

11. Explore Networking

Click on the *Networking* tab to review the latest networking information and logs. You can also create Bond, Team, Bridge, and VLAN through Cockpit.

In the following exercise, we will walk you through enabling eth1 for DHCP:

```
A. Under Networking:Interfaces section, click on eth1

B. You should see real-time data on sending and receiving network
   packets

C. To enable eth1, toggle the off switch to on
```

D. Under General, check box "Connect automatically"

E. *eth1* status should acquire the new IP info now

F. End of the exercise

11. Managing multiple servers

To add additional nodes:
1. Click on Dashboard
2. Click on plus (+) button
3. Enter hostname: "server1"
4. Click on "add"

Repeat the process again and add "server2"

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros
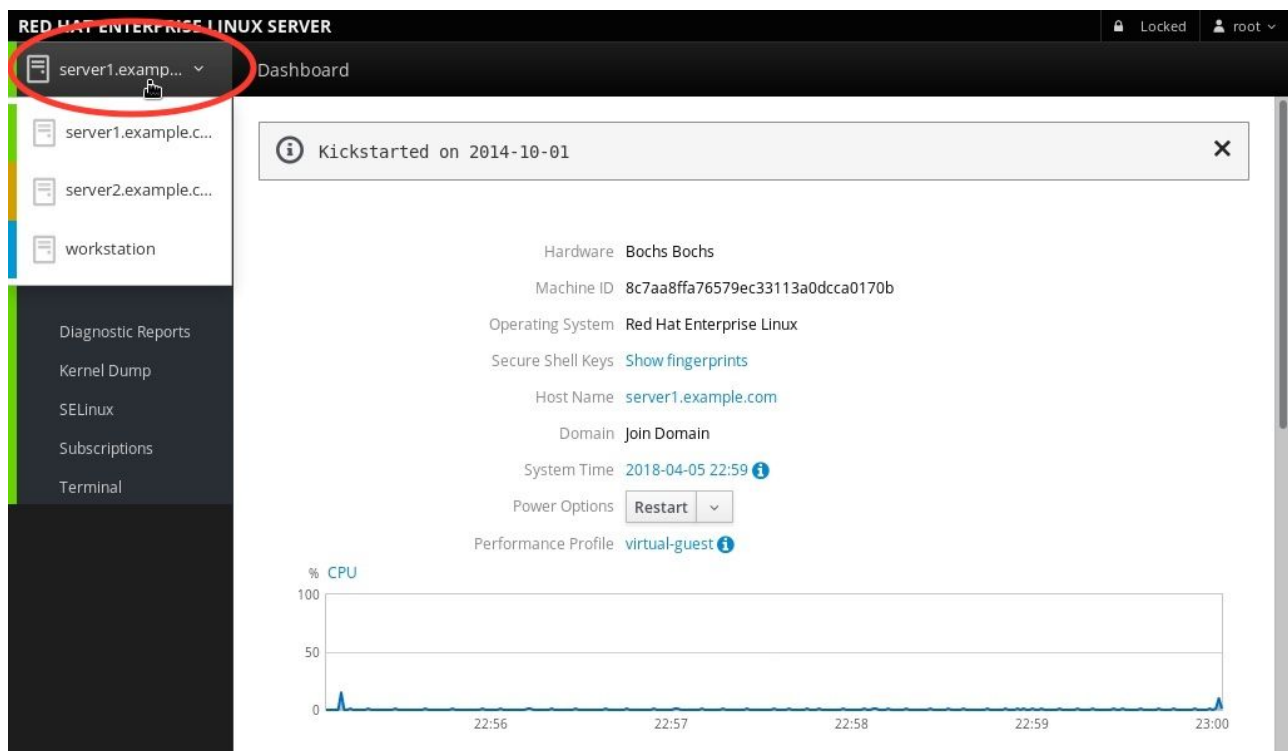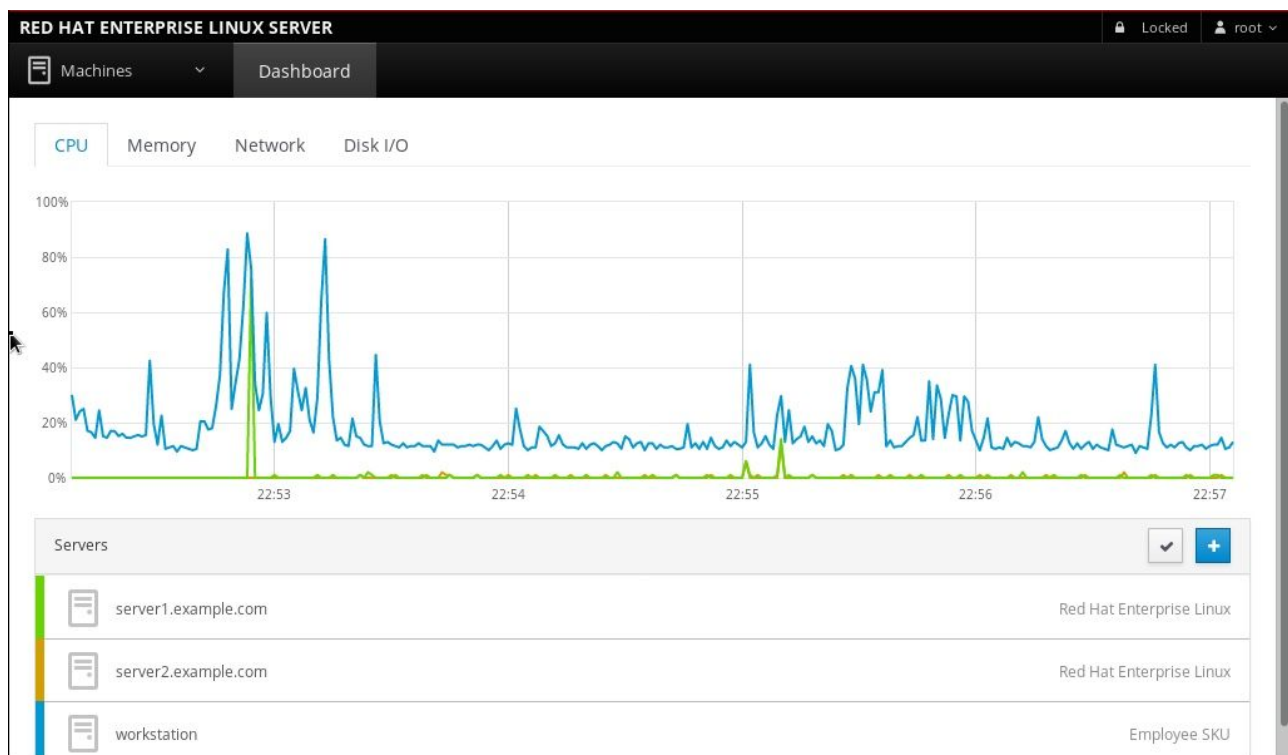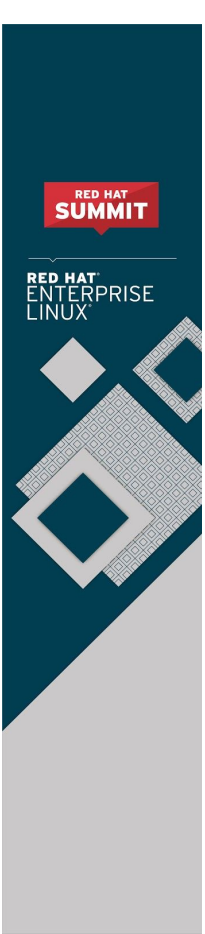
Red Hat Summit, San Francisco May 8th-10th 2018

Once server1 and server2 are added to the dashboard. You are now able to monitor the system resources of all 3 servers from a single dashboard. In previous steps, we also demonstrated using cockpit to manage system services, read system logs, configure network devices and etc. You can also perform those tasks on any of the 3 servers (workstation, server1, and server2) from Cockpit dropdown menu.

12. Miscellaneous

There are many other features within Cockpit. Feel free to explore around.

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Cockpit is also gaining new features and enhancements with each release. You can find the latest release information and features at http://cockpit-project.org/blog/category/release.html

OpenSCAP Scanning

Red Hat Enterprise Linux provides tools that allow for a fully automated compliance audit. These tools are based on the Security Content Automation Protocol (SCAP) standard and are designed for automated tailoring of compliance policies.

OpenSCAP is a compliance scanning, reporting and remediation toolset that is policy driven.

The compliance policy can vary substantially across organizations and even across different systems within the same organization. Differences among these policies are based on the purpose of these systems and its importance for the organization. The custom software settings and deployment characteristics also raise a need for custom policy checklists.

Security Compliance Tools Supported on RHEL7

SCAP Workbench — The scap-workbench graphical utility is designed to perform configuration and vulnerability scans on a single local or remote system. It can be also used to generate security reports based on these scans and evaluations.

OpenSCAP — The oscap command-line utility is designed to perform configuration and vulnerability scans on a local system, to validate security compliance content, and to generate reports and guides based on these scans and evaluations.

Script Check Engine (SCE) — SCE is an extension to the SCAP protocol that allows administrators to write their security content using a scripting language, such as Bash, Python, or Ruby. The SCE extension is provided in the openscap-engine-sce package.

SCAP Security Guide (SSG) — The scap-security-guide package provides the latest collection of security policies for Linux systems. The guidance consists of a catalog of practical hardening advice, linked to government requirements where applicable. The project bridges the gap between generalized policy requirements and specific implementation guidelines.

Note: If you require performing automated compliance audits on multiple systems remotely, you can utilize OpenSCAP solution for Red Hat Satellite.(not covered in this lab)

SCAP is a vendor-neutral way of expressing security policy, and as such it is widely used in modern enterprises. SCAP specifications create an ecosystem where the format of security

content is well known and standardized while the implementation of the scanner or policy editor is not mandated. Such a status enables organizations to build their security policy (SCAP content) once, no matter how many security vendors do they employ.

SCAP Components

Languages — This group consists of SCAP languages that define standard vocabularies and conventions for expressing compliance policy.

eXtensible Configuration Checklist Description Format (XCCDF) — A language designed to express, organize, and manage security guidance.

Open Vulnerability and Assessment Language (OVAL) — A language developed to perform logical assertion about the state of the scanned system.

Open Checklist Interactive Language (OCIL) — A language designed to provide a standard way to query users and interpret user responses to the given questions.

Asset Identification (AI) — A language developed to provide a data model, methods, and guidance for identifying security assets.

Asset Reporting Format (ARF) — A language designed to express the transport format of information about collected security assets and the relationship between assets and security reports.

Enumerations — This group includes SCAP standards that define naming format and an official list or dictionary of items from certain security-related areas of interest.

Common Configuration Enumeration (CCE) — An enumeration of security-relevant configuration elements for applications and operating systems.

Common Platform Enumeration (CPE) — A structured naming scheme used to identify information technology (IT) systems, platforms, and software packages.

Common Vulnerabilities and Exposures (CVE) — A reference method to a collection of publicly known software vulnerabilities and exposures.

Metrics — This group comprises of frameworks to identify and evaluate security risks.

Common Configuration Scoring System (CCSS) — A metric system to evaluate security-relevant configuration elements and assign them scores in order to help users to

prioritize appropriate response steps.

Common Vulnerability Scoring System (CVSS) — A metric system to evaluate software vulnerabilities and assign them scores in order to help users prioritize their security risks.

Integrity — An SCAP specification to maintain integrity of SCAP content and scan results.

Trust Model for Security Automation Data (TMSAD) — A set of recommendations explaining usage of existing specification to represent signatures, hashes, key information, and identity information in context of an XML file within a security automation domain.

Each of the SCAP components has its own XML-based document format and its XML name space. A compliance policy expressed in SCAP can either take a form of a single OVAL definition XML file, data stream file, single zip archive, or a set of separate XML files containing an XCCDF file that represents a policy checklist.

For more examples please see the RHEL 7 and OpenSCAP documentation.

Getting Started

Install the required packages
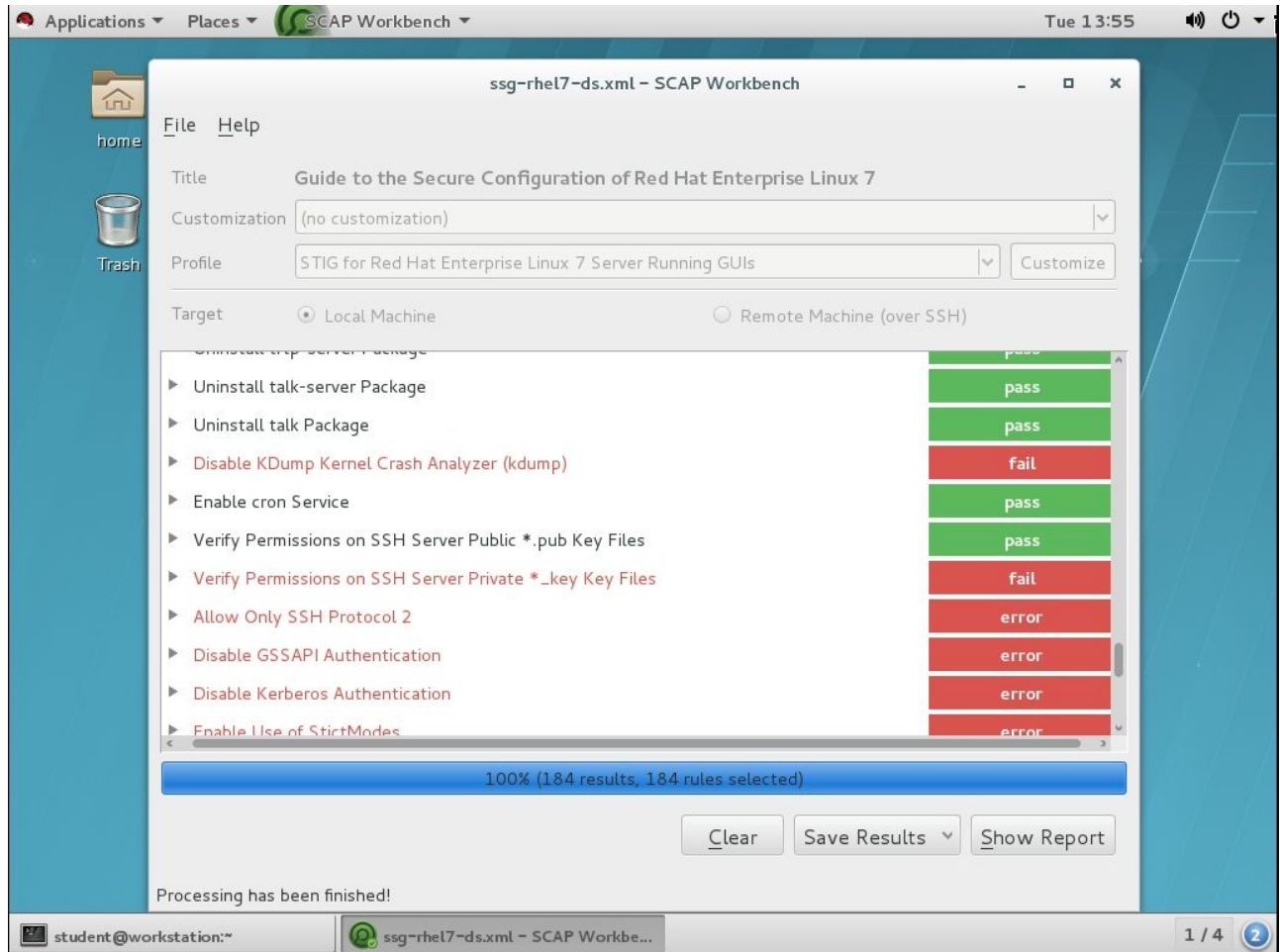
```
# yum install scap-workbench scap-security-guide
```

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

Run a Compliance Scan with SCAP Workbench

1. Run the scap-workbench utility

```
# scap-workbench
```

2. Select RHEL7 as your default security guide

3. Open Other Content (File Menu)
         ssg-> content-> ssg-rhel7-xccdf.xml

4. Select (no customization)

5. Select a profile from the list
         STIG for Red Hat Enterprise Linux 7 Running GUIs

6. Select Local Machine as the Target.

!! WARNING: DO NOT SELECT THE REMEDIATE CHECKBOX !!

The Definitive RHEL7 Hands-On Lab

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

4. Execute the Scan, then wait.
If you are running as root, no Password is necessary (just Cancel the Authentication Request)

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

5. Click on "Show Report" and review. The report is viewed through firefox so it may take a moment to load.

Relax and Recover (rear)

Introduced in RHEL 7.2, Relax-and-Recover is a setup-and-forget Linux bare metal disaster recovery solution. It is easy to set up and requires no maintenance so there is no excuse for not using it. The utility produces a bootable image and restores from backup using this image.

Some additional background points of interest:
- Supported boot media: ISO, PXE, OBDR tape, USB or eSata storage.
- Network protocols: sftp, ftp, http, nfs and cifs
- 3rd party Integration: IBM TSM, HP Dataprotector, Symantec NetBackup, SEP Sesam, CommVault Galaxy, EMC Networker (Legato), Bacula and a couple of others…

NOTE: *check with instructor regarding nfs server and hostname changes*

Getting Started

1. Install the required packages

```
# yum install rear genisoimage syslinux -y
```

2. In order to properly configure Grub to boot our rescue image, we need generate a hashed password.

```
# grub2-mkpasswd-pbkdf2
Enter password: redhat
Reenter password: redhat
PBKDF2 hash of your password is
grub.pbkdf2.sha512.10000.50F96A687B8BCFAE6090234ACB363ABD465A1DBA9E086C
26D0E268354EAF7238CEA68889D7B7B0C486BF2AD30A3035C4259EBDFE8A19F4140D241
386100B6AAE.C9615BDD47B42A98575DA91854E2017A71B4345F4D52AB03A54347A222C
00A3D74A4987CE511DFA76E6B70C57ECE45A3106E4C53C160AFE1C2A3F61123FCFF15
```

3. We will be doing a NFS based backup and recovery. Modify the configuration file **/etc/rear/local.conf** on the workstation with these settings. You should use the password hash that was created above!

Notes:

1. *You can copy /var/tmp/rear-local-config.template to /etc/rear/local.conf to save some time.*
2. *Be sure to check the hostnames for required changes.*
3. Use "vgs" on the commandline to determine the root logical volume and assign it with INCLUDE_ONLY_VG={"rootvg"}

```
OUTPUT=ISO
OUTPUT_URL=nfs://server.example.com/tmp
BACKUP=NETFS
BACKUP_URL=nfs://server.example.com/tmp
SSH_ROOT_PASSWORD="redhat"
INCLUDE_ONLY_VG={"rhel"}
BACKUP_PROG_EXCLUDE=("${BACKUP_PROG_EXCLUDE[@]}" '/media' '/var/tmp' '/var/crash')
NETFS_KEEP_OLD_BACKUP_COPY=
GRUB_RESCUE=y
GRUB_RESCUE_PASSWORD="grub.pbkdf2.sha512.10000.50F96A687B8BCFAE6090234ACB363ABD465
A1DBA9E086C26D0E268354EAF7238CEA68889D7B7B0C486BF2AD30A3035C4259EBDFE8A19F4140D241
386100B6AAE.C9615BDD47B42A98575DA91854E2017A71B4345F4D52AB03A54347A222C00A3D74A498
7CE511DFA76E6B70C57ECE45A3106E4C53C160AFE1C2A3F61123FCFF15"
```

4. Verify that an NFS server has been configured for the lab environment

```
# showmount -e server.example.com
Export list for server.example.com:
/tmp *
```

5. Now create the disaster recovery image (est: 2 minutes)

```
#  rear -v mkrescue

Relax-and-Recover 1.17.2 / Git
Using log file: /var/log/rear/rear-workstation.log
Creating disk layout
Creating root filesystem layout
Copying files and directories
Copying binaries and libraries
Copying kernel modules
Creating initramfs
Making ISO image
```

```
Wrote ISO image: /var/lib/rear/output/rear-workstation.iso (135M)
Modifying local GRUB configuration
Copying resulting files to nfs location
```

You should regularly check the current rescue image against the current host configuration. If file systems got resized, hardware changed or whatever then it is time to rerun mkrescue. You could easily put an entry in a cron: **rear checklayout || /usr/sbin/rear mkrescue**

6. Inspect the grub config, toward the bottom you should see a stanza for Relax and Recover

cat /boot/grub2/grub.cfg

```
menuentry "Relax and Recover" --class os --users "" {
    set root='hd0,msdos1'
    linux  /rear-kernel selinux=0 console=ttyS0,9600 console=tty0
    initrd /rear-initrd.cgz
    password_pbkdf2 rearadmin
"grub.pbkdf2.sha512.10000.505CC0C55BEE0DDCD233AD0B99F76A4420F389973B42D
7C7FAD6004558831E06E11D90A97C0FBA57D40588F876F3856C5467ADAFE3E6D73A6C08
A44F12482172.F6BA6F0539C6CC2DCEDB94885ED617D3F472214FEF2D8404BAD8D3A88C
6F709E3F21EA383436A0FB04369094FC66270B1BAE6298E623EDB972F8E915009D200F"
}
```

7. Inspect additional /etc/grub.d configs for rear modifications

cat /etc/grub.d/01_users
```
#!/bin/sh
cat << EOF
set superusers="rearadmin"
password_pbkdf2 rearadmin
grub.pbkdf2.sha512.10000.F0288AFF505F766DF9300990B05C3881E15FA9D5A80CDF
CF85D836D9D8E0A6CF8F1B296F974349AEE091785F0215BC0289EF3732C621A2D6B77D0
03A35BC9B22.D3107EC335B9BC9FA877EE097AD5C639BBF1094B0955692E045E00FF6CA
B326BB23ACBF9C72D50152101F0FFB4E179D46D9D7B63CA63A3DB8CFCB9B77473487F
EOF
```

8. Inspect the /boot directory for the rear recovery images

ls -l /boot/rear*
```
-rw-r--r--. 1 root root 134443207 Apr 17 20:08 /boot/rear-initrd.cgz
-rwxr-xr-x. 2 root root   4902000 May  5  2014 /boot/rear-kernel
```

System Backup

1. Create backup files (est: 10 minutes)

rear -d -v mkbackup

2. Inspect the NFS server directories

```
# mount server1:/tmp /mnt
# cd /mnt
# ls -la workstation
drwxr-x---. 2 root root       4096 Apr 17 22:41 .
drwxrwxrwt. 8 root root       4096 Apr 17 19:03 ..
-rw-------. 1 root root   11146363 Apr 17 22:41 backup.log
-rw-------. 1 root root 2230269768 Apr 17 22:41 backup.tar.gz
-rw-------. 1 root root        202 Apr 17 22:35 README
-rw-------. 1 root root     195409 Apr 17 22:35 rear.log
-rw-------. 1 root root  142227456 Apr 17 22:35 rear-workstation.iso
-rw-------. 1 root root          0 Apr 17 22:41 selinux.autorelabel
-rw-------. 1 root root        283 Apr 17 22:35 VERSION
```

System Recovery

1. Reboot the system and select "Relax and Recover" from the menu

2. Notice the grub username and password from the output of grub.cfg above? Your username and password are:

```
Username: rearadmin
Password: redhat
```

3. Login at "root". There will NOT be a password.

```
Workstation login: root
```

4. Finally, run the recovery (est: 5 minutes)

```
# rear recover
```

5. Reboot the system

```
# reboot
```

Final Words

Rear also provides the means to restore the image on to different hardware. Therefore, relax and recover can be used as a migration utility.

!! THANK YOU !!
FOR ATTENDING THE RHEL 7 LAB
Red Hat SUMMIT 2017

Improve your Enterprise Linux knowledge with the Pros

Red Hat Summit, San Francisco May 8th-10th 2018

APPENDIX A (rhel7-lab-prep.yml)

```
---
- hosts: localhost

  tasks:
        - name: Root Password (ALL)
        user: name=root
password="$5$PK7KM9pT$AehNIMYtW22TEE.D0U3ORWCDFMFvARwJLf8YPvVgAf."

        - name: Student Password (ALL)
        user: name=student
password="$5$OJx9uJm8$4VYM2llNxqX/7I4M3vXZM8ZJ9FE2msmHpcOqXlDesC8"

        - name: Required packages for lab section (ALL)
        yum: name=bash-completion state=installed

        - name: Required packages for lab section (SYSTEMD)
        yum: name=httpd,mariadb,mariadb-server,php,php-mysql state=installed

        - name: Required state for lab section (SYSTEMD mariadb)
        service: name=mariadb state=stopped enabled=no

        - name: Required state for lab section (SYSTEMD httpd)
        service: name=httpd state=stopped enabled=no

        - name: Required packages for lab section (PCP)
        yum: name=pcp,pcp-gui state=installed

        - name: Required state for lab section (PCP pmcd)
        service: name=pmcd state=stopped enabled=no

        - name: Required state for lab section (PCP pmlogger)
        service: name=pmlogger state=stopped enabled=no

        - name: Required packages for lab section (NUMA)
        yum: name=numad,numactl state=installed

        - name: Required state for lab section (NUMA)
        service: name=numad state=stopped enabled=no

        - name: Required packages for lab section (LVM)
        yum: name=system-storage-manager state=installed

        - name: LAB Setup (LVM - create block devices)
        shell: for i in {0..5} ; do [[ -e /var/tmp/disk${i}.img ]] || fallocate -l 100M
/var/tmp/disk${i}.img ; done

        - name: LAB Setup (LVM - create new partitions)
        shell: for i in {0..5} ; do printf "o\nn\np\n\n\n\nt\n8e\nw\n" | fdisk
/var/tmp/disk${i}.img  ; done

        - name: LAB Setup (LVM - setup kpartx loopbacks)
        shell: for i in {0..5} ; do kpartx -av /var/tmp/disk${i}.img  ; done

        - name: Required packages for lab section (NMCLI)
        yum: name=teamd state=installed

        - name: Required packages for lab section (COCKPIT)
        yum: name=cockpit state=installed
```

```
        - name: Required state for lab section (COCKPIT)
        service: name=cockpit state=stopped enabled=no

        - name: Required packages for lab section (OPENSCAP)
        yum: name=scap-workbench,scap-security-guide state=installed

        - name: Required packages for lab section (REAR)
        yum: name=rear,genisoimage,syslinux state=installed

        - name: LAB Setup (REAR - create config template)
        shell:
        cmd: |
        cat > /var/tmp/rear-local-config.template << EOF
                OUTPUT=ISO
                OUTPUT_URL=nfs://server1.example.com/tmp
                BACKUP=NETFS
                INCLUDE_ONLY_VG={"rootvg"}
                BACKUP_URL=nfs://server1.example.com/tmp
                SSH_ROOT_PASSWORD="redhat"
                BACKUP_PROG_EXCLUDE=("${BACKUP_PROG_EXCLUDE[@]}" '/media' '/var/tmp'
'/var/crash')
                NETFS_KEEP_OLD_BACKUP_COPY=
                GRUB_RESCUE=y

GRUB_RESCUE_PASSWORD="grub.pbkdf2.sha512.10000.50F96A687B8BCFAE6090234ACB363ABD465A1DBA9E
086C26D0E268354EAF7238CEA68889D7B7B0C486BF2AD30A3035C4259EBDFE8A19F4140D241386100B6AAE.C9
615BDD47B42A98575DA91854E2017A71B4345F4D52AB03A54347A222C00A3D74A4987CE511DFA76E6B70C57EC
E45A3106E4C53C160AFE1C2A3F61123FCFF15"
                EOF
        args:
        creates: /var/tmp/rear-local-conf.template
```

APPENDIX B (LAB Requirements)

2 VMs

       2 cpus

       2 GB memory

       10GB disk

       4 nics (1 configured and active, rest on same subnet)

Workstation

       GNOME / "Server with GUI"

       Firefox

       Ansible

       Repos = rhel-7-server-rpms, rhel-7-server-extras-rpms, rhel-7-server-supplementary rpms, epel

Server

       nfs server with /etc/exports = "/tmp   *(rw,no_root_squash)"