SysAdmin Notes for RHCE

Somenath Sinha

January 2018

Contents

Part I

Advanced System Management

Chapter 1

Configuring Authentication

1.1 Understanding RedHat Identity Management

RedHat Identity Management is based on the FreeIPA (Identity, Policy, Audit) Project. The project bundles together several services in one solution. Some of the services are:

Options	Description
389 Directory Server	This is an **LDAPv3** Directory Server – a replacement for *OpenLDAP*.
Single Sign-on	Provided by MIT **Kerberos** KDC.
Integrated Certificate System	Based on the *Dogtag* project.
Integrated NTP Server	*Chrony* must be disabled to use this!
Integrated DNS Server	Based on *ISC Bind* Service.

Thus, the Identity Management provided by IPA bundles up some pretty complicated projects together and provides an easy interface to manage them all. However, IPA conflicts with other products, such as other *LDAP, Kerberos, Certificate System, NTP or DNS* servers shouldn't be running on the same system. Thus, ideally Identity Management should be set up on a dedicated server.

Kerberos is a Network Authentication Protocol that makes clients prove their identity to the server, and vice versa. Other than the authentication tools, it also supports strong cryptography over the network to keep the data safe in-transit.

1.1.1 IdM Server Components and Requirements

An IdM server needs some from of *Host Name Resolution*, which can be either through a DNS server or via the `/etc/hosts` file. Note that the hostname of the Identity Management server itself must also be specified.

Next we need both the **ipa-server** package, which installs the server components, and the **ipa-client** package which installs the client components. While the client package isn't required to be installed on the server, while configuring a client that talks to an IPA server, then this is one of the solutions available. Another method would be to use **authconfig**.

After the required RPM packages have been installed, we will run **ipa-server-install** which provides an easy, scripted way to install an IPA server, and all we have to do is answer a few questions, at the end of which we get a fully-functional IPA server.

The **ipa** tool is a generic client interface, that's also the administration interface. Thus, it can perform several tasks such as adding users (`ipa user-add <username>`), set the password for an user (`ipa passwd <username>`), see the IPA properties for a user account (`ipa user-find <username>`), etc. *ipa-xxx* can be used instead as well, where *xxx* represents the different tasks. Authentication can also be configured using **authconfig**.

1.1.2 Preparing IdM Installation

First and foremost, the *host name resolution* must be set up, since the installation will fail if the host can't find its own name. Additionally, the DNS name must also be known since the Kerberos domain that we'll configure will be based on the DNS name.

Next, the **nscd** service must be disabled, along with any existing LDAP and Kerberos services. If NTP and ISC Bind must also be disabled if installed (due to possible conflicts). The LDAP, Kerberos, NTP, DNS and certificate system ports must be opened in the firewall.

1.1.3 Installing IdM

The **ipa-server**, **bind** and **nds-ldap** packages must be installed using, following which, we have to run the command **ipa-server-install**, which will perform a wizard-like scripted installation.

```
# yum -y install ipa-server bind nds-ldap
# ipa-server-install
```

If we don't want to enter the information interactively, we can also provide them as options. The hostname, the domain name, a realm name (domain name in upper-case).

```
# ipa-server-install --hostname=vmPrime.somuVMnet.local -n somuVMnet.local -r
↪  SOMUVMNET.COM -p password -a password -U --no-ntp
```

The appropriate flags needed are:

Options	Description
–hostname	The hostname of the server
-n	The Domain name of the server
-r	Realm Name (Domain name in All-Caps)
-p	Password for Directory Manager
-a	Password for admin user
-U	Unattended Install; Doesn't prompt for anything
–no-dns	Do not install the DNS Server

Now, the SSH Daemon must be restarted to ensure that SSH obtains Kerberos credentials:

```
# systemctl restart sshd
```

Then, we generate a new Kerberos ticket and then verify Kerberos authentication for the default admin user by using:

```
# kinit admin
Password for admin@SOMUVMNET.LOCAL:
```

```
# klist
Ticket cache: KEYRING:persistent:0:0
Default principal: admin@SOMUVMNET.LOCAL

Valid starting       Expires              Service principal
2017-12-26T15:44:09  2017-12-27T15:44:05  krbtgt/SOMUVMNET.LOCAL@SOMUVMNET.LOCAL
```

This will show us if we have a valid Kerberos ticket. For any administrative tasks on the IPA server, having a valid Kerberos ticket is mandatory. Finally, we need to verify IPA access using:

```
# ipa user-find admin
```

This will show us the details of the admin user as created in the LDAP directory, along with all of its properties.

1.1.4 Understanding Kerberos Tickets

Kerberos tickets are the keys to the proper functioning of Identity Management. To be able to manage the IdM server, we need to log in to the IdM Domain and generate a Kerberos ticket for the admin user, using the command:

```
# kinit admin
```

We can check the validity of the ticket at any time using:

```
# klist
```

1.1.5 Managing the IdM Server

After generating a Kerberos ticket with `kinit admin`, we use the **ipa** command to manage the IdM server. `ipa help commands` shows us a short overview of all the available commands and their usage. For any specific command, we have `ipa help <command>` (such as `ipa help user-add`).

Another method to manage the IdM server is to navigate, using our web browser, to `https://vmPrime.somuVMnet.local` (if our server is named *vmPrime.somuVMnet.local*). This will load the IPA management web interface. Through this interface, after we've authenticated as admin, we will be guided through the various aspects of setting up the IdM environment.

1.1.6 Creating User Accounts

The required commands to create an user called *lisa* and verify the account creation are:

```
# kinit admin
# ipa user-add lisa
# ipa passwd lisa
# ipa user-find lisa
```

1.2 Using authconfig to Setup External Authentication

There are the **authconfig** utilities to setup external authentication (via LDAP), which consist of: *authconfig, authconfig-tui* and *authconfig-gtk*. The GUI utility can be installed using `yum -y install authconfig-gtk`. The utility is started with `authconfig-gtk`.

In the **authconfig-gtk** utility, we have to choose LDAP as the User Account Database in the Identity and Authentication tab. This might prompt for the installation of two packages: *nss-pam-ldapd* (the package that integrates the three) and *pam_krb5* (the package that integrates PAM with Kerberos). Now, we can enter the details for the LDAP server to setup authentication.

In cases of servers which don't have a GUI (or there is some inconvienience with the GUI, such as the apply button hidden by the status bar, etc.), the **authconfig-tui** is a very good alternative. In case of automated scripts, however, the **authconfig** command line utility is the best option.

1.3 Configuring a System to Authenticate using Kerberos

To connect a system for authentication to an LDAP server using Kerberos credentials, a part of the configuration has to be done with authconfig. But even before that, certain things must be ensured. *First*, we need to make sure that the IP address of the server we're trying to connect to can be resolved from the hostname, using `/etc/hosts`:

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
90.0.16.100        vmDeux.somuVMnet.com        vmDeux
```

This is important so that we can use the FQDN of the server later while using the `authconfig-tui` utility. Next, the system must be configured to use the DNS component hosted within the IPA server. For this, all we need to do is add the IP address of the IPA server as the first nameserver entry in `/etc/resolv.conf`:

```
# Generated by NetworkManager
search somuvmnet.local
nameserver 90.0.16.100 # IP Address of DNS Server @ vmDeux.somuVMnet.local
nameserver 8.8.8.8
nameserver 202.38.180.7
```

It is important to place the IP address of the DNS server for a nameserver as the first entry because that's the only one configured to *know* the custom FQDNs of the machines on our network. So, this connectivity is essential since the Kerberos client needs to be connected to the Kerberos server.

Finally, we can start the `authconfig-tui` utility, and enter the following details:

```
# In Authentication Configuration:
[*] Use LDAP
[*] User Kerberos Authentication

# LDAP Settings
[*] Use TLS
Server:        ldap://vmdeux.somuvmnet.local
Base DN:        dc=somuvmnet, dc=local
```

```

# Kerberos Settings
Realm:        SOMUVMNET.LOCAL
[*] Use DNS to resolve hosts to realms
[*] Use DNS to resolve KDCs for realms
```

First, we've just setup the system to use LDAP using Kerberos authentication. Next, we've made it necessary to use a TLS certificate to ensure the security of the connection. Then, the details of the LDAP server have to be entered.

In the Kerberos authentication step, the *Realm* refers to the Kerberos realm that the server is a part of. If we've setup the DNS component of the IPA server properly, then the system is able to detect the KDCs properly for each realm, as well as assign hosts to their realm appropriately. Now, the TLS Certificate for the IPA Server have to be downloaded and put in the `/etc/openldap/cacerts` directory (from whichever location the IPA Server stored them in, typically `/root/cacert.p12` for the root user):

```
# cd /etc/openldap/cacerts/
# scp vmdeux.somuvmnet.local:/root/cacert.p12 .
```

At this point, we should be good to go. We can verify the LDAP connectivity by trying to login as an LDAP user. For this we use (for an LDAP user lisa):

```
# su - lisa
Last login: Tue Dec 26 18:52:05 IST 2017 on pts/0
su: warning: cannot change directory to /home/lisa: No such file or directory
-sh-4.2$
```

The warning is natural if no home directory has been configured yet.

1.3.1 Troubleshooting Authentication

When authentication doesn't work, for some reason related to the certificates, then there is an easy fix as well. Depending on whether our LDAP and Kerberos credentials are being cached by **nslcd** or **sssd**, we can edit their configuration file to ignore the validity of the certificate. This is because the *self-signed cacert* may not meet the standards dictated and required by the program. For this, we can add to `/etc/nslcd.conf`:

```
tls_reqcert never
```

If SSSD is used instead, then we can edit `/etc/sssd/sssd.conf` and add the following line:

```
ldap_tls_reqcert = never
```

When using Certificates that are well signed from an External Certificate Authority, this of course becomes unnecessary.

1.4 Understanding authconfig Configuration Files

1.4.1 Authconfig Configuration

The primary configuration of the authconfig utitlity is located at `/etc/sysconfig/authconfig`. The contents of this file is used by other config files, such as `USELDAP=yes`.

```
CACHECREDENTIALS=yes
FAILLOCKARGS="deny=4 unlock_time=1200"
FORCELEGACY=no
FORCESMARTCARD=no
IPADOMAINJOINED=no
IPAV2NONTP=no
PASSWDALGORITHM=sha512
USEDB=no
USEECRYPTFS=no
USEFAILLOCK=no
USEFPRINTD=no
USEHESIOD=no
USEIPAV2=no
USEKERBEROS=yes
USELDAP=yes
USELDAPAUTH=no
USELOCAUTHORIZE=yes
USEMKHOMEDIR=no
USENIS=no
USEPAMACCESS=no
USEPASSWDQC=no
USEPWQUALITY=yes
USESHADOW=yes
USESMARTCARD=no
USESSSD=yes
USESSSDAUTH=no
USESYSNETAUTH=no
USEWINBIND=no
USEWINBINDAUTH=no
WINBINDKRB5=no
```

These are the settings we provided to the **authconfig** utility.

1.4.2 SSSD Configuration

Things like the Kerberos password, the LDAP search base, etc. and other IPA specific settings are stored in the `/etc/sssd/sssd.conf` file, to ensure that the connection to the IPA Server is successfully initiated and it's possible to login and use the services provided by it. Typical contents of this file look like:

```
[sssd]
config_file_version = 2
services = nss, pam
# SSSD will not start if you do not configure any domains.
# Add new domain configurations as [domain/<NAME>] sections, and
# then add the list of domains (in the order you want them to be
# queried) to the "domains" attribute below and uncomment it.
; domains = LDAP

```

```
[nss]

[pam]

# Example LDAP domain
; [domain/LDAP]
; id_provider = ldap
; auth_provider = ldap
# ldap_schema can be set to "rfc2307", which stores group member names in the
# "memberuid" attribute, or to "rfc2307bis", which stores group member DNs in
# the "member" attribute. If you do not know this value, ask your LDAP
# administrator.
; ldap_schema = rfc2307
; ldap_uri = ldap://ldap.mydomain.org
; ldap_search_base = dc=mydomain,dc=org
# Note that enabling enumeration will have a moderate performance impact.
# Consequently, the default value for enumeration is FALSE.
# Refer to the sssd.conf man page for full details.
; enumerate = false
# Allow offline logins by locally storing password hashes (default: false).
; cache_credentials = true

# An example Active Directory domain. Please note that this configuration
# works for AD 2003R2 and AD 2008, because they use pretty much RFC2307bis
# compliant attribute names. To support UNIX clients with AD 2003 or older,
# you must install Microsoft Services For Unix and map LDAP attributes onto
# msSFU30* attribute names.
; [domain/AD]
; id_provider = ldap
; auth_provider = krb5
; chpass_provider = krb5
;
; ldap_uri = ldap://your.ad.example.com
; ldap_search_base = dc=example,dc=com
; ldap_schema = rfc2307bis
; ldap_sasl_mech = GSSAPI
; ldap_user_object_class = user
; ldap_group_object_class = group
; ldap_user_home_directory = unixHomeDirectory
; ldap_user_principal = userPrincipalName
; ldap_account_expire_policy = ad
; ldap_force_upper_case_realm = true
;
; krb5_server = your.ad.example.com
; krb5_realm = EXAMPLE.COM
```

This is probably one of the most important configuration files when **SSSD** is being used. If **nslcd** is being used instead, then the config file of interest is `/etc/nslcd.conf`.

1.4.3 Kerberos Configuration File

The Kerberos configuration file (for connecting to a Kerberos Server) is stored in `/etc/krb5.conf` and typically has contents like:

```
# Configuration snippets may be placed in this directory as well
includedir /etc/krb5.conf.d/

includedir /var/lib/sss/pubconf/krb5.include.d/
```

```
[logging]
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log

[libdefaults]
dns_lookup_realm = true
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
rdns = false
# default_realm = EXAMPLE.COM
default_ccache_name = KEYRING:persistent:%{uid}

dns_lookup_kdc = true
default_realm = SOMUVMNET.LOCAL
[realms]
# EXAMPLE.COM = {
#   kdc = kerberos.example.com
#   admin_server = kerberos.example.com
# }

SOMUVMNET.LOCAL = {
}

[domain_realm]
# .example.com = EXAMPLE.COM
# example.com = EXAMPLE.COM
somuvmnet.local = SOMUVMNET.LOCAL
.somuvmnet.local = SOMUVMNET.LOCAL
```

Here, the DNS domain to realm mapping is specified, to tell us which domain on the DNS belongs to which Kerberos realm.

1.4.4 NSSwtich Configuration

This file specifies the locations and the order in which passwords are searched for authentication. This includes the order in which passwords, shadow and groups are searched. The order is typically like:

```
passwd:     files sss ldap
shadow:     files sss ldap
group:      files sss ldap
```

This instructs the system to look for passwd files in the local file system first, then SSS and finally LDAP. The same is true for the two following categories of shadow and group.

1.4.5 NSLCD Configuration

While this file may be missing from newer versions of RHEL, this is an older version of LDAP configuration file. This file is supposed to be replaced by the `/etc/sssd/sssd.conf` file, and thus, all relevant settings should be provided in that file.

Chapter 2

Configuring iSCSI Target and Initiator

2.1 Understanding iSCSI Target and Initiator

SCSI (Small Computer System Interface) [read as *scuzzy*] is an alternative to ATA (a.k.a. IDE) Hard drives, which most consumer computers stick to. While SCSI drives provide significantly more throughput for certain scenarios, IDE suffices for most home computer usage. However, in case of servers, SCSI proves to be a much better alternative, since they provide more reliability and data transfer speed (much higher than ATA), owing to the fact that data transfer occurs in full-duplex mode (i.e., data can be read and written at the same time at full speeds). They also boast higher speeds (such as 15,000 RPM) as compared to ATA speeds (7200 RPM). Another reason servers tend to use SCSI (or related technologies, such as Serially Attached SCSI or SAS) is that the protocol makes it easy to *daisy-chain* several SCSI devices to the same controller, several times that of IDE devices. In fact, in the pre-USB era, SCSI was the go-to common interface for connecting peripherals or even devices such as printers.

Traditional SCSI devices use a long cable and a SCSI **Command Descriptor Block (CDB)** command to interact with the SCSI devices. In case of iSCSI, the same CDBs are used, but they're transmitted over IP packets over a network, instead of the cable. Thus, the SCSI devices are emulated by using a storage backend and presenting them on the network using iSCSI targets. SCSI targets are typically storage devices, while the hosts they're connected to are the initiators. Thus, this technology enables us to share PVs or LVs on the network, represented by iSCSI targets.

A **Storage Area Network (SAN)** is a network that provides access to a consolidated, block level data storage. *Block devices* provide a buffered data storage method, where data is transferred from the kernel buffer to the physical device. Also, data can be read and written in entire blocks. SANs thus present devices such as disk arrays as locally attached storage to servers. **Fiber Channel** or **FC** is a high speed network technology developed to enable fast data transfers between servers and SANs. Ethernet structures utilizing iSCSI technology can be as fast as their FC structure counterparts, thus making the technology enterprise ready for SAN creation.

2.1.1 iSCSI Operation

In the case of iSCSI storage, we have the SAN, on which runs a *iSCSI target* which can provide access to the storage backend. For any server that needs to access the files hosted by the SAN, it needs to run an **iSCSI initiator**, which performs a discovery operation first. During this, the SAN tells it about the iSCSI devices it has to offer. Once this is complete, the iSCSI initiator can login to the devices.

2.1.2 iSCSI Components

Both the iSCSI targets and the storage backends need to be set up for the SAN to operate. The storage backend can be an entire disk, a dedicated partition, a logical volume or even a file! The servers, running the iSCSI initiators, will see the iSCSI targets as new storage devices after successfully logging in to them. This can be verified by viewing the output of the `/proc/partitions` file. A tool called *lsscsi* can also alternatively used, although it is not installed by default.

2.1.3 Basic iSCSI Terminology

Terms	Description
IQN	iSCSI Qualified Name - an unique name assigned to each iSCSI target and initiator, used to identify them.
Initiator	The iSCSI client that is identified by its IQN.
Target	The service on the iSCSI server that gives access to the storage backend.
ACLs	Access Control Lists that are based on the node's IQNs.
Portal	Also known as **nodes**, this is the combination of the IP address and the port that are used by both targets and initiators to establish connections.
discovery	The process through which an indicator finds the available targets that are configured for a given portal.
LUNs	The *Logical Unit Number* is a number used to identify the logical unit (i.e., block devices shared through the target) being addressed by the iSCSI Controller.
login	The act which gives an initiator the relevant LUNs.
TPG	The *Target Portal Group* is a collection of IP Addresses and TCP ports to which a particular iSCSI Target will listen.

So, there can be more than one portals per server, and more than one targets per portal.

2.1.4 After connecting an initiator to an iSCSI Target

The new block devices thus accessed will appear as local devices (/dev/sdb, /dev/sdc etc.) Note that if a LUN is available and used by multiple servers, multiple devices can access the LUN post connection, i.e., multiple servers can use the disk at the same time. This is a bit dangerous, since it requires using clustering, for providing multiple servers to use the storage. Otherwise for a file system like XFS or Ext4, two servers writing to the same file can cause data loss.

To avoid this, shared file systems such as GFS2 can be used. In GFS2, the file system cache is shared among all the nodes. Thus, all nodes writing to the file system know what all the other nodes are doing.

2.2 Setting up an iSCSI Target

The iSCSI target works with several storage backend devices on the SAN. These storage devices can be anything that can be used for PVs when using traditional LVM. All these devices are put together in a volume group, which is then subdivided into several LVs. These LVs are each assigned a LUN.

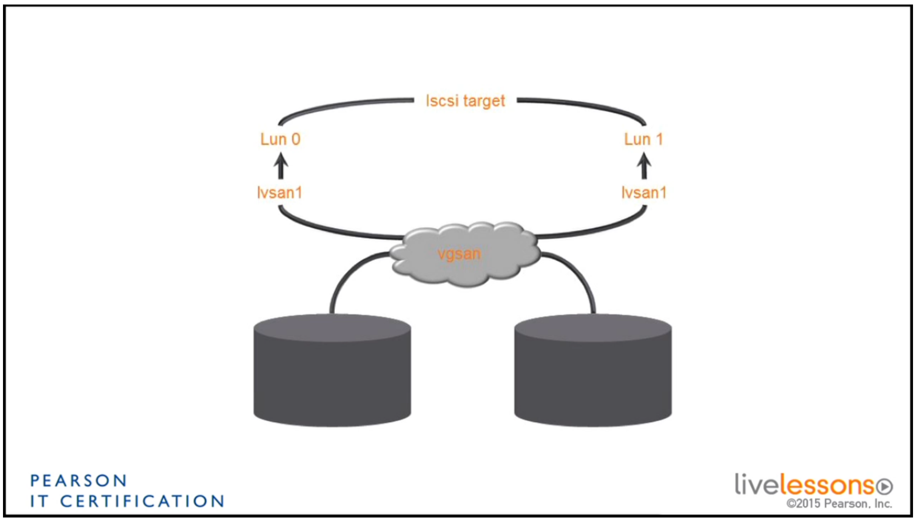

Figure 2.1: iSCSI Target Setup

These LUNs are presented using the iSCSI targets. Thus, the iSCSI configuration is created on top of a traditional LVM configuration.

2.2.1 Creating the LVM

Let us consider we have an empty disk of 1GB on which we want to build the iSCSI configuration. This can be verified using:

```
# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda               8:0    0   20G  0 disk
├─sda1            8:1    0    1G  0 part /boot
└─sda2            8:2    0   19G  0 part
├─centos-root 253:0    0  17G  0 lvm  /
└─centos-swap 253:1    0   2G  0 lvm  [SWAP]
sdb               8:16   0    1G  0 disk
sr0              11:0    1  8.1G  0 rom
```

We can directly create the VG `vgsan` on it, using:

```
# vgcreate vgSAN /dev/sdb
Physical volume "/dev/sdb" successfully created.
Volume group "vgSAN" successfully created
# pvs
PV         VG     Fmt  Attr PSize    PFree
/dev/sda2  centos lvm2 a--    <19.00g       0
/dev/sdb   vgSAN  lvm2 a--  1020.00m 1020.00m
# vgs
```

```
VG      #PV #LV #SN Attr   VSize    VFree
centos    1   2   0 wz--n-  <19.00g       0
vgSAN     1   0   0 wz--n- 1020.00m 1020.00m
```

The output of the `pvs` and `vgs` commands show that the PV `/dev/sdb` is now a part of *vgSAN*, which has a free space of 1020MB. Now we create two LVs *lvSAN1* and *lvSAN2* on the VG, using:

```
# lvcreate -L 500M -n lvSAN1 vgSAN
Logical volume "lvSAN1" created.
# lvcreate -l 100%FREE -n lvSAN2 vgSAN
Logical volume "lvSAN2" created.
# lvs
LV     VG     Attr       LSize   Pool Origin ... Convert
root   centos -wi-ao---- <17.00g
swap   centos -wi-ao----   2.00g
lvSAN1 vgSAN  -wi-a----- 500.00m
lvSAN2 vgSAN  -wi-a----- 520.00m
```

Now, our LVM setup is complete, and we can proceed with the iSCSI setup. For this, first of all we need to install the iSCSI software, called **targetcli**. The targetcli utility is a relatively new one capable of managing multiple types of storage devices.

2.2.2 Creating the iSCSI configuration using targetcli

We start the utility using:

```
# targetcli
Warning: Could not load preferences file /root/.targetcli/prefs.bin.
targetcli shell version 2.1.fb46
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/>
```

This interface can be navigated using the same commands as the bash shell. Using the `cd` command produces the output:

```
/> ls
o- / ..................................................................... [...]
  o- backstores .......................................................... [...]
  | o- block .............................................. [Storage Objects: 0]
  | o- fileio ............................................. [Storage Objects: 0]
  | o- pscsi .............................................. [Storage Objects: 0]
  | o- ramdisk ............................................ [Storage Objects: 0]
  o- iscsi ........................................................ [Targets: 0]
  o- loopback ..................................................... [Targets: 0]
/>
```

The *backstores* part allow us to work with the different storage devices. To enter backstores, we simply enter `cd` command, and select it from the menu. This will change the prompt to `/backstores>`. Here, we can see it contains the *block*, the *fileio*, the *pSCSI* and the *ramdisk* devices. Their significance is explained below:

Types	Description
Block	Refers to any block device that we want to share using iSCSI. This includes all traditional disks, partitions and even LVMs.
fileio	Refers to a file that can be used as a storage source. This refers to a big file created using a tool such as `dd`.
pscsi	Physical SCSI - a SCSI pass-through backstore is created for such devices.
ramdisk	RAM storage, wiped with every reboot, and is thus a **very** bad idea.

Now, since all our LVs are block devices (by their very nature), we have to create our LUNs inside the block category. This we can do using:

```
/backstores> block/ create block1 /dev/vgSAN/lvSAN1
Created block storage object block1 using /dev/vgSAN/lvSAN1.
```

The command instructs the targetcli utility to enter the block category, and create a block device called *block1* from the `/dev/vgSAN/lvSAN1` device. We can create another block device for the partition and a 1G custom file device using:

```
/backstores> block/ create block2 /dev/vgSAN/lvSAN2
Created block storage object block2 using /dev/vgSAN/lvSAN2.
/backstores> fileio/ create file1 /root/diskFile1 1G
Created fileio file1 with size 1073741824
```

When creating a file, we can merely specify the size (1GB) and the name & location (/root/diskFile1) to have the `targetcli` utility create the file for us, instead of copying from `/dev/zero` to a file using `dd`. All the different devices thus added can be seen with:

```
/backstores> ls
o- backstores ............................................................ [...]
  o- block ............................................... [Storage Objects: 2]
  | o- block1 ............ [/dev/vgSAN/lvSAN1 (500.0MiB) write-thru deactivated]
  | | o- alua ................................................ [ALUA Groups: 1]
  | |   o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
  | o- block2 ............ [/dev/vgSAN/lvSAN2 (520.0MiB) write-thru deactivated]
  |   o- alua ................................................ [ALUA Groups: 1]
  |     o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
  o- fileio .............................................. [Storage Objects: 1]
  | o- file1 ................. [/root/diskFile1 (1.0GiB) write-back deactivated]
  |   o- alua ................................................ [ALUA Groups: 1]
  |     o- default_tg_pt_gp ..................... [ALUA state: Active/optimized]
  o- pscsi ............................................... [Storage Objects: 0]
  o- ramdisk ............................................. [Storage Objects: 0]
```

Now that the block devices are ready, we can go to the `/iscsi` environment and prepare the iSCSI targets. Initially, there will be no targets:

```
/backstores> cd /iscsi
/iscsi> ls
o- iscsi ......................................................... [Targets: 0]
```

2.2.3 Target Creation

Now, we create a target that provides access to the backing storage devices called *block1*, *block2* and *file1*. This can be done using the `create` command, followed by an IQN. IQNs are typically created using a naming format:

`iqn.<yearOfCreation>-<monthOfCreation>.<reverseDomainName>:<targetName>`

Thus, ours will be named: `iqn.2018-01.local.somuvmnet:target1`. This can be done with:

```
/iscsi> create iqn.2018-01.local.somuvmnet:target1
Created target iqn.2018-01.local.somuvmnet:target1.
Created TPG 1.
Global pref auto_add_default_portal=true
Created default portal listening on all IPs (0.0.0.0), port 3260.
```

Thus, both a target and a TPG are created at the same time. The target thus created can be viewed with:

```
/iscsi> ls
o- iscsi ....................................................... [Targets: 1]
  o- iqn.2018-01.local.somuvmnet:target1 ............................ [TPGs: 1]
    o- tpg1 .......................................... [no-gen-acls, no-auth]
      o- acls ..................................................... [ACLs: 0]
      o- luns ..................................................... [LUNs: 0]
      o- portals ............................................... [Portals: 1]
        o- 0.0.0.0:3260 ................................................ [OK]
```

2.2.4 TPG Configuration

Within the target is a TPG (Target portal group), which represents the entire configuration of the target. This includes all the ACLs, the LUNs and the portals related to the target.

ACLs

Next, we need to create the ACLs for our target. For this, we need to cd into the ACL environment of our target using (Note that tab-autocompletion works for this tool):

```
/iscsi> cd iqn.2018-01.local.somuvmnet:target1/tpg1/acls
/iscsi/iqn.20...et1/tpg1/acls>
```

We create the ACL node using:

```
/iscsi/iqn.20...et1/tpg1/acls> create iqn.2018-01.local.somuvmnet:vmdeux
Created Node ACL for iqn.2018-01.local.somuvmnet:vmdeux
```

Note that the identifier provided to create the node ACL is the IQN that has been set on the second server. The structure now looks like:

```
/iscsi/iqn.20...et1/tpg1/acls> cd /iscsi
/iscsi> ls
  o- iscsi ....................................................... [Targets: 1]
    o- iqn.2018-01.local.somuvmnet:target1 ............................ [TPGs: 1]
      o- tpg1 .......................................... [no-gen-acls, no-auth]
        o- acls ..................................................... [ACLs: 1]
        | o- iqn.2018-01.local.somuvmnet:vmdeux ................. [Mapped LUNs: 0]
        o- luns ..................................................... [LUNs: 0]
        o- portals ............................................... [Portals: 1]
          o- 0.0.0.0:3260 ................................................ [OK]
```

LUNs

Now, inside the *tpg1* node, we create a LUN by using:

```
/iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/block/block1
Created LUN 0.
Created LUN 0->0 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
```

Now, we can repeat the command a couple of times to create the LUNs for *block2* and *file1* as well:

```
/iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/block/block2
Created LUN 1.
Created LUN 1->1 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
/iscsi/iqn.20...:target1/tpg1> luns/ create /backstores/fileio/file1
Created LUN 2.
Created LUN 2->2 mapping in node ACL iqn.2018-01.local.somuvmnet:vmdeux
```

The contents of *tpg1* should now look like:

```
/iscsi/iqn.20...:target1/tpg1> ls
o- tpg1 ................................................. [no-gen-acls, no-auth]
  o- acls ............................................................ [ACLs: 1]
  | o- iqn.2018-01.local.somuvmnet:vmdeux ..................... [Mapped LUNs: 3]
  |   o- mapped_lun0 .................................. [lun0 block/block1 (rw)]
  |   o- mapped_lun1 .................................. [lun1 block/block2 (rw)]
  |   o- mapped_lun2 .................................. [lun2 fileio/file1 (rw)]
  o- luns ............................................................ [LUNs: 3]
  | o- lun0 ............. [block/block1 (/dev/vgSAN/lvSAN1) (default_tg_pt_gp)]
  | o- lun1 ............. [block/block2 (/dev/vgSAN/lvSAN2) (default_tg_pt_gp)]
  | o- lun2 ............... [fileio/file1 (/root/diskFile1) (default_tg_pt_gp)]
  o- portals ...................................................... [Portals: 1]
    o- 0.0.0.0:3260 ....................................................... [OK]
```

We can see that not only have the LUNs been created, but they've been assigned to the ACL as well! Thus, it becomes critical to create ACLs before the LUNs because the default behaviour of `targetcli` is to automatically assign any LUN that's been created to the ACLs in the TPG. Now, we have to create a portal.

Portals

We can create portal which will bear the IP address of the server on which our SAN will advertise the LUNs for this particular target. We do this by:

```
iscsi/iqn.20...:target1/tpg1> portals/ create 90.0.16.27
Using default IP port 3260
```

The complete configuration of the iSCSI setup can be viewed with:

```
/iscsi/iqn.20...:target1/tpg1> cd
/> ls
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block .............................................. [Storage Objects: 2]
  | | o- block1 ........... [/dev/vgSAN/lvSAN1 (500.0MiB) write-thru activated]
```

```
  | | | o- alua ............................................... [ALUA Groups: 1]
  | | |   o- default_tg_pt_gp ................... [ALUA state: Active/optimized]
  | | o- block2 ........... [/dev/vgSAN/lvSAN2 (520.0MiB) write-thru activated]
  | |   o- alua ............................................... [ALUA Groups: 1]
  | |     o- default_tg_pt_gp ................... [ALUA state: Active/optimized]
  | o- fileio ........................................... [Storage Objects: 1]
  | | o- file1 ................ [/root/diskFile1 (1.0GiB) write-back activated]
  | |   o- alua ............................................... [ALUA Groups: 1]
  | |     o- default_tg_pt_gp ................... [ALUA state: Active/optimized]
  | o- pscsi ............................................ [Storage Objects: 0]
  | o- ramdisk .......................................... [Storage Objects: 0]
  o- iscsi ...................................................... [Targets: 1]
  | o- iqn.2018-01.local.somuvmnet:target1 .......................... [TPGs: 1]
  |   o- tpg1 .......................................... [no-gen-acls, no-auth]
  |     o- acls ..................................................... [ACLs: 1]
  |     | o- iqn.2018-01.local.somuvmnet:vmdeux ............... [Mapped LUNs: 3]
  |     |   o- mapped_lun0 ........................... [lun0 block/block1 (rw)]
  |     |   o- mapped_lun1 ........................... [lun1 block/block2 (rw)]
  |     |   o- mapped_lun2 ........................... [lun2 fileio/file1 (rw)]
  |     o- luns ..................................................... [LUNs: 3]
  |     | o- lun0 ........ [block/block1 (/dev/vgSAN/lvSAN1) (default_tg_pt_gp)]
  |     | o- lun1 ........ [block/block2 (/dev/vgSAN/lvSAN2) (default_tg_pt_gp)]
  |     | o- lun2 .......... [fileio/file1 (/root/diskFile1) (default_tg_pt_gp)]
  |     o- portals ............................................... [Portals: 1]
  |       o- 0.0.0.0:3260 ................................................ [OK]
  o- loopback ................................................... [Targets: 0]
```

2.2.5 Adding a rule to the firewall

Now, we need to allow the TCP connections through port 3260 to use for SAN, using:

```
# firewall-cmd --add-port=3260/tcp --permanent
success
# firewall-cmd --reload
success
```

2.2.6 Starting target.service

Even though **targetcli** saves the present configuration to disk, a service called *target.service* must be enabled to ensure that the saved configuration is loaded each time after reboots. This is done with:

```
# systemctl start target
# systemctl enable target
Created symlink from /etc/systemd/system/multi-user.target.wants/target.service to
↪   /usr/lib/systemd/system/target.service.
# systemctl status target
● target.service - Restore LIO kernel target configuration
Loaded: loaded (/usr/lib/systemd/system/target.service; enabled; vendor preset: disabled)
Active: active (exited) since Tue 2018-01-02 16:28:20 IST; 25s ago
Main PID: 4291 (code=exited, status=0/SUCCESS)

Jan 02 16:28:19 vmprime.somuvmnet.local systemd[1]: Starting Restore LIO kern...
Jan 02 16:28:20 vmprime.somuvmnet.local systemd[1]: Started Restore LIO kerne...
```

This particular services instructs the kernel of its responsibilities as a SAN server and how the iSCSI targets are configured, so that it can accept incoming connections from iSCSI initiators and act accordingly.

2.3 Connecting the iSCSI Initiator to an iSCSI SAN

Now that the iSCSI SAN server is setup, we need an iSCSI initiator on a different (remote) server that can use the SAN. For this, the very first requirement is to obtain the software in the `iscsi-initiator-utils` package, which help in creating the initiator. We do this by using:

```
# yum -y install iscsi-initiator-utils
```

2.3.1 Setting up an initiator name

Next, we need to setup an **initiator name**, which must be the one we used in the ACL for the iSCSI *target*. To do this, we edit the `/etc/iscsi/initiatorname.iscsi` file. It's contents should be:

```
InitiatorName=iqn.2018-02.com.somuVMnet:vmPrime
```

The iSCSI configuration file is located at `/etc/iscsi/iscsid.conf` and this can be used to optimize the iSCSI configuration for the server.

2.3.2 iscsiadm Command

Now, we're going to set up the initiator using the `iscsiadm` command. The syntax of this command can be a bit cryptic, and thus it's recommended to use the man page's example section for a jump start on the syntax of the command.

The primary purpose of the **iscsiadm** command is to discover iSCSI targets and login to them, as well as access and manage the open-iscsi configuration database.

2.3.3 Discovery

Now, the initiator is ready for discovery. For this, we use the command:

```
# iscsiadm --mode discoverydb --type sendtargets --portal 10.0.99.12 --discover
10.0.99.12:3260,1 iqn.2018-02.local.somuvmnet:vmdeux
```

The `--mode discoverydb` option instructs the iscsiadm command to operate on the *discoverydb* section of the configuration database. It can also be abbreviated to `-m discoverydb`. The `--type sendtargets` (or `-t st`) tells the action to perform, i.e., send a list of targets. The `--portal 10.0.99.12` (`-p 10.0.99.12`) specifies the portal to be used for the action. Finally, the `--discovery` (`-D`) flag tells the command to perform discovery and add records if necessary. The output returned is a list of the targets on that particular portal. The most important piece of information here is the IQN of the relevant target.

2.3.4 Login

The login is performed on a particular IQN at a particular portal/node. This is achieved using:

```
# iscsiadm --mode node --targetname iqn.2018-02.local.somuVMnet:vmDeux --portal
    10.0.99.12 --login
Logging in to [iface: default, target: iqn.2018-02.local.somuvmnet:vmdeux, portal:
    10.0.99.12,3260] (multiple)
Login to [iface: default, target: iqn.2018-02.local.somuvmnet:vmdeux, portal:
    10.0.99.12,3260] successful.
```

Now the *mode* has been changed to **node** since we're dealing with a particular portal to login. The `--targetname iqn.2018-02.local.somuVMnet:vmDeux` option can be shortened to `-T iqn.2018-02.local.somuVMnet:vmDeux` and the portal can have a port specified with `-p 10.0.99.12:3260`.

Note that if the iqn of the initiator hasn't been set properly then login won't succeed with a failure due to authentication message. In that case, probably the IQN of the initiator hasn't been set properly. We need to edit the `/etc/iscsi/initiatorname.iscsi` file again and ensure it's identical to that in the ACL of the target. After the change, the **iscsid** service needs to be restarted for the new IQN to be used by the initiator: `systemctl restart iscsid`.

The presence of the new partitions as locally connected devices can be verified using:

```
# cat /proc/partitions
major minor  #blocks  name

8        0   10485760 sda
8        1    1048576 sda1
8        2    9436160 sda2
11        0    3963904 sr0
253        0    8384512 dm-0
253        1    1048576 dm-1
8       16     520192 sdb
8       32     520192 sdc
8       48      10240 sdd
[root@vmPrime ~]# lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda             8:0    0   10G  0 disk
├─sda1          8:1    0    1G  0 part /boot
└─sda2          8:2    0    9G  0 part
  ├─rhel-root 253:0    0    8G  0 lvm  /
  └─rhel-swap 253:1    0    1G  0 lvm  [SWAP]
sdb             8:16   0  508M  0 disk
sdc             8:32   0  508M  0 disk
sdd             8:48   0   10M  0 disk
```

The sdb, sdc and sdd devices are all LUNs on the iSCSI target. The `lsscsi` tool provides the iSCSI information for the target in even greater depth:

```
# lsscsi
[0:0:0:0]    disk    VMware,  VMware Virtual S 1.0   /dev/sda
[4:0:0:0]    cd/dvd  NECVMWar VMware SATA CD01 1.00  /dev/sr0
[40:0:0:0]   disk    LIO-ORG  block1           4.0   /dev/sdb
[40:0:0:1]   disk    LIO-ORG  block2           4.0   /dev/sdc
[40:0:0:2]   disk    LIO-ORG  file1            4.0   /dev/sdd
```

All the details of this connection to the IQN via the node is stored in the file `/var/lib/iscsi/nodes/iqn.2018-02.local.somuvmnet:vmdeux/10.0.99.12,3260,1/default`. Once logged in, after every reboot, the iSCSI initiator will automatically login to the SAN and present the LUNs of the target as locally mounted devices. To prevent that, we need to explicitly logout.

2.3.5 Logout

The logout operation needs the exact same parameters to be passed, other than `--login` which of course gets changed to `--logout`.

```
# iscsiadm -m node -T iqn.2018-02.local.somuvmnet:vmdeux -p 10.0.99.12 --logout
Logging out of session [sid: 7, target: iqn.2018-02.local.somuvmnet:vmdeux, portal: 10.0.99.12,3260]
Logout of [sid: 7, target: iqn.2018-02.local.somuvmnet:vmdeux, portal: 10.0.99.12,3260] successful.
```

2.3.6 Deleting node information

To delete all the information pertaining to an iSCSI target we use:

```
# iscsiadm -m node -T iqn.2018-02.local.somuvmnet:vmdeux -o delete
```

Another option would be to delete the folder with the IQN name of the target from `/var/lib/iscsi/nodes/`.

2.4 Verifying the iSCSI Connection

2.4.1 Verification on the iSCSI Initiator

To verify the iSCSI connection we use the **iscsiadm** command. The `-P` command is used to specify the print-level which means that the information is shown as a tree of varying levels of information (the higher the print level, more information is given).

To verify the iSCSI connection, we need information about the session, acquired using:

```
# iscsiadm -m session -P 1
Target: iqn.2018-02.local.somuvmnet:vmdeux (non-flash)
Current Portal: 10.0.99.12:3260,1
Persistent Portal: 10.0.99.12:3260,1
**********
Interface:
**********
Iface Name: default
Iface Transport: tcp
Iface Initiatorname: iqn.2018-02.local.somuvmnet:vmprime
Iface IPaddress: 10.0.99.11
Iface HWaddress: <empty>
Iface Netdev: <empty>
SID: 8
```

```
iSCSI Connection State: LOGGED IN
iSCSI Session State: LOGGED_IN
Internal iscsid Session State: NO CHANGE
# iscsiadm -m session -P 2
Target: iqn.2018-02.local.somuvmnet:vmdeux (non-flash)
Current Portal: 10.0.99.12:3260,1
Persistent Portal: 10.0.99.12:3260,1
**********
Interface:
**********
Iface Name: default
Iface Transport: tcp
Iface Initiatorname: iqn.2018-02.local.somuvmnet:vmprime
Iface IPaddress: 10.0.99.11
Iface HWaddress: <empty>
Iface Netdev: <empty>
SID: 8
iSCSI Connection State: LOGGED IN
iSCSI Session State: LOGGED_IN
Internal iscsid Session State: NO CHANGE
*********
Timeouts:
*********
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 15
*****
CHAP:
*****
username: <empty>
password: ********
username_in: <empty>
password_in: ********
************************
Negotiated iSCSI params:
************************
HeaderDigest: None
DataDigest: None
MaxRecvDataSegmentLength: 262144
MaxXmitDataSegmentLength: 262144
FirstBurstLength: 65536
MaxBurstLength: 262144
ImmediateData: Yes
InitialR2T: Yes
MaxOutstandingR2T: 1
# iscsiadm -m session -P 3
iSCSI Transport Class version 2.0-870
version 6.2.0.874-2
Target: iqn.2018-02.local.somuvmnet:vmdeux (non-flash)
Current Portal: 10.0.99.12:3260,1
Persistent Portal: 10.0.99.12:3260,1
**********
Interface:
**********
Iface Name: default
Iface Transport: tcp
Iface Initiatorname: iqn.2018-02.local.somuvmnet:vmprime
Iface IPaddress: 10.0.99.11
Iface HWaddress: <empty>
Iface Netdev: <empty>
```

```
SID: 8
iSCSI Connection State: LOGGED IN
iSCSI Session State: LOGGED_IN
Internal iscsid Session State: NO CHANGE
*********
Timeouts:
*********
Recovery Timeout: 120
Target Reset Timeout: 30
LUN Reset Timeout: 30
Abort Timeout: 15
*****
CHAP:
*****
username: <empty>
password: ********
username_in: <empty>
password_in: ********
************************
Negotiated iSCSI params:
************************
HeaderDigest: None
DataDigest: None
MaxRecvDataSegmentLength: 262144
MaxXmitDataSegmentLength: 262144
FirstBurstLength: 65536
MaxBurstLength: 262144
ImmediateData: Yes
InitialR2T: Yes
MaxOutstandingR2T: 1
************************
Attached SCSI devices:
************************
Host Number: 40       State: running
scsi40 Channel 00 Id 0 Lun: 0
Attached scsi disk sdb                State: running
scsi40 Channel 00 Id 0 Lun: 1
Attached scsi disk sdc                State: running
scsi40 Channel 00 Id 0 Lun: 2
Attached scsi disk sdd                State: running
```

2.4.2 Verification on the iSCSI Target

To verify the iSCSI config on the target, we need only check the contents of the targetcli command:

```
# targetcli
targetcli shell version 2.1.fb46
Copyright 2011-2013 by Datera, Inc and others.
For help on commands, type 'help'.

/> ls
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block .................................................. [Storage Objects: 2]
  | | o- block1 ........... [/dev/vgSAN/lvSAN1 (508.0MiB) write-thru activated]
  | | | o- alua ............................................... [ALUA Groups: 1]
  | | |   o- default_tg_pt_gp .................. [ALUA state: Active/optimized]
```

```
| | o- block2 ........... [/dev/vgSAN/lvSAN2 (508.0MiB) write-thru activated]
| |   o- alua .............................................. [ALUA Groups: 1]
| |     o- default_tg_pt_gp ................... [ALUA state: Active/optimized]
| o- fileio ............................................ [Storage Objects: 1]
| | o- file1 ............... [/root/diskFile1 (10.0MiB) write-back activated]
| |   o- alua .............................................. [ALUA Groups: 1]
| |     o- default_tg_pt_gp ................... [ALUA state: Active/optimized]
| o- pscsi ............................................. [Storage Objects: 0]
| o- ramdisk ........................................... [Storage Objects: 0]
o- iscsi ....................................................... [Targets: 1]
| o- iqn.2018-02.local.somuvmnet:vmdeux ............................ [TPGs: 1]
|   o- tpg1 .......................................... [no-gen-acls, no-auth]
|     o- acls ..................................................... [ACLs: 1]
|     | o- iqn.2018-02.local.somuvmnet:vmprime .............. [Mapped LUNs: 3]
|     |   o- mapped_lun0 ............................ [lun0 block/block1 (rw)]
|     |   o- mapped_lun1 ............................ [lun1 block/block2 (rw)]
|     |   o- mapped_lun2 ............................ [lun2 fileio/file1 (rw)]
|     o- luns ..................................................... [LUNs: 3]
|     | o- lun0 ........ [block/block1 (/dev/vgSAN/lvSAN1) (default_tg_pt_gp)]
|     | o- lun1 ........ [block/block2 (/dev/vgSAN/lvSAN2) (default_tg_pt_gp)]
|     | o- lun2 .......... [fileio/file1 (/root/diskFile1) (default_tg_pt_gp)]
|     o- portals ............................................... [Portals: 1]
|       o- 10.0.99.12:3260 ................................................ [OK]
o- loopback .................................................... [Targets: 0]
```

Chapter 3

System Performance Reporting

3.1 Understanding System Performance Parameters

The definition of performance of a system is dependent upon the expectations from a system. For example, **low latency** is desired from *database servers*, while **high throughput** is needed from *file servers*.

Actual performance has to be judged on the basis of performance level agreements. This has to be clearly defined for anyone - "*The web server should always react within 10 seconds*" is better than "*generic load should be less than 60%*", because that's what the end user will care about!

Thus, first we need to decide upon which metrics we want to measure, and then collect baseline data for it via monitoring systems.

3.1.1 Typical Performance Focus Areas

Factor	Description
Memory	The single most important factor that affects server performance. When enough memory isn't available, swap has to be used to house the excess pages and then the IO performance suffers, thus bogging down the entire system. It even affects the network throughput.
Disk	Another very important factor in overall server performance. When the disk is slow, too much memory is wasted to buffer data that's waiting to be written to disk. Processes will also have to wait longer to access data from the disk.
Network	Network is no longer a significant bottleneck, since most network connections aren't 10Mbps anymore - enterprise infrastructure uses Gigabit connections as a standard.
CPU	While the CPU has many tunables, in general it is not a very significant factor in performance deterioration. It is only for certain workloads that CPU becomes a factor in performance. The gain from CPU optimizations can be expressed in nanoseconds.

3.1.2 Common Performance Monitoring Tools

Terms	Description
top	While it's a very basic tool, it's also very rich in features. It provides an excellent generic overview of everything going on in the system. Typical use case for `top` is to detect problems and then use a more specialized tool to diagnose further.
iostat	A dedicated tool to detect Input/Output problems. It shows statistics about I/O. To detect which process is creating a high I/O load, a valuable tool is **iotop**.
vmstat	This tool shows statistics about virtual memory usage.
sar	The **System Activity Reporter** specializes in providing long-term data about what the system has been doing and long term performance statistics.

3.2 Understanding top

This is perhaps the single most important performance monitoring utility due to the kind of data it provides. There are alternatives to top such as `htop`, but top is programmed efficiently and doesn't have too much overhead. Comparing the two - htop uses about 5 times as much system resources as top!

The first feature of interest in the output of top is the **load average**, which consists of three numbers: the load average for the last 1, 5 and 15 minutes. The load average is the average of the number of processes in a runnable state, i.e., currently being executed by the CPU or waiting for CPU, over the concerned period of time. Optimally, all CPUs should be utilized as much as possible, but no process should be waiting for the CPU. The output of the `nproc` command tells us the effective number of CPUs available ($=$ Physical CPUs $\times$ logical cores per CPU).

The individual CPU utilization per CPU core can be shown by pressing the `1` key. A typical output is:

```
# top
%Cpu0  :  5.5 us,  3.3 sy,  0.0 ni, 90.7 id,  0.0 wa,  0.5 hi,  0.0 si,  0.0 st
```

Here, the number after the CPU indicates the core number. The *us* value refers to CPU usage in percentage in user space, i.e., by processes started by the end user without administrative privileges. The *sy* does the same, but for processes started by the users with root privileges. The *id* value is the percentage of time the processor remains idle. The next important metric is the number before *wa* which represents the waiting time, i.e., percentage of time processes spend waiting for I/O. A high value here indicates that the there's something wrong with the I/O channel and may indicate imminent disk failure.

Next, the memory statistics are shown, which includes the amount of memory completely free and amount of memory used to cache files that are frequently requested. Buffers contain data that needs to be written to disk during high I/O loads. While these are technically *non-essential*, it's suggested that 30% of the total memory be dedicated to buffers/cache usage.

We can also toggle the fields being shown by pressing the `f` key. If we quit top using the `q` key, the edits to the configuration are gone the moment we quit. However, if we quit using `Shift + W`, then the configuration is written to the `.toprc` file.

3.3 Understanding iostat

The iostat tool is a part of the `sysstat` package, which needs to be installed to use the `iostat` command. The command by itself provides a snapshot of the I/O statistics at the time of the invocation of the command. However, it takes two arguments in the syntax: `iostat <interval> <loops>`. The interval refers to the gap between displaying statistics and the loops refer to the number of times the command should show its output. Typical output for the command is:

```
# iostat 3 2
Linux 3.10.0-693.17.1.el7.x86_64 (vmPrime.somuVMnet.local)        Tuesday 27 February
 ↪  2018         _x86_64_        (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
0.50     0.00    0.64    0.49    0.00    98.37

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               1.20        54.56         3.33     584199      35622
scd0              0.00         0.10         0.00       1054          0
dm-0              1.11        51.31         3.13     549442      33537
dm-1              0.01         0.21         0.00       2228          0
sdb               0.00         0.10         0.00       1044          0
sdc               0.00         0.10         0.00       1044          0
sdd               0.00         0.03         0.00        336          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
5.44     0.00    1.36    0.00    0.00    93.20

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.68         0.00        10.88          0         32
scd0              0.00         0.00         0.00          0          0
dm-0              2.04         0.00        32.48          0         95
dm-1              0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
sdc               0.00         0.00         0.00          0          0
sdd               0.00         0.00         0.00          0          0
```

In the output, **tps** refers to the number of transactions per second. The *kB_read/s* and the *kB_wrtn/s* values are self explanatory. The next two columns show the total kBs read and written respectively.

3.3.1 Usage scenario

Let us consider a scenario where top shows us that processes spend 60% of their execution time waiting for I/O. Let us consider that the concerned server is connected to 6 different disks or other storage devices. We can use the output of the iostat command to determine which disk is so slow.

If we consult the output from the command, we can see that dm-0 has the greatest tps. To find out which device is dm-0, we can simply go to the `/dev/mapper` directory and see what links to it:

```
# \ls -l /dev/mapper
total 0
crw-------. 1 root root 10, 236 Feb 27 20:53 control
lrwxrwxrwx. 1 root root       7 Feb 27 20:53 rhel-root -> ../dm-0
lrwxrwxrwx. 1 root root       7 Feb 27 20:53 rhel-swap -> ../dm-1
```

3.3.2 iotop

The **iotop** command needs to be installed using `yum -y install iotop`. It shows the processes that are doing the most amount of I/O in descending order. Typical output looks like:

```
# iotop
Total DISK READ :      45.37 M/s | Total DISK WRITE :       0.00 B/s
Actual DISK READ:      45.37 M/s | Actual DISK WRITE:       0.00 B/s
TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
5696 be/4 root       45.37 M/s    0.00 B/s  0.00 % 73.97 % dd if=/dev/sda of=/dev/null
5450 be/4 root        0.00 B/s    0.00 B/s  0.00 % 12.66 % [kworker/0:2]
1 be/4 root        0.00 B/s    0.00 B/s  0.00 %  0.00 % systemd --switched-root --system
 ↪  --deserialize 21
...
```

Here we can see that the `dd if=/dev/sda of=/dev/null` is performing the most amount of I/O by copying the entire hard disk to `/dev/null`.

3.4 Understanding vmstat

3.4.1 Virtual Memory

Let us consider the typical output of top sorted on the basis of the Virtual Memory being used:

```
# top
 PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
1920 somu      20   0 1901016 215552  47856 S  0.7 11.6   0:47.79 gnome-shell
...
```

We can see that the `gnome-shell` is using 1901016 KiB of virtual memory, which is $\approx$ 1.82 GiB of virtual memory. Virtual Memory in Linux is memory that doesn't really exist. If we take a look at the `/proc/meminfo` file, we see:

```
VmallocTotal:   34359738367 kB
```

If we convert the VmallocTotal (Total amount of virtual memory that is possible for the kernel to allocate) to human readable units, it comes up to 32PB! That's not possible on most enterprise gear, let alone consumer hardware. Thus, the memory here doesn't really exist.

The key point here is that the kernel frequently needs to dish out unique memory address pointers to programs that demand it, but not actually assign any real memory till it's needed, i.e., the program tries to write to that location.

The kernel, instead of assigning real memory locations to programs, assigns memory in a virtual address space, which it then maps on to real memory on demand. The program itself remains blissfully oblivious to the knowledge of whether the memory it is referencing is virtual or real. All the trouble of fetching data on requirement and saving data falls on the kernel.

3.4.2 Resident Memory

A much more important concept is that of Resident Memory. Contrastingly to the Virtual Memory, the Resident memory is really used and is the total amount RAM being assigned to the process.

3.4.3 vmstat

The `vmstat` command when used without arguments shows various statistics pertaining to the resource consumption on the system:

```
# vmstat
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 3  0      0 269904   2116 860396    0    0   176    25  124  129  2  2 95  2  0
```

The significance of each is:

Terms	Description
proc	This part shows information about the processes: the **r** shows the number of running processes, **b** shows the number of blocking processes. A blocking process is a process that's waiting for something (e.g., I/O).
memory	This is the total amount of memory in swap, as well as real physical memory (RAM) used for buffers and cache.
swap	The two sub-categories are *swap-in*(`si`) and *swap-out*(`so`). If at any time we see that the system is utilizing swap memory, we can use `vmstat` to find out if the swap is being used actively, i.e., whether data is being written to or read from it actively.
io	The IO section deals with the number of blocks of I/O that's being performed - *blocks-in*(`bi`) and *blocks-out*(`bo`) provide a way to measure the real I/O activity at the moment, thus helping us discern if the server is spending a lot of time reading or writing during high I/O waits.
system	The metrics shown are *interrupts*(`in`) and *context switches*(`cs`). Interrupts are generally generated when a piece of hardware demands CPU attention. Context switches occur when the CPU switches the present task it's working on after being triggered by the scheduler. It is critical to the multi-tasking ability of a server since multiple processes need to coordinate and divide the CPU cycles. A high number of context-switches would indicate that the CPU isn't getting enough time per process.
cpu	These metrics refer to the percentage of CPU time spent executing programs in the *user-space*(`us`), *system space*(`sy`), *idle*(`id`) or *waiting* (`wa`).

Just like `iostat`, the `vmstat` provides an option to show the information at multiple points in time - the first argument is the time delay and the second the number of loops. To re-run vmstat every 2 seconds for 5 times we use:

```
# vmstat 2 5
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0      0 255824   2116 877492    0    0   135    20  118  122  1  1 96  2  0
 0  0      0 255576   2116 877492    0    0     0     0  174  178  6  3 91  0  0
 0  0      0 255576   2116 877492    0    0     0     0  152  145  6  2 92  0  0
```

```
 1  0      0 255384   2116 877492    0    0     0     0  179   161   8   2 90   0   0
 0  0      0 255452   2116 877492    0    0     0     0  256   406   9   8 83   0   0
```

Just like in `iostat` output, the first line has higher values of certain metrics because it gives a generic overview of the system operations where applicable. The next lines portray the activity within the delay time frame.

For detailed memory utilization statistics, we use `vmstat -s`:

```
# vmstat -s
1865964 K total memory
730672 K used memory
902516 K active memory
466968 K inactive memory
255684 K free memory
2116 K buffer memory
877492 K swap cache
1048572 K total swap
0 K used swap
1048572 K free swap
6813 non-nice user cpu ticks
1125 nice user cpu ticks
7128 system cpu ticks
543996 idle cpu ticks
8383 IO-wait cpu ticks
0 IRQ cpu ticks
180 softirq cpu ticks
0 stolen cpu ticks
734741 pages paged in
106795 pages paged out
0 pages swapped in
0 pages swapped out
667859 interrupts
685758 CPU context switches
1519798095 boot time
4208 forks
```

To change the display unit being used, we use the `-S (k/K/m/M)` option to change units. (where K=KiB and k=kB).

3.5 Understanding sar components

`sar` stands for the **System Activity Reporter**. It is a part of the *sysstat* package (like `iostat` and `vmstat`), and it collects data on an interval of 10 minutes by default. However, it can also be used to collect instantaneous data about the system as well.

What truly distinguishes `sar` from the other tools is the fact that it can be tasked to data collection for an extended period of time and the queried for information about a very specific period.

To make sorting and finding data in `sar` easier, it is recommended to set `LANG=C` before starting sar. Every Linux OS has an environment variable called **LANG** that affects the behavior of several utilities as well as setting the language. POSIX standard mandates that a locale called either POSIX or C be defined. Thus, it disables localization and makes the output predictable. Unless the option is set, timestamps are formatted in AM/PM which makes filtering said data harder. With the `LANG=C` option however, the timestamps use the military time format (24-hour format). A handy option is to generate an alias such as:

```
# echo "alias sar='LANG=C sar'" >> /etc/bashrc
```

sar data is collected via cron jobs in `/etc/cron.d/sysstat`. The collected data is written to `/var/log/sa`. The file `/etc/sysconfig/sysstat` has a `HISTORY` variable which dictates how long data should be stored. Typically, it's on a monthly basis.

3.5.1 /etc/cron.d/sysstat

This cronjob launches two different jobs are launched: **sa1** and **sa2**. The *sa1* job is for collecting short term statistics data while the *sa2* job executes once a day to collect data for long term statistics. Both of these write the results of their monitoring in a file in the `/var/log` directory.

3.5.2 /var/log/sa/sa[dd]

These are actually a bunch of files that start with the prefix `sa` and end with the date in `dd` format. Thus, typical file names are: *sa01, sa25, sa31,* etc. These files are unreadable by typical pagers like `less` and needs to be read by using the `sar` utility itself, by issuing commands like `sar -q` to get information about disk statistics, etc. One common mistake while accustoming to `sar` is to forget to start the sysstat services, since without them the data for the sar log files aren't populated and the utility has no data to work with.

3.6 Setting up sar

If the **sysstat** package isn't already installed, we first need to install it using `yum -y install sysstat`. Next, we ensure that the cron job for data collection via *sa1* and *sa2* were set up properly in `/etc/cron.d/sysstat` file, which should have the contents:

```
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
# Generate a daily summary of process accounting at 23:53
53 23 * * * root /usr/lib64/sa/sa2 -A
```

Thus, *sa1* is collecting data every 10 mins and *sa2* executes everyday at 11:53PM to collect long term data for the day. Now sar is ready to collect data, but if we were to query the sar already, we'd come up empty, since `sar` hasn't had the opportunity to log data yet!

Next, we check the config in `/etc/sysconfig/sysstat` file, which typically looks like:

```
# sysstat-10.1.5 configuration file.

# How long to keep log files (in days).
# If value is greater than 28, then log files are kept in
# multiple directories, one for each month.
HISTORY=28

# Compress (using gzip or bzip2) sa and sar files older than (in days):
COMPRESSAFTER=31

# Parameters for the system activity data collector (see sadc manual page)
```

```
# which are used for the generation of log files.
SADC_OPTIONS="-S DISK"

# Compression program to use.
ZIP="bzip2"
```

The primary feature of interest in this file is the value of the HISTORY variable which decides how long the collected data is stored.

Now, we have to wait for the `sadc` (*System Activity Data Collector*) utility to collect data for **sar** to analyse.

3.7 Analyzing sar data

Some of the most common options that print certain categories of the collected data are:

3.7.1 I/O operations

The `sar -b` command shows us the total transfers per second (`tps`), read tps(`rtps`), write tps(`wtps`), blocks read per second (`bread`)(1 block = 512B) and blocks written to per second (`bwrtn`). Typical output of the command looks like:

```
# sar -b
00:00:02          tps      rtps      wtps    bread/s    bwrtn/s
00:10:01         2.70      1.15      1.55      64.96     995.40
00:20:01         0.47      0.27      0.20      12.97       8.26
00:30:01         0.07      0.00      0.07       0.00       0.86
00:40:01         0.08      0.00      0.08       0.00       1.04
...
```

Thus, the use of military time makes the output a lot easier to process.

3.7.2 Processor information

It is possible to get the information about a single processor (i.e., a single logical core on a physical CPU, since linux considers each a separate processor) using the `sar -P <processorNumber>` command. Typical usage is (to find the usage of processor 0):

```
# sar -P 0
00:00:02        CPU     %user     %nice   %system   %iowait    %steal     %idle
00:10:01          0      5.21      0.00      3.09      0.59      0.00     91.11
00:20:01          0      0.12      0.00      0.24      0.15      0.00     99.48
...
04:10:01          0      0.29      0.00      0.36      0.10      0.00     99.25
Average:          0      1.04      0.00      1.03      0.31      0.00     97.62
```

3.7.3 Network Statistics

The `sar -n DEV` command shows the network statistics for each interface:

```
# sar -n DEV
Linux 3.10.0-693.17.1.el7.x86_64 (vmPrime.somuVMnet.local)        02/28/18        _x86_64_        (1 CPU)

00:00:02        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
00:10:01           lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:10:01   virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:10:01       virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:10:01        ens33     63.83     19.03     90.10      1.19      0.00      0.00      0.00
...
16:40:01        ens33      1.28      1.13      0.15      0.36      0.00      0.00      0.00

Average:        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s
Average:           lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:   virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:       virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00
Average:        ens33      1.19      0.92      0.83      0.12      0.00      0.00      0.00
```

To view the statistics for just one interface, we can use `sar -n DEV` \ grep <interface-Name>|:

```
# sar -n DEV | grep ens33
00:10:01        ens33     63.83     19.03     90.10      1.19      0.00      0.00      0.00
...
16:40:01        ens33      1.28      1.13      0.15      0.36      0.00      0.00      0.00
Average:        ens33      1.19      0.92      0.83      0.12      0.00      0.00      0.00
```

Chapter 4

System Optimization Basics

4.1 Understanding /proc contents

To optimize a Linux system, we should know in depth about the `/proc` file system. This file system provides an interface to the kernel using which we can take a look at the present state of the system as well as optimize it as per our requirements.

The `/proc` file system is filled with several **status files** which tell us about many operational system parameters. In fact, may performance monitoring utilities get their data from the `/proc` file system itself!

In older versions of Linux, there were several hard-coded parameters that needed to be changed to optimize the system, and thus required a recompilation of the kernel. Modern Linux kernels however, get many of these parameters from the `/proc/sys` file system and thus offer optimization in a flexible manner!

The `/proc/sys` file system offers different interfaces that are related to different aspects of the file system which can be tuned through their corresponding interface.

4.2 Analysing the /proc filesystem

4.2.1 Process Directories

The `/proc` file system has a process directory named with a number corresponding to the *PID* of every running process on the system:

```
# ls /proc
1     17    2064  2514  333  362  465  7    93           locks
10    1759  2070  2530  334  363  466  732  acpi         mdstat
1180  1764  2093  26    335  364  479  734  asound       meminfo
1186  18    2094  2611  336  365  480  736  buddyinfo    misc
1187  1856  2109  2616  337  366  481  757  bus          modules
1191  1875  2116  2664  338  367  482  758  cgroups      mounts
1195  1880  212   2665  339  368  483  760  cmdline      mpt
1196  1884  2128  27    340  369  484  762  consoles     mtrr
1199  19    2147  273   341  370  485  766  cpuinfo      net
12    1903  2150  274   342  371  486  767  crypto       pagetypeinfo
1297  1907  2151  275   343  372  487  768  devices      partitions
```

```
1298  1921  2164  278   344  373  488  772  diskstats    sched_debug
1299  1933  2167  279   345  374  489  773  dma          schedstat
13    1938  2175  28    346  375  5    779  driver       scsi
1331  1940  2180  284   347  376  560  791  execdomains  self
1336  1948  2181  286   348  377  561  793  fb           slabinfo
1337  1953  2186  287   349  378  587  794  filesystems  softirqs
1338  1954  2188  288   350  379  589  795  fs           stat
1344  1961  2208  295   351  38   590  8    interrupts   swaps
1383  1962  2243  3     352  380  60   800  iomem        sys
14    1970  2250  323   353  381  601  801  ioports      sysrq-trigger
15    1973  2321  324   354  382  616  802  irq          sysvipc
1569  1979  2322  325   355  39   638  803  kallsyms     timer_list
1584  1983  2358  326   356  4    640  818  kcore        timer_stats
1586  2     2426  327   357  40   641  827  keys         tty
16    2001  2454  328   358  406  642  831  key-users    uptime
1640  2043  2460  329   359  407  645  835  kmsg         version
1643  2048  2462  330   36   41   646  852  kpagecount   vmallocinfo
1686  2053  2481  331   360  453  647  869  kpageflags   vmstat
1691  2059  25    332   361  454  648  9    loadavg      zoneinfo
```

Inside each of these process directories, there are several files that tell us about the present status of the process, and provide other necessary details about it. For example, inside the directory for the process with *PID 881*, we find:

```
# ls /proc/561/
attr             cpuset   limits      net            projid_map  stat
autogroup        cwd      loginuid    ns             root        statm
auxv             environ  map_files   numa_maps      sched       status
cgroup           exe      maps        oom_adj        schedstat   syscall
clear_refs       fd       mem         oom_score      sessionid   task
cmdline          fdinfo   mountinfo   oom_score_adj  setgroups   timers
comm             gid_map  mounts      pagemap        smaps       uid_map
coredump_filter  io       mountstats  personality    stack       wchan
```

The **cmdline** file in this directory tells us about the command that is being run in the process. For example, the command that spawned the process with PID 561 is:

```
# cat /proc/561/cmdline
/usr/lib/systemd/systemd-journald
```

4.2.2 Status files

The `/proc` directory also houses several files that tell us about the different aspects of what our Operating System is doing and how it's performing. For example, the `/proc/partitions` file contains a list of all the storage devices that our system can access, as well as all the partitions that are housed on those devices:

```
# cat /proc/partitions
major  minor    #blocks name
8         0   10485760 sda
8         1    1048576 sda1
8         2    9436160 sda2
11         0    1048575 sr0
253         0    8384512 dm-0
253         1    1048576 dm-1
```

Similarly, the /proc/cpuinfo file has information about the CPU configuration, and the /proc/meminfo has information about the memory configuration. A list of all the file systems supported by the currently running operating systems (that have a kernel module presently loaded) can be found in /proc/filesystems:

```
# cat /proc/filesystems
nodev        sysfs
nodev        rootfs
nodev        ramfs
nodev        bdev
nodev        proc
nodev        cgroup
nodev        cpuset
nodev        tmpfs
nodev        devtmpfs
nodev        debugfs
nodev        securityfs
nodev        sockfs
nodev        pipefs
nodev        anon_inodefs
nodev        configfs
nodev        devpts
nodev        hugetlbfs
nodev        autofs
nodev        pstore
nodev        mqueue
nodev        selinuxfs
             xfs
nodev        rpc_pipefs
nodev        nfsd
             fuseblk
nodev        fuse
nodev        fusectl
```

Thus, if we insert the module for vFat or ext4 into the kernel, it'll show up in the /proc/filesystems listing:

```
# modprobe vfat
[root@vmPrime proc]# modprobe ext4
[root@vmPrime proc]# cat /proc/filesystems
nodev        sysfs
nodev        rootfs
...
nodev        selinuxfs
             xfs
nodev        nfsd
             fuseblk
nodev        fusectl
             vfat
             ext3
             ext2
             ext4
```

4.2.3 /proc/sys

The /proc/sys file system is our interface to optimization of the system. It has folders related to the different aspects of the kernel's functions:

```
# ls /proc/sys
abi  crypto  debug  dev  fs  kernel  net  sunrpc  user  vm
```

Among these, the most important ones are **fs** for *file systems*, **kernel** for *kernel optimizations*, **net** for *networking* and **vm** for the *virtual memory*.

For example, the `/proc/sys/vm` directory contains several files that help tune the kernel for memory optimization:

```
# ls /proc/sys/vm
admin_reserve_kbytes        lowmem_reserve_ratio        oom_dump_tasks
block_dump                  max_map_count               oom_kill_allocating_task
compact_memory              memory_failure_early_kill   overcommit_kbytes
dirty_background_bytes      memory_failure_recovery     overcommit_memory
dirty_background_ratio      min_free_kbytes             overcommit_ratio
dirty_bytes                 min_slab_ratio              page-cluster
dirty_expire_centisecs      min_unmapped_ratio          panic_on_oom
dirty_ratio                 mmap_min_addr               percpu_pagelist_fraction
dirty_writeback_centisecs   mmap_rnd_bits               stat_interval
drop_caches                 mmap_rnd_compat_bits        swappiness
extfrag_threshold           nr_hugepages                user_reserve_kbytes
hugepages_treat_as_movable  nr_hugepages_mempolicy      vfs_cache_pressure
hugetlb_shm_group           nr_overcommit_hugepages     zone_reclaim_mode
laptop_mode                 nr_pdflush_threads
legacy_va_layout            numa_zonelist_order
# cat swappiness
30
```

One such file is the `swappiness` file - which defines the willingness of a server to write data to the swap. The default value of *30* means it isn't very willing. If we were to increase the value set in the file, we would make it more likely for the kernel to use swap in cases where it normally wouldn't have.

4.3 Optimizing through /proc

Optimization through the `/proc/sys` file system is an easy affair that needs us to only change the value contained in certain files to tune the associated aspect. Let us consider the `/proc/sys/net/ipv4` directory, which deals with IPv4 networking on the system:

```
# cd /proc/sys/net/ipv4
# ls
# ls ip_*
ip_default_ttl  ip_forward           ip_local_reserved_ports
ip_dynaddr      ip_forward_use_pmtu  ip_nonlocal_bind
ip_early_demux  ip_local_port_range  ip_no_pmtu_disc
# cat ip_forward
0
```

The value of `0` in the *ip_forward* file shows that this system doesn't do IP forwarding, i.e., it doesn't forward specific IP packets for other IPs, i.e., it doesn't act as a router. To change this, we use:

```
# echo 1 > ip_forward
# cat ip_forward
1
```

When we echo values into the proc file system, it is only temporary and is wiped with every reboot - thus giving us an opportunity to test our settings directly before making anything permanent. If we like the results, we can make it permanent - otherwise reboot to restore the original values.

4.3.1 Sysctl

The utility `sysctl` can be used to display as well as control all the tunables that exist for a system:

```
# sysctl -a
abi.vsyscall32 = 1
crypto.fips_enabled = 0
debug.exception-trace = 1
...
vm.swappiness = 30
vm.user_reserve_kbytes = 55126
vm.vfs_cache_pressure = 100
vm.zone_reclaim_mode = 0
```

Thus, there is no need to manually traverse the file system hierarchy in the `proc` file system. We can directly set the values using `sysctl` command. Further, it can help us find every single tunable that is related to a keyword very easily by piping the output to *grep*. For example, if we want to tune ICMP (the protocol behind Ping and other control messages that can be sent over IP packets) , we can use:

```
# sysctl -a | grep icmp
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
net.ipv4.icmp_echo_ignore_all = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_errors_use_inbound_ifaddr = 0
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.icmp_msgs_burst = 50
net.ipv4.icmp_msgs_per_sec = 1000
net.ipv4.icmp_ratelimit = 1000
net.ipv4.icmp_ratemask = 6168
sysctl: reading key "net.ipv6.conf.ens33.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
sysctl: reading key "net.ipv6.conf.virbr0.stable_secret"
sysctl: reading key "net.ipv6.conf.virbr0-nic.stable_secret"
net.ipv6.icmp.ratelimit = 1000
net.netfilter.nf_conntrack_icmp_timeout = 30
net.netfilter.nf_conntrack_icmpv6_timeout = 30
```

Thus, `sysctl -a` and `grep` together make it extremely easy to discover the tunable we need to optimize any facet pertaining to our current needs.

4.4 Introducing sysctl

The purpose of the **sysctl** utility is that it can act as an interface to control the entirity of the `/proc/sys` file system, including making **permanent** changes to the tunables for system optimization. Thus, it's an invaluable tool to handle kernel runtime parameters.

We can put our parameters in `/etc/sysctl.conf` or the related directory `/etc/sysctl.d`. All of these parameters are read by **sysctl** and then passed on to the `/proc/sys` file system (also known as **procfs**). It is encouraged to thoroughly test out the changes by first echoing the parameter values to the concerned file in `/proc/sys` first so that any errors can be weeded out and then making the changes permanent by creating a `.conf` file in the `/etc/sysctl.d` directory.

4.5 Using sysctl

Any user changes to kernel runtime parameters should be put inside the `/etc/sysctl.d` directory. Anything put in this directory overwrites the vendor presets in the `/usr/lib/sysctl.d` directory, and of course, the default kernel parameter values. One such file containing vendor preset kernel parameters is the `/usr/lib/sysctl.d/00-system.conf`:

```
# Kernel sysctl configuration file
#
# For binary values, 0 is disabled, 1 is enabled.  See sysctl(8) and
# sysctl.conf(5) for more details.

# Disable netfilter on bridges.
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

The files in the `/usr/lib/` directory in general shouldn't be touched since they're mostly vendor presets and thus get overwritten with every update. The only way to have our settings overwrite theirs is via configuring the settings in the concerned directory in the `/etc` directory.

The kernel parameters controllable through the sysctl utility are all named according to the convention : if the file is `/proc/sys/net/ipv4/ip_local_port_range`, then its equivalent sysctl setting will be `net.ipv4.ip_local_port_range`, i.e., each directory in the path name of the file after `/proc/sys` seperated by a '.', terminating with the file name. This can be demonstrated with:

```
# cat /proc/sys/net/ipv4/ip_local_port_range
32768        60999
# sysctl -a | grep net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 32768        60999
```

Thus, to change the swappiness permanently, we write the setting in `/etc/sysctl.conf` (or a new configuration file in `/etc/sysctl.d`) and then reboot to see the changes:

```
vm.swappiness = 60
```

4.5.1 sysctl -w

To directly load a sysctl setting immediately without reading it from the `/proc/sys` file system, we use: `sysctl -w <parameter=value>`.

4.5.2 sysctl -p

To load an entire configuration file and set all of the kernel parameters mentioned in it, we use `sysctl -p <configFile.conf>`.

4.6 Modifying Network Behaviour through /proc and sysctl

We can see our IP address(es) and ping it via:

```
# ip a
1: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
link/ether 00:0c:29:3b:b9:1c brd ff:ff:ff:ff:ff:ff
inet 10.0.99.11/24 brd 10.0.99.255 scope global ens33
valid_lft forever preferred_lft forever
inet6 fe80::f408:1ebf:7742:9fd8/64 scope link
valid_lft forever preferred_lft forever
# ping -c 2 10.0.99.11
PING 10.0.99.11 (10.0.99.11) 56(84) bytes of data.
64 bytes from 10.0.99.11: icmp_seq=1 ttl=64 time=0.420 ms
64 bytes from 10.0.99.11: icmp_seq=2 ttl=64 time=0.135 ms

--- 10.0.99.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.135/0.277/0.420/0.143 ms
```

Now, ping is an ICMP control message, and ICMP control messages can be blocked by changing an associated kernel parameter. To find the parameter responsible for ignoring ICMP echo requests that ping needs, we use:

```
# sysctl -a | grep icmp
net.ipv4.icmp_echo_ignore_all = 0
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_errors_use_inbound_ifaddr = 0
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.icmp_msgs_burst = 50
net.ipv4.icmp_msgs_per_sec = 1000
net.ipv4.icmp_ratelimit = 1000
net.ipv4.icmp_ratemask = 6168
net.ipv6.icmp.ratelimit = 1000
net.netfilter.nf_conntrack_icmp_timeout = 30
net.netfilter.nf_conntrack_icmpv6_timeout = 30
```

The very first displayed parameter is exactly what we need. So, we first test it with:

```
# echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
[root@vmPrime ~]# ping 10.0.99.11
PING 10.0.99.11 (10.0.99.11) 56(84) bytes of data.
^C
--- 10.0.99.11 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3000ms
```

Now that our host has stopped replying to pings, we can make the changes permanent by adding to the `/etc/sysctl.conf` file (and then rebooting):

```
net.ipv4.icmp_echo_ignore_all = 1
```

Chapter 5

Configuring Logging

5.1 Understandig Logging In RHEL 7

Due to the introduction of Systemd several services of the older Unix System V now have a counter part in their systemd equivalent. Such is the case for rsyslogd and journald. The former is responsible for logging in System V systems while journald does it in systemd systems. However, due to the concern of backwards compatibility (i.e., being able to use tools written for older versions of Linux which may have used System V utilities), modern distros like RHEL 7 also have rsyslogd installed.

There can be thus, three different ways of logging application information in RHEL 7:

- Directly write to a log file somewhere - no standardized way of accessing logs.
- Write to Systemd's *Journald* - logs are accessible via **journalctl**
- Write to *rsyslogd* - logs are accessible via `/var/log`.

An important point to note here is that **rsyslog** is still the central logging authority, but journald simply adds features to the way that logging is organized. Thus, journald doesn't really replace rsyslog.

This however means that there's scope for confusion on part of the user (or admin) who's handling the system - to understand exactly where a certain programs might write it's logs to. Thus, we can connect the two together to show the same information.

5.2 Connecting Journald to Rsyslogd

We merely need to add a few lines of configuration to have both services report their own logs to each other:

```
# To connect Journald to rsyslogd:
## In /etc/rsyslog.conf:
$ModLoad imuxsock
$OmitLocalLogging off
## In /etc/rsyslog.d/listend.conf:
$SystemLogSocketName /run/systemd/journal/syslog

```

```
# To connect rsyslogd to journald:
## In /etc/rsyslog.conf:
$ModLoad omjournal
*.* :omjournal:
```

NOTE - *this is legacy syntax and will probably not work anymore. Plus, it causes errors in journalctl. Further investigation is needed.*

rsyslog messages are sent to **journald**, and vice versa. However, sending to journald is disabled by default in rsyslog.conf. To fix this we add the load the module omjournal (*output module journal*) using `$ModLoad omjournal`. Next, we use rsyslog's notation for indicating the facility, priority and destination. The statement `*.* :omjournal:` means for any facility and any priority, we want the log to be forwarded to *omjournal*.

Receiving from journal is enabled by default in `rsyslog.conf`. This is done via: `$ModLoad imuxsock` (Input Module UniX SOCKet), which instructs rsyslog to listen to a socket. Now, local logging has to be enabled using the `$OmitLocalLogging off` option. Finally, the socket name on which rsyslog will listen will have to be specified in the `/etc/rsyslog.d/listend.conf` file, and has to be set to the value `/run/systemd/journal/syslog`. Everything on the systemd end is already configured and needs no manual intervention. This completes the integration of the two.

5.2.1 Modules

Thus, the connection between rsyslog and journald is facilitated by modules. There are several types of modules, which can be identified and classified by:

Prefix	Type	Description
im*	Input Module	Source of information for the rsyslog journal; Loaded in `/etc/rsyslog.conf` and socket name specified in `/etc/rsyslog.d/listend.conf`.
om*	Output Module	Destination to which data from rsyslog will be sent; Configured in `/etc/rsyslog.conf`
		Other modules such as parser modules, message modification modules, etc.

Together these modules lets us manipulate the log messages any way we want.

5.2.2 Importing text files to log : httpd error log

To import the HTTPD error log to rsyslog, the following needs to be added to the file `/etc/rsyslog.conf`:

```
$ModLoad imfile
$InputFileName /var/log/httpd/error_log
$InputFileTag apache-error:
$InputFileState state-apache-error
$InputRunFileMontitor
```

This takes the error log read and maintained by apache and inserts the data into rsyslog. The `$InputRunFileMonitor` enables monitoring of the specified file.

5.2.3 Exporting data to an output module : exporting to a database

Since rsyslog writes the data to a simple text file, and for managing log information the ability to query is very important, we may choose to export the data to a database. Assuming we're using a MySQL database:

```
$ModLoad ommysql
$ActionOmmysqlServerPort 1234
*.* :ommysql:database-serverName,database-name,database-userid,database-password
```

The first line loads the MySQL Output module for rsyslog. Then, we define the server port on which the logs will be forwarded. Then, finally, we configure the output module using the *facility, priority :destination* method where we send everything (all facilities and every priority[*.*]) to the output module, while also providing the database details.

5.3 Setting up Remote Logging

It is via the use of modules that remote logging can be configured. This can be done using two types of protocols: TCP and UDP. UDP is the classical solution and is backwards compatible, but due to the very nature of the protocol, the message transfer isn't connection oriented. What this means is that messages may get lost in transit, and thus data loss may occur. Contrastingly, TCP only does data exchange after a connection has been established, and thus provides more reliable logging. So, if everything in our syslog configuration is compatible with TCP, then we should definitely opt for TCP syslog reception. Example usage for both are provided in a commented section in the `/etc/rsyslog.conf` file:

```
# Provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514

# Provides TCP syslog reception
#$ModLoad imtcp
#$InputTCPServerRun 514
```

The TCP syslog reception needs to be done exactly as the example states and so uncommenting the lines is all we need to do. The **imtcp** module enables the reception of log information via TCP connection and the `InputTCPServerRun` option chooses port 514 to use as incoming port for the messages (the IP is the IP of the server itself).

Now, for the other servers, the configuration has to be such that they send their own logging data to the same IP and port as we just configured. If our logging server is running on 10.0.50.11:514, then the *forwarding rule* configuration (an example of which is present in the `rsyslog.conf` file itself) becomes:

```
# ### begin forwarding rule ###
# The statement between the begin ... end define a SINGLE forwarding
# rule. They belong together, do NOT split them. If you create multiple
# forwarding rules, duplicate the whole block!
# Remote Logging (we use TCP for reliable delivery)
#
# An on-disk queue is created for this action. If the remote host is
# down, messages are spooled to disk and sent when it is up again.
#$ActionQueueFileName fwdRule1 # unique name prefix for spool files
#$ActionQueueMaxDiskSpace 1g   # 1gb space limit (use as much as possible)
```

```
#$ActionQueueSaveOnShutdown on # save messages to disk on shutdown
#$ActionQueueType LinkedList   # run asynchronously
#$ActionResumeRetryCount -1    # infinite retries if host is down
# remote host is: name/ip:port, e.g. 192.168.0.1:514, port optional
*.* @@10.0.50.11:514
# ### end of the forwarding rule ###
```

Note that the `@@` sign in the line `*.* @@10.0.50.11:514` statement signifies the use of TCP. If UDP is to be used instead, we replace it with a single `@` sign, thus making the statement: `*.* @10.0.50.11:514`. The forwarding rule asks rsyslogd to forward logs from any facility of any priority should be sent to the server over at 10.0.50.11 on port 514 via TCP. Now, after rsyslog service has been restarted on both the server and the client, remote logging should work.

An additional thing to remember is that on the logging server, i.e., the server accepting the logs, the port 514 needs to be unblocked for TCP traffic using:

```
# firewall-cmd --add-port=514/tcp --permanent
# firewall-cmd --reload
```

If SELinux blocks TCP traffic or the port isn't the standard port for remote logging, i.e., port 514, then the associated port needs to be given the right security context of `syslogd_port_t`. For the TCP port 514, the command to allow logging on the server is:

```
# semanage port -a -t syslogd_port_t -p tcp 514
```

Part II

Networking and Apache

Chapter 6

Configuring Advanced Networking

6.1 Networking Basics Resumed

6.1.1 Network configuration tools

Terms	Description
ip addr show	Shows address information about all network interfaces.
ip -s link show ens33	Shows statistics about packets but for interface `ens33`. Same as `ip -s link`, but for a specific interface.
ip route	Shows routing information
traceroute / **tracepath**	For analysing a particular route or path.
netstat / **ss**	Analyse ports and services currently listening for incoming connections.

6.1.2 Network Manager

NetworkManager is used to both manage and monitor network settings. While the settings made with the IP tool act directly on the NICs, they're temporary and wiped with every boot or even bringing the interface down and up again. The network manager uses config scripts in `/etc/sysconfig/network-scripts` to store our configs and use them after every boot. The settings can be managed using either `nmcli` or `nmtui`. The former is preferred for scripts while `nmtui` is preferred for manual configs.

nmcli concepts

- A **device** or an **interface** is a network interface, corresponding to the hardware NIC (Network Interface Card).
- A **connection** is a collection of configuration settings for a *device*.
- Multiple connections can exist for the same device, but since they operate on the same settings for the device, only one of them can be active.

- All the connections (and some details) can be shown with the command `nmcli con show`.
- To show all the details for a particular connection, we have to use the command `nmcli con show <interface name>` like `nmcli con show wlo1` (where *wlo1* is the name of the connection).
- To see the connection status for a device, we use `nmcli dev status`. This shows us which devices are connected and which connection they're presently using.
- To see the details of the actual NIC device, we use `nmcli dev show <deviceName>`.

6.1.3 Creating Network Interfaces with nmcli

To add a new connection using `nmcli` that has the name *dhcp* that auto-connects using dynamic IP on interface *eno1*, we use:

```
# nmcli con add con-name "dhcp" type ethernet ifname eno1
```

To add a new connection *static* that uses a static ip that doesn't connect automatically, we use:

```
# nmcli con add con-name "static" type ethernet ifname eno1 autoconnect no ip4
↪   192.168.122.102 gw4 192.168.122.1
```

Now, the available connections can be checked with `nmcli dev status`. The we can connect the *static* connection using `nmcli con up static` and then switch back to the original connection *dhcp* using `nmcli con up dhcp`.

6.1.4 Modifying Network Interfaces using nmcli

To see the details of the *static* connection, we use `nmcli con show static`. Then, to add/modify the DNS server address for that connection, we use the `con mod` keywords, which makes the command:

```
# nmcli con mod "static" ipv4.dns 192.168.122.1
```

Note that the modification requires the `ipv4` keyword instead `ip4`. To define a second IPv4 DNS for the *static* connection, we use the + symbol to denote that a new value for the item should be added and the old value shouldn't be overwritten. The command then becomes:

```
# nmcli con mod "static" +ipv4.dns 8.8.8.8
```

An existing static IP address and gateway can be edited using:

```
# nmcli con mod "static" ipv4.addresses "192.168.100.10/24 192.168.100.1"
```

A secondary IPv4 address can be added using:

```
# nmcli con mod "static" +ipv4.addresses "10.0.0.10/24"
```

Finally, to activate all the above settings, we use: `nmcli con up static`.

6.1.5 Working directly with Configuration Files

All the `nmcli` tool really does while adding or modifying settings is write the changes to the configuration files in `/etc/sysconfig/network-scripts/ifcfg-<interfaceName>`. We may choose to edit them directly if needed. Then, after making the necessary modifications, we ask the NetworkManager service to reload the configuration using `nmcli con reload`.

6.1.6 Managing Hostname and DNS

The hostname is stored in the file `/etc/hostname` and can be edited directly or using the `hostnamectl set-hostname <newHostName>` command. The current hostname can then be viewed using `hostnamectl status`.

The value of the search domain and preferred nameserver (i.e., the one that the NetworkManager uses by default) is auto-pushed from `/etc/sysconfif/network-scripts/ifcfg-<connectionName>` to the file `/etc/resolv.conf`.

6.2 Understanding Routing

Let us consider the following network:

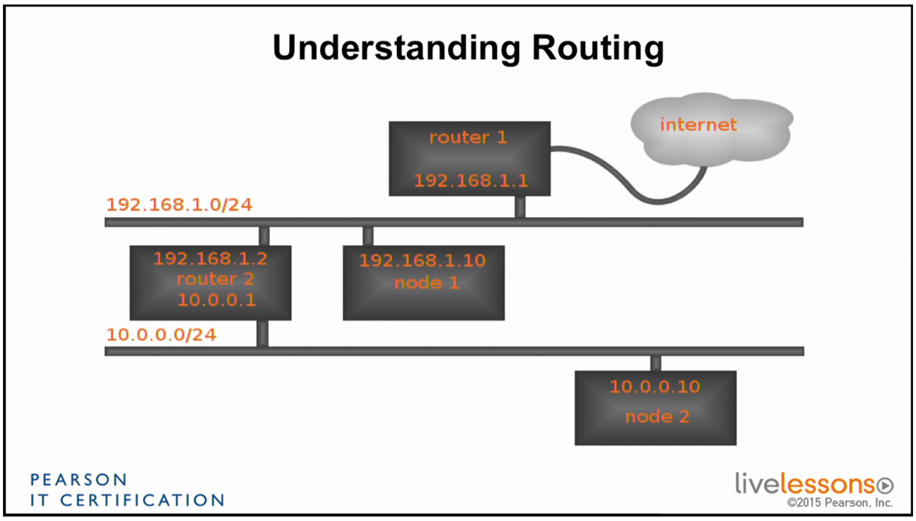

Figure 6.1: Sample Network

Here, we see two different networks - the `10.0.0.0/24` network connected to the inner `192.168.1.0/24` network via *router 2* (`10.0.0.1`), which in turn connects to the internet via the edge router with IP `192.168.1.1` - *router 1*.

For any packet headed to the internet on network 2, i.e., any packet originating from *node 2*, the default gateway will have to be *router 2* (`10.0.0.1`). This gets the packet on to the `192.168.1.0/24` network, where the default gateway is *router 1* (`192.168.1.1`), which passes it on to the internet.

However, when the packets originate from node 1 (`192.168.1.10`), there are two possible routes - if the packet is destined for the `10.0.0.0/24` network, then the gateway should be *router 2* (`192.168.1.2`). But if the packet is for any other network, then the default gateway of *router 1* (192.168.1.1) should be used. Thus, a static route should be defined on node 1 for the 10.0.0.0/24 network.

6.3 Setting up Static Routing

The most convenient way to set up static routes is to use `nmtui`. Let's assume we're setting up static routing for node 2 in our last example.

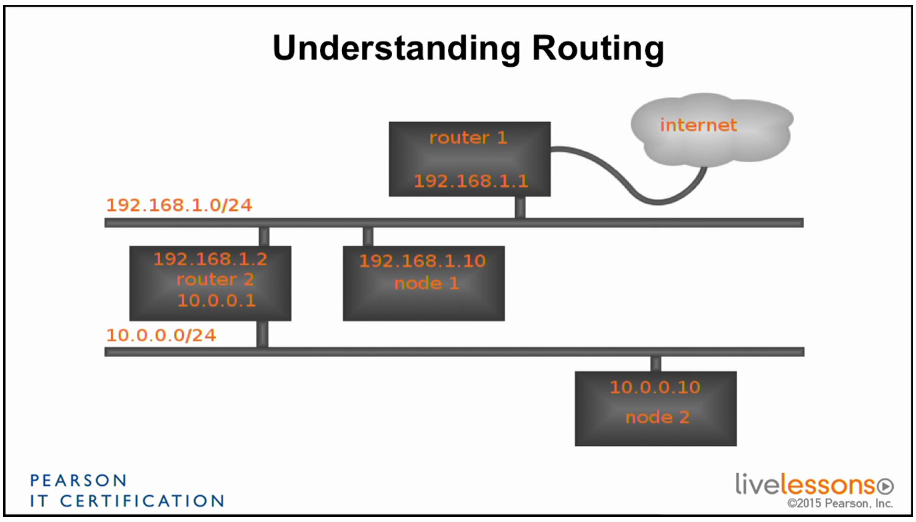

Figure 6.2: Network Diagram

We need to edit the existing connection to include the new static route. For this, we select the options: `Edit a Connection` → Select the connection to use → `Edit...` → `Routing` section → `Edit...` → `Add...` → Type the address of the network for which the static route will be defined in `Destination/Prefix` (with the Network ID and prefix, like, `10.0.0.0/24`) → Add the IP address of the router that leads to the network in the `Next Hop` section (`192.168.1.2` in our case).

The **metric** of the connection is how a router chooses which route to take when there are multiple routes available to another network. Thus, it's only useful when there are multiple routes available for the same network, and is irrelevant to us right now. We now choose `<Ok>` → `<Ok>` → `<Quit>`.

Note however, that the new route won't be added to the network configuration till either the connection is *refreshed* (by reactivating the connection) or the NetworkManager service is restarted. We could do this by `nmtui` → `Activate a Connection` → Select the connection which we edited → `Activate`. Now the output of `ip route show` will show the static route as well.

If the interface name was *ens33*, The `/etc/sysconfig/network-scripts` directory now has a new file called : `route-ens33` with the following contents:

```
ADDRESS0=10.0.0.0
NETMASK0=255.255.255.0
GATEWAY0=192.168.1.2
```

Note that the nmtui utility has translated the /24 prefix from the **CIDR** (Classless Inter-Domain Routing) notation `10.0.0.0/24` to the standard Network IP and Network Masks, where `/24` translates to the network mask of `255.255.255.0`.

6.4 Understanding Network Bridges

A network bridge is a device that connects two or more networks to form one extended network. For example, an Ethernet bridge connects two or more LANs to create a unified, extended LAN. Virtual bridges are special purpose network interfaces used in virtualized environments.

Let us consider that the physical host has a NIC called `eno1`. The entire virtualized network in the diagram then has to communicate with any external networks via this interface. However, they can't all just send their packets to the driver of the NIC. Thus, they need a virtual bridge `virbr0`. There can be multiple virtual bridges too.

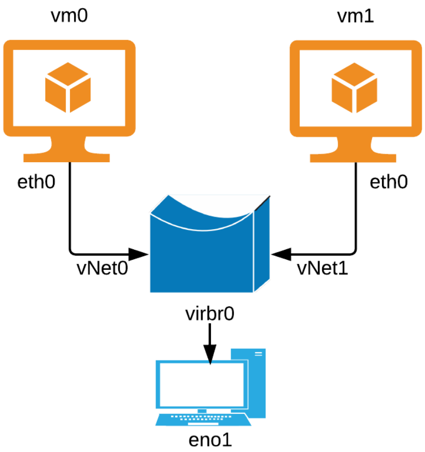

Figure 6.3: A virtualized network

The virtual bridge acts like a physical switch in the network and merely passes data between the networks. Note that it is incapable of routing decisions. All network traffic - even the traffic that originates from the physical KVM host are handled by it and thus, the virtual bridge decides who can send their packets at a specific moment.

Each of the virtual machines have their own virtualized Ethernet interface called `eth0` which have to be connected to an interface (port) on the virtual bridge. The virtual bridge names them `vnet0` and `vnet1` accordingly.

6.4.1 Working with Network Bridges

Let us consider a physical host with two KVM virtual machines running on it. Then, we can see their details using:

```
# virsh list --all
Id     Name                               State
----------------------------------------------------
3      vm0                                running
4      vm1                                running
```

Linux has an inbuilt layer 2 Ethernet bridge. This can be controlled using the `brctl` command. The status of the devices (VMs) connected to the bridge can be viewed with:

```
# brctl show
bridge name          bridge id                 STP enabled          interfaces
virbr0               8000.525400683445         yes                  virbr0-nic
                                                                    vnet0
                                                                    vnet1
```

The `vnet0` and `vnet1` interfaces are from the `vm0` and `vm1` virtual machines that are running on the host machine. The details of these interfaces can be seen with:

```
# ip link show
...
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
 ↪  DEFAULT qlen 1000
link/ether 00:0c:29:d8:97:c2 brd ff:ff:ff:ff:ff:ff
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DEFAULT
 ↪  qlen 1000
link/ether 52:54:00:68:34:45 brd ff:ff:ff:ff:ff:ff
...
7: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
 ↪  UNKNOWN mode DEFAULT qlen 1000
link/ether fe:54:00:0d:4a:d5 brd ff:ff:ff:ff:ff:ff
8: vnet1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master virbr0 state
 ↪  UNKNOWN mode DEFAULT qlen 1000
link/ether fe:54:00:21:93:04 brd ff:ff:ff:ff:ff:ff
```

The virtual bridge `virbr0` connects several devices together: the virtual ethernet interfaces from the VMs, `vmnet0` and `vmnet1` to the external LAN via interface `ens33`, which is the NIC for the physical host. The virtual bridge only shows active interfaces connected to it, i.e., only when the VMs are running will they appear on the output of `brctl show`.

6.4.2 Difference between network device and interface

In terms of hardware, a device refers to the physical NIC that's connected to the host, while an interface refers to the physical port that an Ethernet cable is plugged into, i.e., the hardware Ethernet port. Back when each NIC had only one interface, the terms *device* and *hardware* meant the same thing. However, with the advent of NICs with multiple interfaces on the same NIC, for example dual or quad port configurations, interfaces refer to each separate hardware port that's capable of accepting a network cable. Thus, for a NIC with 4 hardware ports, the single device has 4 interfaces.

Linux however, doesn't see these interfaces as connected devices (unless configured to do so) and treat them like separate hardware devices, even though they're on the same card! So, there can be multiple interfaces per hardware device. However, from Linux's perspective, they're all separate network devices, thus making the terms *interface* and *device* synonymous.

6.5 Setting up Network Bridges

The roles of the interfaces on the bridge are defined by the connections (profiles) for the available interfaces. Thus, we generate new profiles for the interfaces that we want to act as slaves, and one connection for the interface that we intend to act as master.

6.5.1 Creating a slave interface on the bridge

The package needed to set up software bridges on RHEL 7 is called `bridge-utils`. To set up bridging all connected interfaces need to be disconnected and then connected to the bridge. The connected interface can be viewed with:

```
# nmcli dev show
GENERAL.DEVICE:                         ens33
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         00:0C:29:3B:B9:1C
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
GENERAL.CONNECTION:                     ens33
GENERAL.CON-PATH:
 ↪  /org/freedesktop/NetworkManager/ActiveConnection/1
WIRED-PROPERTIES.CARRIER:               on
IP4.ADDRESS[1]:                         10.0.99.11/24
IP4.GATEWAY:                            10.0.99.2
IP4.DNS[1]:                             10.0.99.2
IP6.ADDRESS[1]:                         fe80::f408:1ebf:7742:9fd8/64
...
```

Now, we disconnect the devices using:

```
# nmcli dev disconnect ens33
Device 'ens33' successfully disconnected.
```

Once the disconnection is complete, we can now start defining the bridge, by adding an interface connection with:

```
# nmcli con add con-name br0-port1 type bridge-slave master br0 ifname ens33
Warning: master='br0' doesn't refer to any existing profile.
Connection 'br0-port1' (3dee7b9b-6197-4eb7-be8d-46290361b9fd) successfully added.
```

In this command, we have created a new connection with the name `br0-port1` and connected it to the interface *ens33*, which refers to the hardware NIC of the host. We've set the connection type to **bridge-slave**, which refers to the fact that the details of the incoming connection to the *slave* interface (*ens33*) will be configured at the bridge. The master of the new slave connection is set to a connection called *br0* (which doesn't yet exist, leading to the warnings).

The advantage of the master-slave configuration in network bridges is that if there are many connected slave interfaces, we need not set up their properties individually, and the master (bridge) thus provides a central point of configuration. The above slave configuration has to be repeated for every slave interface that we wish to connect to the bridge.

6.5.2 Creating a master interface on the bridge

The connection for the master interface will determine the settings for all the slave interfaces connected to it. We create the master connection by setting the type to `bridge` (instead of `bridge-slave`, unlike the previous cases):

```
# nmcli con add con-name br0 type bridge ifname br0
Connection 'br0' (493c6288-7f40-4a9b-9ac0-aee7a7aebe71) successfully added.
# brctl show
bridge name         bridge id               STP enabled         interfaces
br0                 8000.000000000000       yes
```

The NetworkManager must be restarted to be actually able to use any of this, since these only refer to the configuration in scripts `/etc/sysconfig/network-scripts` that need to be created by the NetworkManager.

```
# cd /etc/sysconfig/network-scripts/
# ls ifcfg-*
ifcfg-br0  ifcfg-br0-1  ifcfg-br0-port1  ifcfg-ens33  ifcfg-lo
```

The contents of the interface configuration file for the br0 master port:

```
DEVICE=br0
STP=yes
BRIDGING_OPTS=priority=32768
TYPE=Bridge
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=br0
UUID=5da1229d-27f1-4261-8491-e30046b9d03d
ONBOOT=yes
```

Since no information about IPs were provided, the boot protocol was chosen to be *DHCP* automatically. The slave interfaces only have the configuration:

```
TYPE=Ethernet
NAME=br0-port1
UUID=3dee7b9b-6197-4eb7-be8d-46290361b9fd
DEVICE=ens33
ONBOOT=yes
BRIDGE=br0
```

Since the configuration via the `nmcli` utility is hard to remember, it's man page has a link to the *nmcli-examples* man page, which has specific examples on setting up a bridge connection, as well as much more of the `nmcli` functionality.

6.6 Understanding Network Bonds and Teams

Both network bonds and teams accomplish roughly the same goal - the aggregation of links or network interfaces to form Link Aggregation Groups (LAG) or virtual links. This means several physical/logical interfaces can be combined to form a *team* that together fulfil a responsibility. Thus, one link may be set up as a primary connection to the WAN while the other may act as a backup or they both may be configured to act together while load balancing. Network bonding has been deprecated in RHEL 7 and thus we'll concentrate on Network teaming.

Network bonding used to perform the same responsibility as network teaming, but in the user space. Contrastingly, network teaming works with a kernel driver but also has a user space daemon, called **teamd**. This *teamd* daemon has several modes of operation called *runners*. These determine the function of the ports in the team and have possible values of: *broadcast, roundrobin, activebackup, loadbalance* and *lacp*.

Terms	Description
broadcast	Any packet is broadcast all over the interfaces.
roundrobin	The port which can transmit data is chosen in a roundrobin fashion.
activebackup	One of the interfaces stays active while the other is backup, ready to kick in the moment the active interface fails.
loadbalance	The network load (i.e., the packets in the network) is split between the interfaces so as to not overload any single interface.
lacp	Link Aggregation Control Protocol - allows formation of LAGs on a peer by automatically negotiating by transmitting LACP packets.

The command to control and manage teams is called `teamdctl`. Thus, to show the state of the team called *team0*, we'd use: `teamdctl team0 state`.

6.7 Configuring Network Teams

There are four parts to creating a team:

- Creating a team interface
- Determining the network configuration
- Assigning the port interfaces
- Bring team and port interfaces up and down respectively.

Once the above has been taken care of, the team connection can be verified with `teamdctl team0 status` (assuming *team0* is the name of the team).

6.7.1 Creating the team interface

We have to create the new team connection, preferably with the same interface name as the team:

```
# nmcli con add type team con-name team0 ifname team0 config '{"runner": {"name":
    "loadbalance"}}'
Connection 'team0' (d0eba200-591c-4ce3-a089-12d8a5692243) successfully added.
```

In this command, we have created a connection of type *team*, which indicates that it'll be a link aggregate. We provide a name to it called *team0* and connect it to an interface called team0 (which is logical - the interface that'll act as the aggregate of the member links). Finally, we provide the configuration as a JSON array: `'{"runner": {"name": "loadbalance"}}'`. This sets the team to act as a load balancer, and thus split the load of the packets over all the interfaces configured in the team.

6.7.2 Determining the network configuration

The team needs to be configured to use an IP address to use as its interface (i.e., team interface). This is specified using the CIDR (Classless Inter-Domain Routing) notation with a Network IP address and a prefix. From this both the Network ID and the Subnet mask can be determined.

```
# nmcli con mod team0 ipv4.addresses 10.0.0.10/24
# nmcli con mod team0 ipv4.method manual
```

Note that the `mod` command uses `ipv4` instead of `ip4`, unlike the `nmcli add` command. The IP assignment method also needs to be switched to manual since DHCP isn't involved here.

6.7.3 Assigning the port interfaces

Now that the master (team) interface has a port defined for it, we also need to assign the individual interfaces that are going to be slaves to the team. This is done using:

```
# nmcli con add type team-slave con-name team0-ens33 ifname ens33 master team0
Connection 'team0-ens33' (78e706bd-395c-456f-b16a-75430c48be2c) successfully added.
# nmcli con add type team-slave con-name team0-enss37 ifname ens37 master team0
Connection 'team0-enss37' (d17ad2f5-9a55-422a-ad5d-f4b327674393) successfully added.
```

The command defines two interfaces (*eth0* and *eth1*) to be slaves to the team interface. They are named according to the format `<teamName>-<interfaceName>`. Now we only need to define a master for them from which they can be controlled.

6.7.4 Bringing the team and port interfaces up/down

Once the above sections have been handled, the team is basically ready for operation. However, we still need to bring the physical devices (that are slaves to the team) to be disconnected and then reconnected as part of the team. Since we've already defined them as a part of the team, we just need to:

```
# nmcli con up team0
Connection successfully activated (master waiting for slaves) (D-Bus active path:
 ↪ /org/freedesktop/NetworkManager/ActiveConnection/7)
# nmcli dev disconnect ens33; nmcli dev dis ens37
Device 'ens33' successfully disconnected.
Device 'ens37' successfully disconnected.
```

The devices *eth0* and *eth1* needed to be disconnected because they're slaves to the team now, and thus, their operation should only be influenced by the team itself. Thus, there's no point in having them exist as separate individual devices (network interfaces).

6.7.5 Verifying the team connection

We can verify the team connection by:

```
# teamdctl team0 state
setup:
  runner: loadbalance
ports:
  ens33
    link watches:
      link summary: up
```

```
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
  ens37
    link watches:
      link summary: up
      instance[link_watch_0]:
        name: ethtool
        link: up
        down count: 0
```

6.7.6 Creating a bridge based on Network Teams

When creating bridges based on network teams, it becomes important to switch off NetworkManager since they're incompatible. The team configuration file `ifcfg-team0` needs to be edited to add the line `BRIDGE=brteam0` has to be added to it to ask it to connect to the bridge device.

Since the team interfaces will be slaves to the connection provided by the team, it's important to ensure that there are no IP configurations in the `ifcfg-team0-port` files anymore. Basically, we need to disable the team driver since the bridge will now control the team interface. For this we do:

```
# nmcli dev dis team0
# systemctl stop NetworkManager; systemctl disable NetworkManager
```

Now we can manually create a configuration file for the bridge connection that has the contents:

```
DEVICE=brteam0
TYPE=Bridge
IPADDR0=192.168.122.100
PREFIX0=24
```

Finally, since we're not using the `NetworkManager` service, we directly restart the networking with: `systemctl restart network`. Now, the bridge on top of the team interface should be active and operational. Again, examples for these congigurations are present in the nmcli-examples man page, accessible with `man 5 nmcli-settings`.

6.8 Configuring IPv6

Since there is a shortage of unique IPv4 addresses when compared to the number of devices that are connected to the internet, IPv6 is the new standard for the IP addresses. Just like IPv4 addresses, it can be divided into two parts: the network ID and the host ID. However, in the case of the IPv6 addresses, the host ID contains the MAC address of the interface itself! A typical IPv6 address looks like:

```
fe80::2af4:9908:7092:34cb/64
```

In this, the network ID is `fe80` while the node part is the `2af4:9908:7092:34cb` with the prefix of `64` defining how long the network ID is. Thus, the computer listens for the Network ID, and then appends it's own MAC address to obtain a truly unique IPv6 address! For configuring connections with both IPv6 and IPv4 addresses, nmcli can be used as:

```
# nmcli con add con-name testCon type ethernet ip6 2001:db8:0:10::d000:310/64 gw6
↪ 2001:db8:0:10::1 ip4 192.168.0.10/24 gw4 192.168.0.1
```

Again, for new connection just like for IPv4 connections where `nmcli con add` takes as argument the term **ip4** instead of *ipv4* (unlike the `con mod` command, which accepts *ipv4*), the `nmcli con add` needs an argument of **ip6** and not *ipv6*. Now to add a DNS server for it:

```
# nmcli con mod testCon +ipv6.dns 2001:4680:4680::8888
```

Chapter 7

Managing Linux Based Firewalls

Chapter 8

Configuring Apache Virtual Hosts

Chapter 9

Managing Advanced Apache Features

Part III

DNS and File Sharing

Chapter 10

Configuring a Cache-only DNS Server

Chapter 11

Configuring NFS File Sharing

Chapter 12

Managing SMB File Sharing

Part IV

Essential Back-end Services

Chapter 13

Setting up an SMTP Server

Chapter 14

Managing SSH

Chapter 15

Managing MariaDB

Chapter 16

Managing Time Services

Chapter 17

Shell Scripting