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.

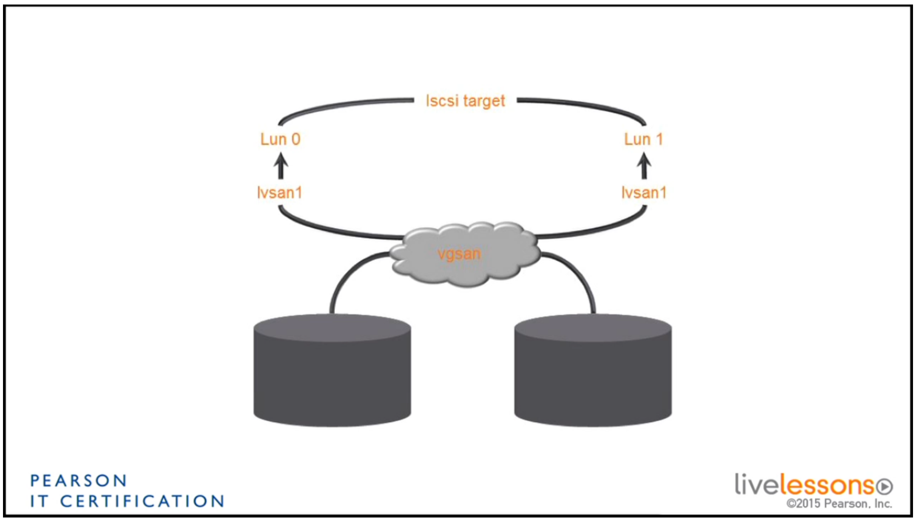

Figure 2.1: iSCSI Target Setup

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`.

Chapter 4

System Optimization Basics

Chapter 5

Configuring Logging

Part II

Networking and Apache

Chapter 6

Configuring Advanced Networking

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