Unit-VI
Securing Ecosystem

Overview

- Big data promises to help organizations better understand their businesses, their customers and their environments to a degree that you could previously have only imagined.
- The potential is enormous—as businesses transform into data-driven machines, the data held by your enterprise is likely to become the key to your competitive advantage.
- As a result, security for both your data and your infrastructure becomes more important than ever before.

Overview

- In the case of data that provides a competitive advantage, the need for security should be obvious.
- If you lose that data, or it winds up in the hands of a competitor, your advantage is lost……But worse, it could become a liability
- Network security breaches from internal and external attackers are on the rise, often taking months to be detected, and those affected are paying the price
- Organizations that have not properly controlled access to their data sets are facing lawsuits, negative publicity, and regulatory fines.

Overview

Consider the following eye-opening statistics:

- A study released this year by Symantec and the Ponemon Institute found that the average organizational cost of one security breach in the United States is 5.4 million dollars. Another recent study shows that the cost of cybercrime in the U.S. economy alone is 140 billion dollars per year.
- One of the largest breaches in recent history involved Sony's Playstation Network in 2011, and experts estimate Sony's costs related to the breach to be somewhere between 2.7 and 24 billion dollars (a wide range, but the breach was so large, it is almost impossible to quantify).
- Netflix and AOL have already faced (and in some cases, settled) millions of dollars in lawsuits over their management of large sets of data and their protection of personal information – even data that they had "anonymized" and released for research.
- Beyond quantifiable costs related to security breaches (loss of customers and business partners, lawsuits, regulatory fines), organizations that have experienced such incidents report that the fallout from a data breach results in a diminished trust of the organization and a damaged reputation that could put a company out of business.

Overview

Few of the questions that the infrastructure administrators and security paranoids would ask are:

- ❑ How secure is a Hadoop ecosystem? How secure is the data residing in Hadoop?
- ❑ How would different teams including business analysts, data scientists, developers, and others in the enterprise access the Hadoop ecosystem in a secure manner?
- ❑ How to enforce existing **Enterprise Security Models** in this new infrastructure?
- ❑ Are there any best practices for securing such an infrastructure?

Why do we need to secure Hadoop?

- ❑ Enterprise data is locked securely within systems such as ERP, CRM, and general ledger systems.
- ❑ In the last decade, enterprise data security has matured significantly
- ❑ Security and privacy standards such as HIPAA, HITECH, PCI, SOX, ISO, and COBIT have evolved.
- ❑ Resulted in a very protective data security enforcement within enterprises including service providers as well as the clients.
- ❑ There is absolutely no tolerance to data security violations.
- ❑ Hadoop has now reached a mature state where enterprises have started adopting it for their Big Data processing needs.
- ❑ Need to bring data to the Hadoop ecosystem for processing.
- ❑ So the immediate question that arises with respect to data security is, how secure is the data storage inside the Hadoop ecosystem?

Why do we need to secure Hadoop?

- ❑ The question is not just about securing the source data which is moved from the enterprise systems to the Hadoop ecosystem.
- ❑ Once these datasets land into the Hadoop ecosystems, analysts and data scientists perform large-scale analytics and machine-learning-based processing to derive business insights.
- ❑ These business insights are of great importance to the enterprise.
- ❑ Any such insights in the hands of the competitor or any unauthorized personnel could be disastrous to the business.
- ❑ It is these business insights that are highly sensitive and must be fully secured.
- ❑ Any data security incident will cause business users to lose their trust in the ecosystem.
- ❑ Unless the business teams have confidence in the Hadoop ecosystem, they won't take the risk to invest in Big Data.
- ❑ Hence, the success and failure of Big Data-related projects really depends upon how secure our data ecosystem is going to be.

Challenges for securing the Hadoop ecosystem

- Big Data not only brings challenges for storing, processing, and analysis but also for managing and securing these large data assets.
- Hadoop was not built with security to begin with.
- As enterprises started adopting Hadoop, the Kerberos-based security model evolved within Hadoop.
- But given the distributed nature of the ecosystem and wide range of applications that are built on top of Hadoop, securing Hadoop from an enterprise context is a big challenge.
- A typical Big Data ecosystem has multiple stakeholders who interact with the system.
- One of the biggest challenges for Big Data projects within enterprises today is about securely integrating the external data sources.
- Hadoop ecosystem tools such as Sqoop and Flume were not built with full enterprise grade security.
- Cloudera, MapR, and few others have made significant contributions towards enabling these ecosystem components to be enterprise grade, resulting in **Sqoop 2, Flume-ng, and Hive Server 2**.
- Security-focused projects within the Hadoop ecosystem
  - Cloudera Sentry
  - Hortonworks Knox Gateway
  - Intel's Project Rhino

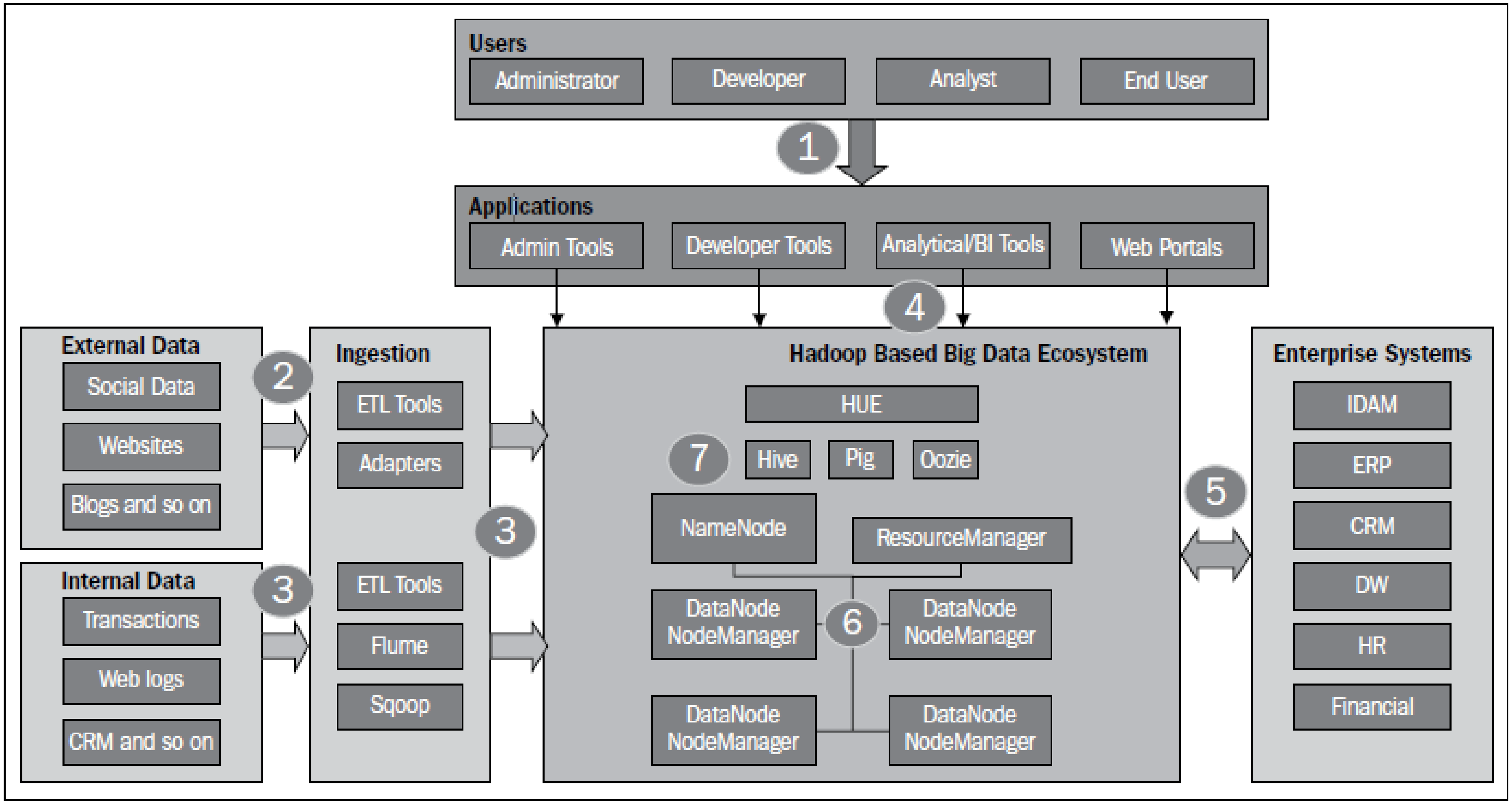
Challenges for securing the Hadoop ecosystem

- ❑ Need the existing enterprise **Identity and Access Management** (**IDAM**) systems with the Hadoop ecosystem.
- ❑ With such integration, enterprises can extend the Identity and Access Management to the Hadoop ecosystem.
- ❑ However, these integrations bring in multiple challenges as Hadoop inherently has not been built with such enterprise integrations in mind.
- ❑ Apart from ecosystem integration, there is often a need to have sensitive information within the Big Data ecosystem, to derive patterns and inferences from these datasets.
- ❑ As sensitive information in moved to the Big Data ecosystem we need to mask/encrypt this sensitive information.
- ❑ Traditional data masking and encryption tools don't scale well for large scale Big Data masking and encryption.
- ❑ Need to identify new means for encryption of large scale datasets.

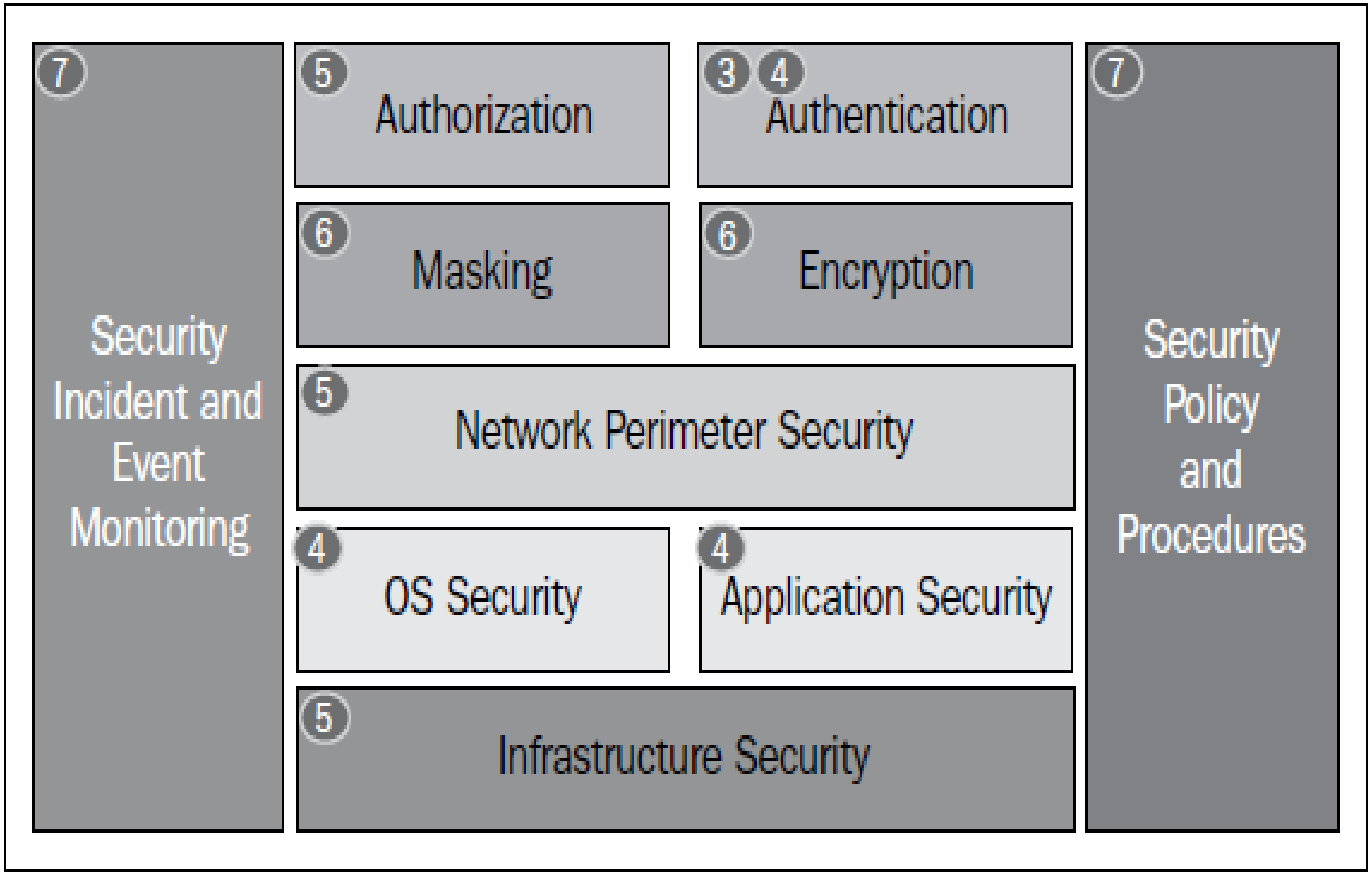
Key security considerations

- To meet the enterprise data security needs for a Big Data ecosystem, a complex and holistic approach is needed to secure the entire ecosystem.
- Some of the key security considerations while securing Hadoop-based Big Data ecosystem are:
  - Authentication:
  - Authorization:
  - Access control
  - Data masking and encryption
  - Network perimeter security
  - System security
  - Infrastructure security
  - Audits and event monitoring

Reference architecture for Big Data security



Key Security Pillars



Tips for Securing Big Data

- Think about security before you start your big data project
- Consider what data may get stored
- Centralize accountability
- Encrypt data both at rest and in motion
- Separate your keys and your encrypted data
- Use the Kerberos network authentication protocol
- Use secure automation
- Add logging to your cluster
- Implement secure communication between nodes and between nodes and applications

Kerberos

- ❑ Need security for an end-to-end Hadoop-based Big Data ecosystem.
- ❑ Hadoop security was implemented as part of the HADOOP-4487 Jira issue, starting in late 2009
- ❑ Currently, there are efforts to implement Single-Sign-On (SSO) Authentication in Hadoop.
- ❑ This is currently not production-ready, and hence will be out of scope of this book.
- ❑ Hadoop security implementation is based on Kerberos.

What is Kerberos?

- In any distributed system, need to establish trust between communicating parties. This is usually done through the authentication process.
- Risk of passwords getting compromised as they travel through the network.
- Kerberos is a secured network authentication protocol that provides strong authentication for client/server applications without transferring the password over the network.
- Kerberos works by using time-sensitive tickets that are generated using the symmetric key cryptography.
- Kerberos is derived from the Greek mythology where Kerberos was the three-headed dog that guarded the gates of Hades.
- The three heads of Kerberos in the security paradigm are:
  - The user who is trying to authenticate.
  - The service to which the client is trying to authenticate.
  - Kerberos security server known as **Key Distribution Center (KDC),** which is trusted by both the user and the service.

Key Kerberos terminologies

- KDC provides two main functionalities known as
  1. **Authentication Service (AS)** and
  2. **Ticket Granting Service** (**TGS**).
- AS is responsible for authenticating users and services, while the TGS provides a ticket that is a time-limited cryptographic message, used by the client to authenticate with server.
- Parties involved in communication (the client and server) are known as **principals**.
- Each party has a principal that is defined within KDC.
- Each party shares the secret key (password) with the KDC.
- Passwords can be stored locally within KDC, but it is good practice to manage this centrally using LDAP.
- Each KDC is associated with a group known as a **realm**.
- Principals defined within a single KDC are in the same realm.

Key Kerberos terminologies

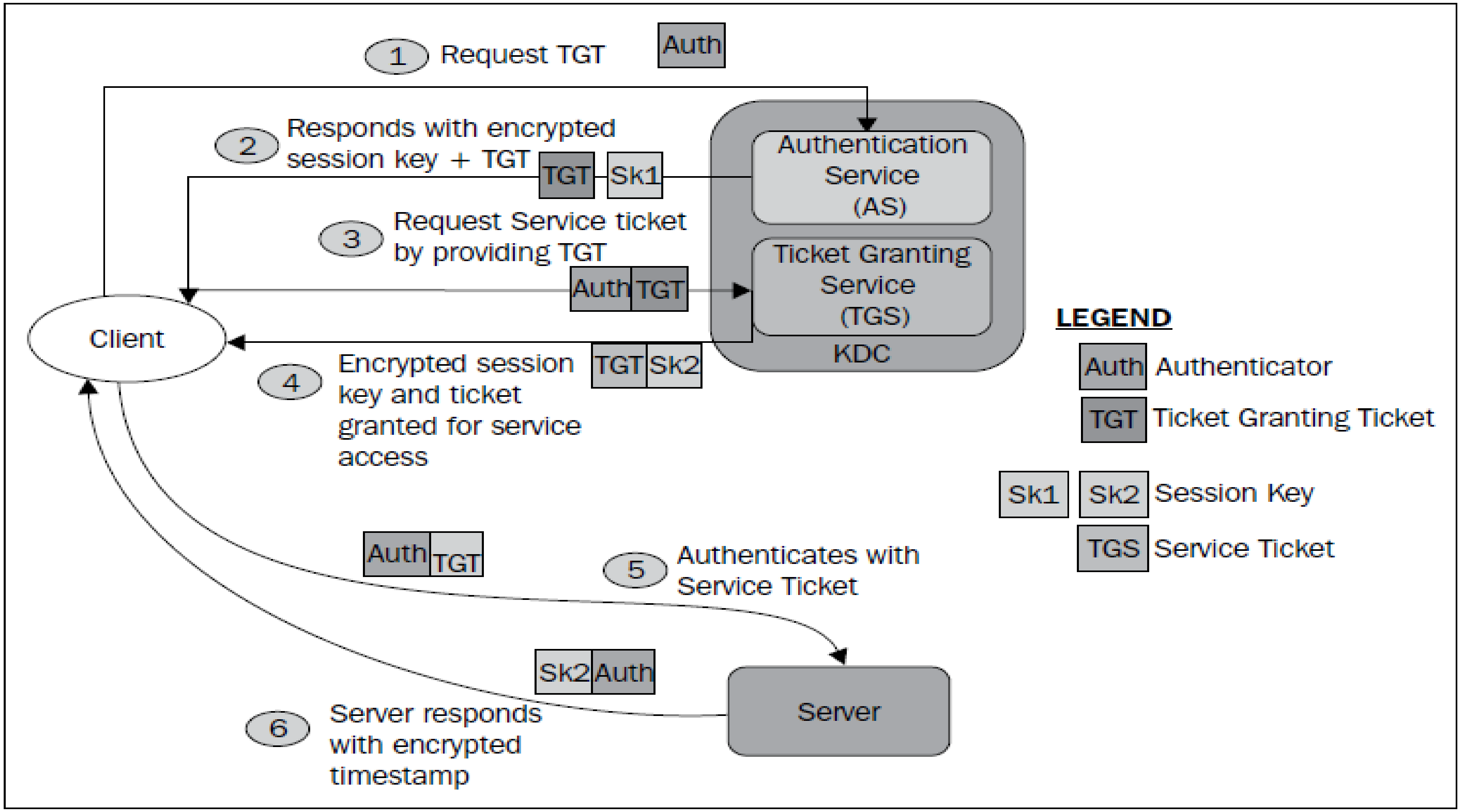
- There could be multiple KDCs, and hence multiple realms in the network.
- In a multiple realm scenario, a client that authenticates with one realm can connect to the server defined in another realm, as long as there is trust established between the two realms/KDCs.
- KDC consists of two main daemons.
  1. **krb5kdc**: This is the Kerberos Authentication Server and is responsible for authenticating the client and also granting tickets.
  2. **kadmind**: This is the administrative server daemon and is responsible for performing administrative operations such as adding a new principal, changing passwords, and such other activities on KDC.

Key Kerberos terminologies

Kerberos provides multiple utilities that are useful for working with

- **KDC. `kadmin` and `kadmin.local`**: These are administrative clients for the **kadmind** daemon to perform administrative tasks. `kadmin.local` directly connects to the KDC database on the local server, while the `kadmin` process allows for remote connectivity.
- **kinit**: This utility is used to authenticate with the KDC and fetch the Kerberos ticket for the user.
- **klist**: This utility is used to list the Kerberos tickets currently in the client’s local credential cache.
- **ktutil**: This is the key tab file utility that is used to update and view the key tab file.
- **kdb5_util**: This is the KDC database utility. This is used to create the KDC database and maintain it.

How Kerberos works?


1 Request TGT
Auth
2 Responds with encrypted session key + TGT
TGT
Sk1
Authentication Service (AS)
3 Request Service ticket by providing TGT
Auth
TGT
Ticket Granting Service (TGS)
KDC
Client
4 Encrypted session key and ticket granted for service access
TGT
Sk2
LEGEND
Auth Authenticator
TGT Ticket Granting Ticket
Sk1 Sk2 Session Key
TGS Service Ticket
Auth
TGT
5 Authenticates with Service Ticket
Sk2
Auth
Server
6 Server responds with encrypted timestamp

Detailed authentication and authorization flow in a Kerberos cluster

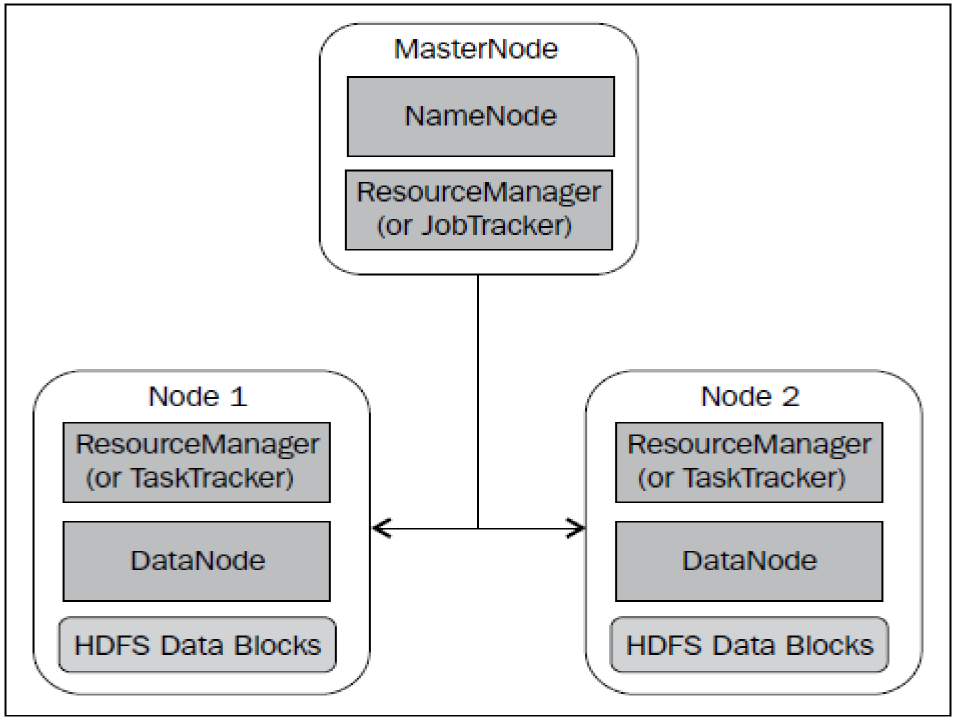
Kerberos advantages

The key advantages of using Kerberos are:

- ❑ A password never travels over the network. Only time-sensitive tickets travel over the network.
- ❑ Passwords or secret keys are only known to the KDC and the principal.
- ❑ Kerberos supports passwords or secret keys to be stored in a centralized credential store that is LDAP-complaint.
- ❑ Servers don't have to store any tickets or any client-specific details to authenticate a client.
- ❑ The client authenticates with KDC and gets the TGT that is used for all subsequent authentications.

Hadoop default security model without Kerberos

- Hadoop implements a security model similar to the POSIX filesystem.
- By default, Hadoop doesn't support any authentication of users or Hadoop services.
- User only authenticates with operating system during logon process.



MasterNode
NameNode
ResourceManager (or JobTracker)
Node 1
ResourceManager (or TaskTracker)
DataNode
HDFS Data Blocks
Node 2
ResourceManager (or TaskTracker)
DataNode
HDFS Data Blocks

Hadoop services and data blocks

Hadoop Kerberos security implementation

- ❑ Enforcing security within a distributed system such as Hadoop is complex.
- ❑ The detailed requirements for securing Hadoop were identified by Owen O'Malley and others as part of the Hadoop security design.
- ❑ The detailed document is attached with the ticket HADOOP-4487 at `https://issues.apache.org/jira/browse/HADOOP-4487`.
- ❑ We will discuss summary of these requirements.

Hadoop Kerberos security implementation

User-level access controls

A brief on the user-level access controls is:

- Users of Hadoop should only be able to access data that is authorized for them.
- Only authenticated users should be able to submit jobs to the Hadoop cluster.
- Users should be able to view, modify, and kill only their own jobs
- Only authenticated services should be able to register themselves as DataNodes or TaskTracker.
- Data block access within DataNode needs to be secured, and only authenticated users should be able to access the data stored in the Hadoop cluster.

Hadoop Kerberos security implementation

Service-level access controls

Here's a gist of the service-level access controls:

- ❑ **Scalable Authentication**: Hadoop clusters consist of a large number of nodes, and the authentication models should be scalable to support such large network authentication
- ❑ **Impersonation**: Hadoop services should be able to impersonate the user submitting the job so that the correct user isolation can be maintained
- ❑ **Self-Served**: Hadoop jobs run for long durations, so they should be able to ensure that the jobs are able to self-renew the delegated user authentication to complete the job
- ❑ **Secure IPC**: Hadoop services should be able to authenticate each other and ensure secured communication between themselves.

➢ To achieve preceding requirements, Hadoop leverages the Kerberos authentication protocol and some internal-generated tokens to secure the Hadoop cluster.

Hadoop Kerberos security implementation

User and service authentication

- User authentication to NameNode and JobTracker services is through Hadoop's remote procedure call using the Simple Authentication and Security Layer (SASL) framework.
- Kerberos is used as authentication protocol to authenticate users within SASL.
- All Hadoop services support Kerberos authentication.
- A client submits the MapReduce job to JobTracker. MapReduce jobs are usually long-running jobs and they need to access the Hadoop resources on behalf of the user. This is achieved using Delegation Token, Job Token, and Block Access Token.

Hadoop Kerberos security implementation

Delegation Token

- A **Delegation Token** authentication is a two-party authentication protocol based on JAVA SASL Digest-MD5.
- A Delegation Token is used between the user and NameNode to authenticate the user.
- Once the user authenticates themselves with NameNode using Kerberos, the user is provided with the Delegation Token by NameNode.
- The user doesn't have to perform Kerberos authentication once he/she obtains the Delegation Token.
- The user also designates the JobTracker or ResourceManager process as the user that will renew the Delegation Token as part of the Delegation Token request.
- The Delegation Token is secured and shared with JobTracker or ResourceManager after authentication.
- JobTracker will use Delegation Token for accessing HDFS resources on behalf of the user.
- JobTracker will automatically renew this Delegation Token for long-running jobs.

Hadoop Kerberos security implementation

Job Token

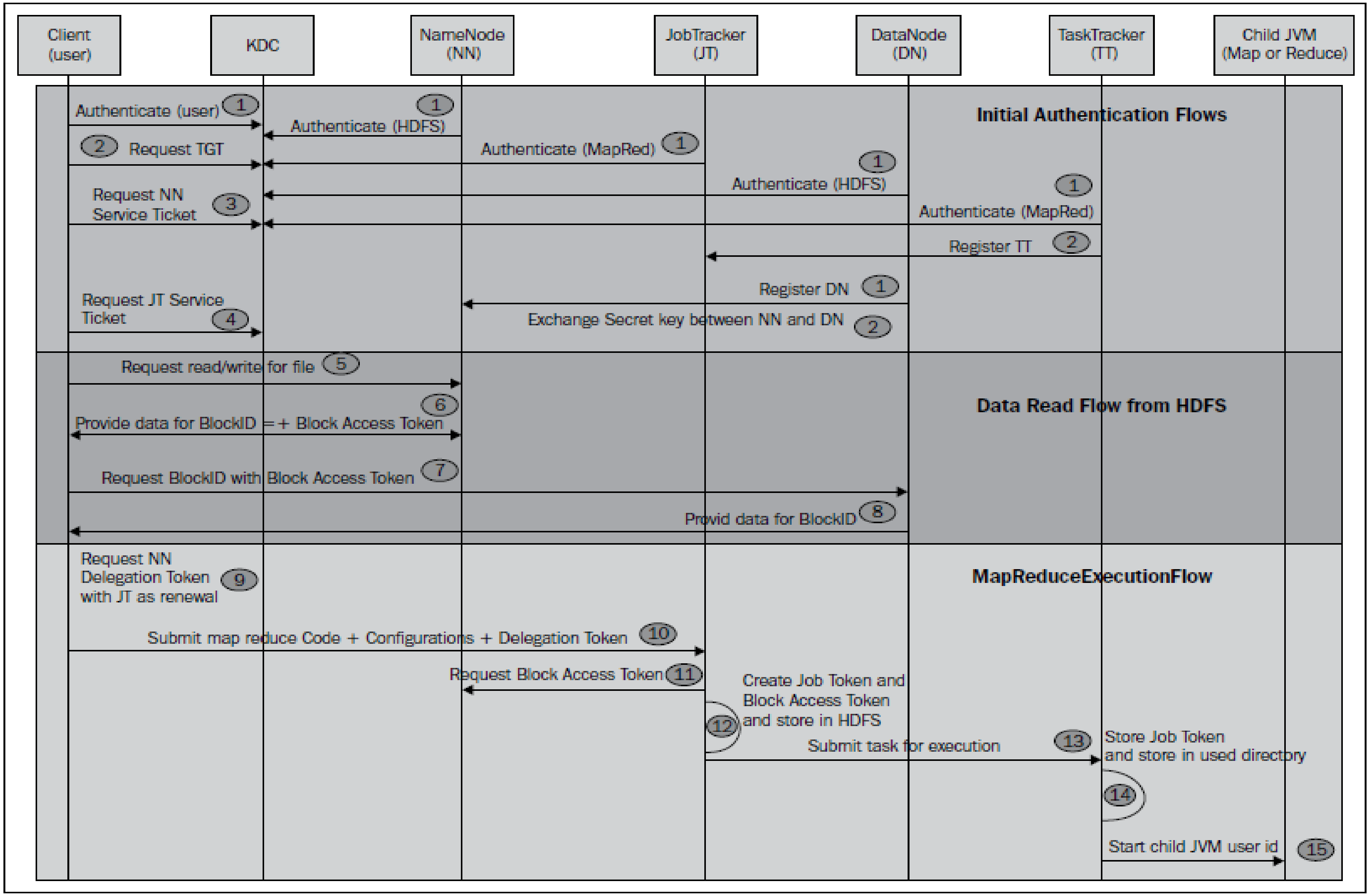
- A job runs on TaskNodes and user access has to be secured in TaskNodes.
- When the user submits MapReduce job to JobTracker, it will create a secret key (**Job Token)** that will be shared with TaskTracker that will run the MapReduce job.
- The **Job Token** will be stored in the local disk of TaskTracker with permission only for the user who submitted the job.
- TaskTracker starts child JVM task (mapper or reducer) using the user ID that submitted the job.
- Thus, the child JVM run will be able to access the Job Token from the local directory and communicate securely with TaskTracker using this Job Token.
- Thus, the Job Token is used to ensure that an authenticated user submitting the job in Hadoop has access to only the folders and jobs for which he is authorized in the local filesystem of TaskNodes.
- Once the Reduce jobs are started in TaskTracker, this TaskTracker contacts TaskTracker that ran the Map task and fetches the mapper output files.
- Job Token is also used by TaskTrackers to securely communicate with each other.

Hadoop Kerberos security implementation

Block Access Token

- ❑ Hadoop needs to fetch the data blocks directly from DataNode
- ❑ Need secured mechanism where the user privileges are securely passed to DataNode.
- ❑ The main purpose of the **Block Access Token (BAT)** is to ensure that only authorized users are able to access the data blocks stored in DataNodes.
- ❑ Authentication performed by NameNode is also enforced at DataNode, Hadoop implements the BAT.
- ❑ BAT is the token provided by NameNode to a Hadoop client to pass data access authentication information to DataNode.
- ❑ BAT is lightweight and contains expirationDate, keyID, ownerID, blockID, and accessModes.
- ❑ BAT generated by NameNode is not renewable and needs to be fetched again once the token expires.

Interactions in a secured Hadoop Cluster


Client (user)
KDC
NameNode (NN)
JobTracker (JT)
DataNode (DN)
TaskTracker (TT)
Child JVM (Map or Reduce)
Initial Authentication Flows
Authenticate (user) 1
Authenticate (HDFS) 1
2 Request TGT
Authenticate (MapRed) 1
Authenticate (HDFS) 1
Request NN Service Ticket 3
Authenticate (MapRed) 1
Register TT 2
Register DN 1
Request JT Service Ticket 4
Exchange Secret key between NN and DN 2
Data Read Flow from HDFS
Request read/write for file 5
Provide data for BlockID = + Block Access Token 6
Request BlockID with Block Access Token 7
Provid data for BlockID 8
MapReduceExecutionFlow
Request NN Delegation Token with JT as renewal 9
Submit map reduce Code + Configurations + Delegation Token 10
Request Block Access Token 11
Create Job Token and Block Access Token and store in HDFS 12
Submit task for execution 13
Store Job Token and store in used directory
14
Start child JVM user id 15

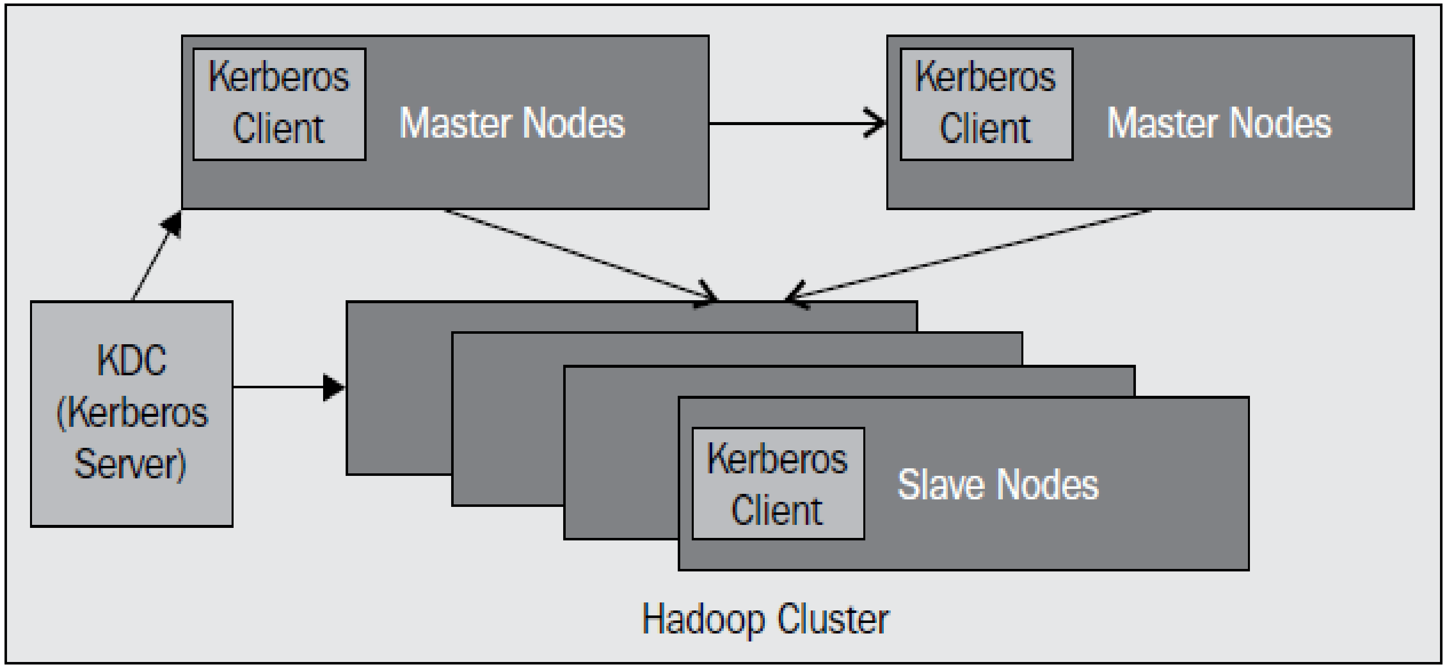
Setting up Kerberos - Prerequisites

The following are prerequisites for installing a secure Hadoop cluster:

- Root or sudo access for the user installing the cluster.
- Hadoop cluster is configured and running in a non-secured mode.
- Proper file permissions are assigned to local and Hadoop system directories.
- Incase, we are building Kerberos from the source code, we will need the GCC compiler
- DNS resolutions and host mappings are working for all machines in the cluster.
- The ports required for Kerberos are port 88 for KDC and port 749 for admin services.
- The name of the Kerberos realm that will be used for authenticating the Hadoop cluster.

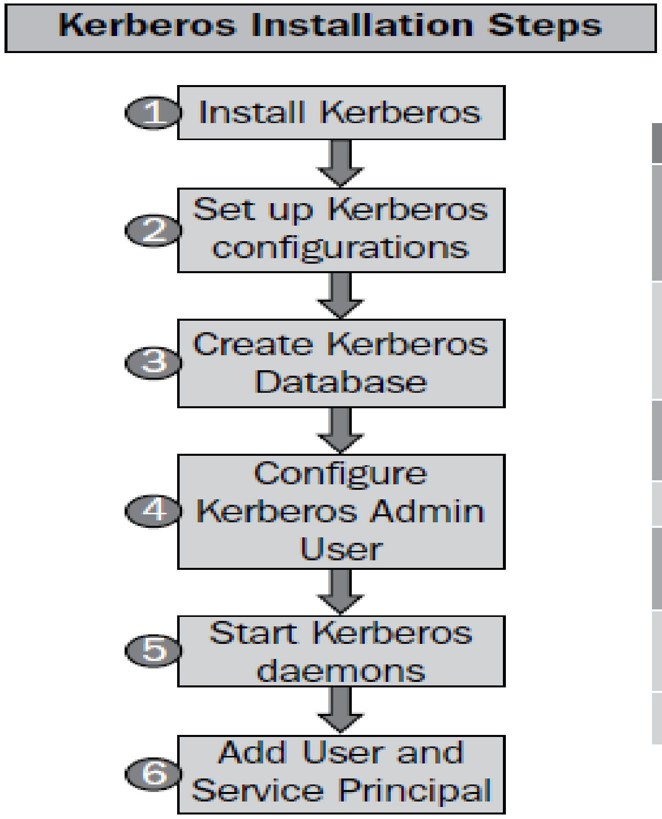
Setting up Kerberos

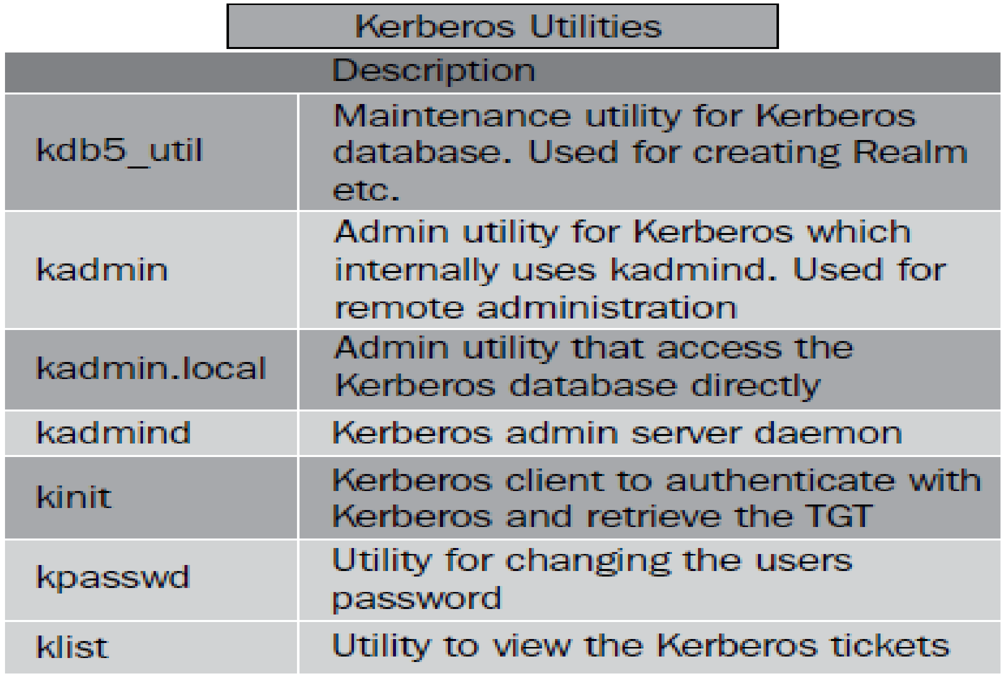
- ❑ The first step is to set up the Kerberos authentication and ensure that the Kerberos authentication for the Hadoop service principals are working for all the nodes on the cluster.
- ❑ To set up Kerberos, we establish a Kerberos Server (KDC) on a separate node and install the Kerberos client on all nodes of the Hadoop cluster.



Setting up Kerberos

- ❑ Figure illustrates high-level steps involved in installing and configuring Kerberos.
- ❑ It also shows the various Kerberos utilities that are available.


Kerberos Installation Steps
1 Install Kerberos
2 Set up Kerberos configurations
3 Create Kerberos Database
4 Configure Kerberos Admin User
5 Start Kerberos daemons
6 Add User and Service Principal



Kerberos Utilities

	Description
kdb5_util	Maintenance utility for Kerberos database. Used for creating Realm etc.
kadmin	Admin utility for Kerberos which internally uses kadmind. Used for remote administration
kadmin.local	Admin utility that access the Kerberos database directly
kadmind	Kerberos admin server daemon
kinit	Kerberos client to authenticate with Kerberos and retrieve the TGT
kpasswd	Utility for changing the users password
klist	Utility to view the Kerberos tickets

Installing the Key Distribution Center

- We will use the following realm and domain :

  Domain name: `mydomain.com`

  Realm name: `MYREALM.COM`

- To set up Kerberos, we need to install the **Key Distribution Center** (**KDC**) on a secured server.
- On RHEL/CentOS/Fedora, to install Kerberos, run the following command with root privileges:

```
yum install krb5-server krb5-libs krb5-workstation
```

- For Ubuntu distribution, the `apt-get` command to install Kerberos is as follows:

```
sudo apt-get install krb5-kdc krb5-admin-server
```

- Multiple distributions of the KDC implementations available, and each distribution caters to a specific version and distribution of Linux.

Configuring the Key Distribution Center

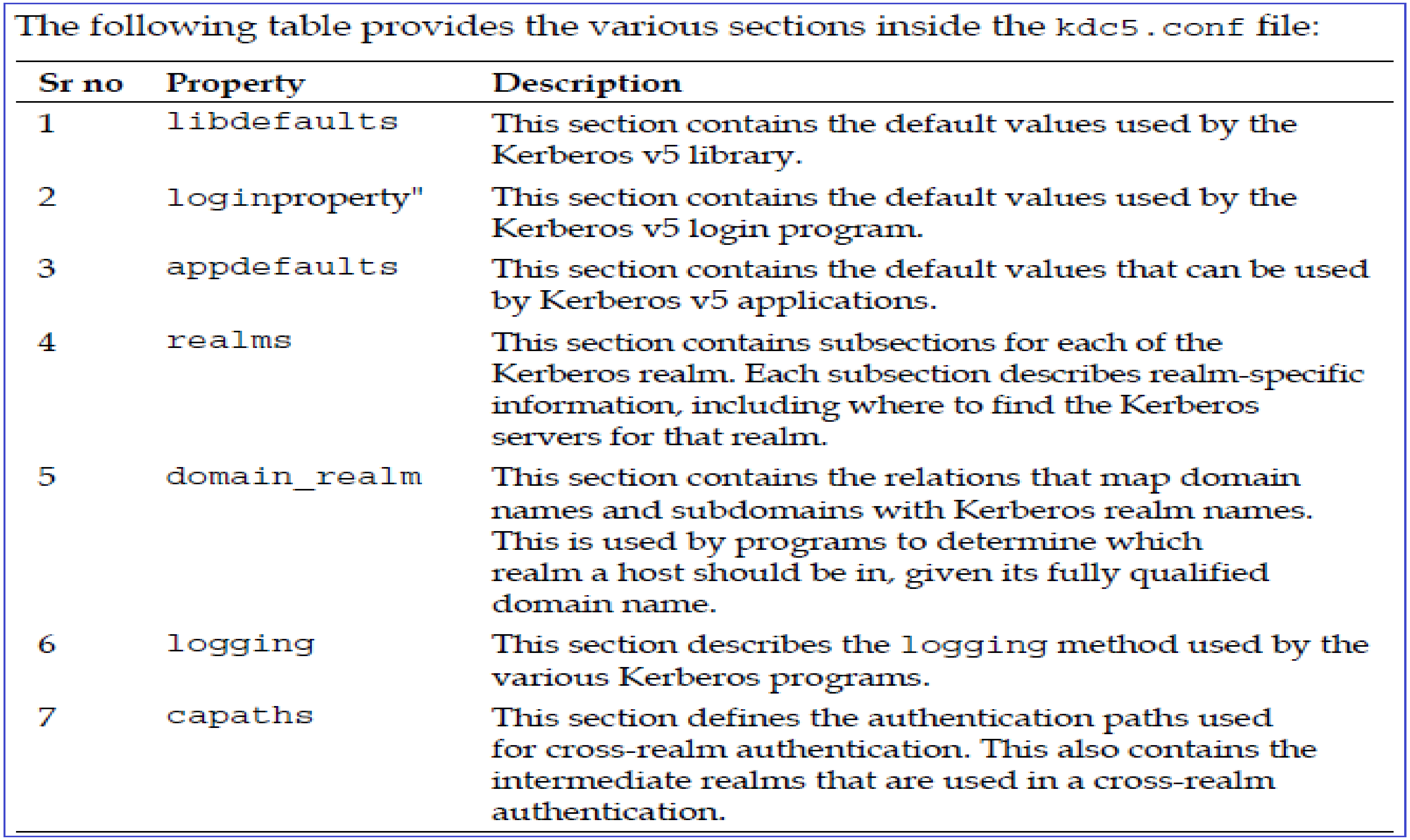
- ❑ We will look at Version 5 of MIT Kerberos.
- ❑ This has three configuration files.
  - ❑ The `krb5.conf file` is kept inside `/etc/ folder`.
  - ❑ The `kdc.conf` and `kadm5.acl` files are placed inside the `/usr/local/var/krb5kdc` folder.
- ❑ All of these configuration files follow the Windows INI file format.
- ❑ `krb5.conf` is a higher-level configuration and provides the configuration related to the location of KDCs, admin servers, and mappings of hostnames with Kerberos realms.
- ❑ Most of the configurations work with the default values for the current realm and Kerberos application.

Configuring the Key Distribution Center

The example configuration for the `krb5.conf` file are provided as follows:

```
[logging]
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log
[kdc]
profile = /usr/local/var/krb5kdc/kdc.conf
[libdefaults]
default_realm = MYREALM.COM
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true
[realms]
MYREALM.COM = {
kdc = KerberosServerHostname
admin_server = KerberosServerHostname
}
[domain_realm]
.mydomain.com = MYREALM.COM
mydomain.com = MYREALM.COM
```

Configuring the Key Distribution Center



The following table provides the various sections inside the `kdc5.conf` file:

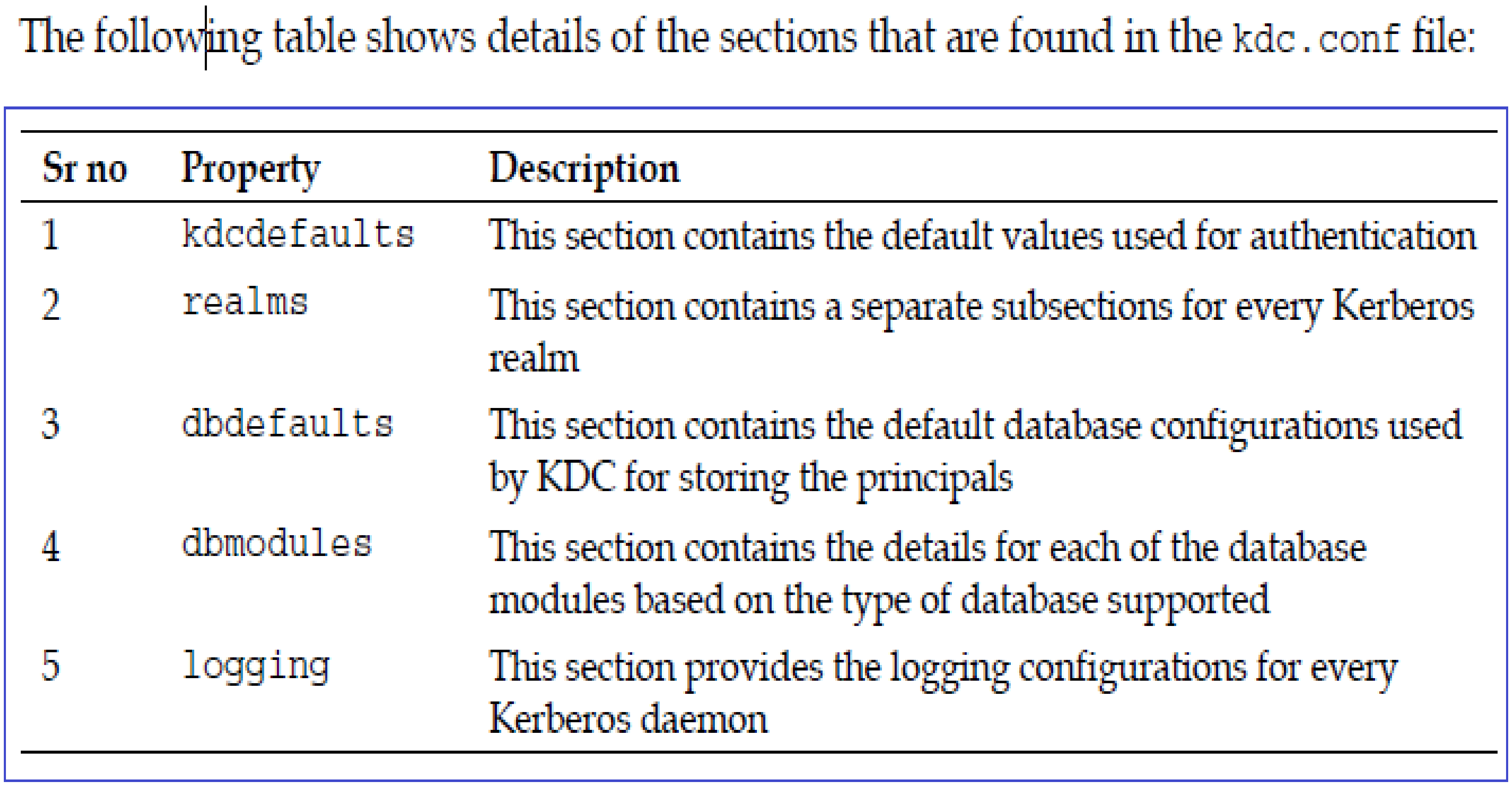
Sr no	Property	Description
1	`libdefaults`	This section contains the default values used by the Kerberos v5 library.
2	`loginproperty"`	This section contains the default values used by the Kerberos v5 login program.
3	`appdefaults`	This section contains the default values that can be used by Kerberos v5 applications.
4	`realms`	This section contains subsections for each of the Kerberos realm. Each subsection describes realm-specific information, including where to find the Kerberos servers for that realm.
5	`domain_realm`	This section contains the relations that map domain names and subdomains with Kerberos realm names. This is used by programs to determine which realm a host should be in, given its fully qualified domain name.
6	`logging`	This section describes the `logging` method used by the various Kerberos programs.
7	`capaths`	This section defines the authentication paths used for cross-realm authentication. This also contains the intermediate realms that are used in a cross-realm authentication.

Configuring the Key Distribution Center

The `kdc.conf` file contains KDC configuration information related to Kerberos tickets, realm-specific configurations, KDC database, and logging details

```
kdcdefaults]
kdc_ports = 88
[realms]
MYDOMAIN.COM = {
profile = /etc/krb5.conf
supported_enctypes = aes128-cts:normal des3-hmac-sha1:normal
arcfourhmac:
normal des-hmac-sha1:normal des-cbc-md5:normal des-cbc-crc:normal
des-cbc-crc:v4 des-cbc-crc:afs3
allow-null-ticket-addresses = true
database_name = /usr/local/var/krb5kdc/principal
acl_file = /usr/local/var/krb5kdc/kadm5.acl
admin_database_lockfile = /usr/local/var/krb5kd/kadm5_adb.lock
admin_keytab = FILE:/usr/local/var/krb5kdc/kadm5.keytab
key_stash_file = /usr/local/var/krb5kdc/.k5stash
kdc_ports = 88
kadmind_port = 749
max_life = 2d 0h 0m 0s
max_renewable_life = 7d 0h 0m 0s
}
```

Configuring the Key Distribution Center



The following table shows details of the sections that are found in the `kdc.conf` file:

Sr no	Property	Description
1	`kdcdefaults`	This section contains the default values used for authentication
2	`realms`	This section contains a separate subsections for every Kerberos realm
3	`dbdefaults`	This section contains the default database configurations used by KDC for storing the principals
4	`dbmodules`	This section contains the details for each of the database modules based on the type of database supported
5	`logging`	This section provides the logging configurations for every Kerberos daemon

Establishing the KDC database

- KDC stores the credentials for each of the users in the KDC database.
- The database can be a file or an LDAP store.
- To configure a file-based KDC database, we run the following commands using the realm name:

```
kdb5_util create -r MYREALM.COM -s
```

- This command creates five files in `/usr/local/var/krb5kdc` folder:
  - The Kerberos database files: `principal` and `principal.ok`
  - The Kerberos administrative database file: `principal.kadm5`
  - The administrative database lock file: `principal.kadm5.lock`
  - The stash file: `.k5.MYREALM.COM.COM`
- The stash file is a local copy of the encrypted master key that resides on the KDC's local disk.
- The stash file is used to automatically authenticate the KDC itself before starting the Kerberos daemons.

Setting up the administrator principal for KDC

- Once the KDC database is created, the administrator principal should be configured in the database.
- To do this, first add the administrator principal in the `/var/Kerberos/krb5kdc/kadm.acl` file that contains the **access control list** (**ACL**) that is used by the `kadmind` daemon to manage the Kerberos database access.
- A typical `kadm.acl` file that provides all administrators with full privilege will have the following entry:

```
*/admin@MYREALM.COM
```

Starting the Kerberos daemons

- Once the configurations are set properly, we are ready to start the Kerberos daemons.
- If Kerberos is installed through `yum` or `apt-get`, the kadmin and KDC server daemon can be started using the following command:

```
service kadmin start
/sbin/service krb5kdc start
```

- Krb5kdc is the KDC server, while the kadmin daemon enables administrators to connect from remote machines and perform Kerberos (KDC) administration using the kadmin client.

Other Configurations

Setting up the first Kerberos administrator

- ❑ Configure the principal password in the KDC database using the `kadmin.local` command on the master KDC server.
- ❑ Run the following command to set up the administrator principal and provide the password for the administrator principal.

```
kadmin.local -p admin/admin
```

- ❑ The `kinit` command is used to authenticate the user with KDC. We can verify the administrator authentication using `kinit` to ensure that KDC is able to authenticate the users.

```
kinit admin@MYREALM.COM
```

Adding the user or service principals

- ❑ After the admin user setup is completed and the Kerberos daemons have started, then we can add the principals to the Kerberos database using the `kadmin` utility.

```
add_principal -randkey user/mydomain.com@MYREALM.COM
```

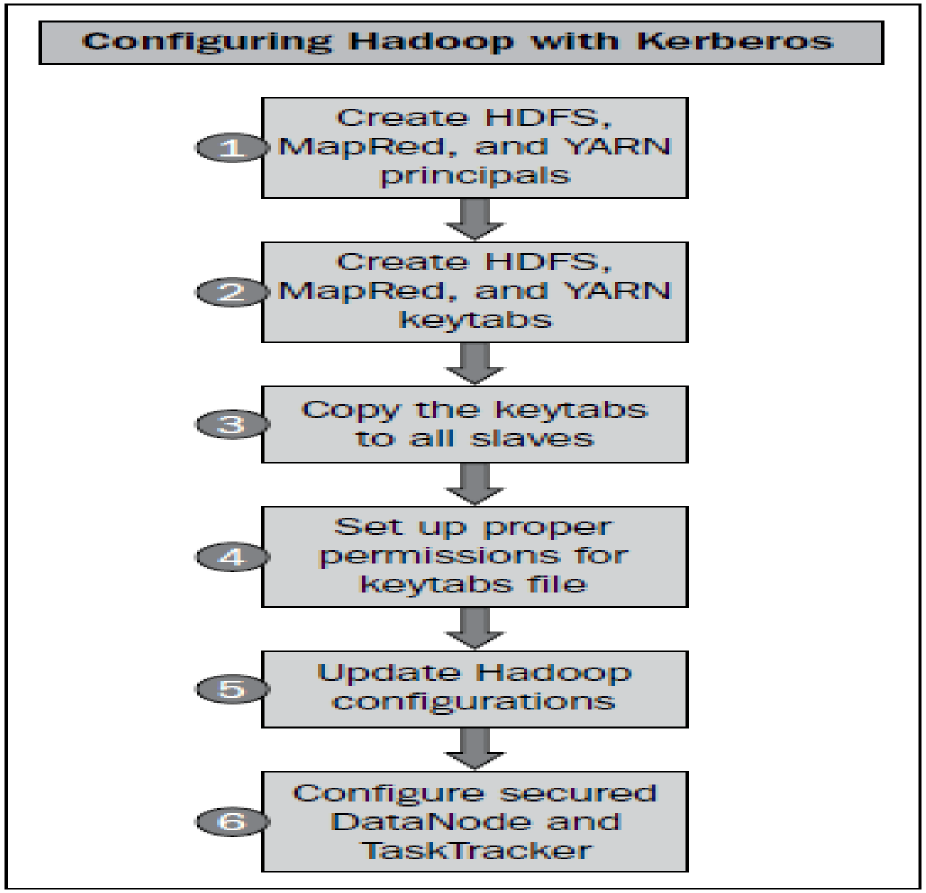
Configuring LDAP as the Kerberos database

- ❑ Next we can add the principals to the Kerberos database using the `kadmin` utility.

```
add_principal -randkey user/mydomain.com@MYREALM.COM
```

Configuring Hadoop with Kerberos authentication

- Once the Kerberos setup is completed and the user principals are added to KDC, we can configure Hadoop to use Kerberos authentication.
- It is assumed that a Hadoop cluster in a non-secured mode is configured and available.
- We will begin the configuration using Cloudera Distribution of Hadoop (**CDH4**).


Configuring Hadoop with Kerberos
1 Create HDFS, MapRed, and YARN principals
2 Create HDFS, MapRed, and YARN keytabs
3 Copy the keytabs to all slaves
4 Set up proper permissions for keytabs file
5 Update Hadoop configurations
6 Configure secured DataNode and TaskTracker

Setting up the Kerberos client on all Hadoop nodes

- ❑ In each of the Hadoop node (master node and slave node), we need to install the Kerberos client.
- ❑ This is done by installing the client packages and libraries on the Hadoop nodes.
- ❑ For RHEL/CentOS/Fedora, we will use the following command:

```
yum install krb5-libs krb5-workstation
```

- ❑ For Ubuntu, we will use the following command:

```
apt-get install krb5-user
```

Setting up Hadoop service principals

- In CDH4, there are three users (`hdfs`, `mapred`, and `yarn`) that are used to run the various Hadoop daemons.
- All the **Hadoop Distributed File System** (**HDFS**)-related daemons such as NameNode, DataNode, and Secondary NameNode are run under the `hdfs` user, while for MRV1, the MapReduce-related daemons such as JobTracker and TaskTracker run using the `mapred` user.
- For MRV2, the `yarn` user runs ResourceManager and NodeManager, while the `mapred` user runs the JobHistory server and the MapReduce application.
- need to create the `hdfs`, `mapred`, and `yarn` principals in KDC to ensure Kerberos authentication for the Hadoop daemons.
- use the following `kadmin` commands to create these principals:

```
kadmin
kadmin: addprinc -randkey hdfs/mydomain.com@MYREALM.COM
kadmin: addprinc -randkey mapred/mydomain.com@MYREALM.COM
kadmin: addprinc -randkey http/mydomain.com@MYREALM.COM
kadmin: addprinc -randkey yarn/mydomain.com@MYREALM.COM
```

Creating a keytab file for the Hadoop services

- A **`keytab`** is a file containing pairs of Kerberos principals and encrypted keys derived from the Kerberos password.
- This file is used for headless authentication with KDC when the services run in the background without human intervention.
- The `keytab` file is created using the `kadmin` commands.
- Need to create the `keytab` file for the `hdfs` and `mapred` users.
- We also need to add the `http` principal to these keytabs, so that the Web UI associated with Hadoop are authenticated using Kerberos.

```
kadmin: xst -norandkey -k hdfs.keytab
hdfs/mydomain.com@MYREALM.COM http/
mydomain.com@MYREALM.COM

kadmin: xst -norandkey -k mapred.keytab
hdfs/mydomain.com@MYREALM.COM
http/mydomain.com@MYREALM.COM

kadmin: xst -norandkey -k yarn.keytab
hdfs/mydomain.com@MYREALM.COM http/
mydomain.com@MYREALM.COM
```

Distributing the keytab file for all the slaves

- Once the `keytab` file is created, it has to move to the `/etc/hadoop/conf` folder.
- `Keytab` file has to be secured and only owner of keytab can see this file.
- For this, the `hdfs` and `mapred` owner of the `keytab` file is changed, and the file permission is changed to `400`.
- The service principals for `hdfs`, `mapred`, and `http` has a fully qualified domain name associated with the username.
- Service principal is host-specific and is unique for each of nodes in cluster.
- Move the `keytab` file to the `conf` folder and secure it

```
$sudo mv hdfs.keytab mapred.keytab /etc/hadoop/conf/
$sudo chown hdfs:hadoop /etc/hadoop/conf/hdfs.keytab
$sudo chown mapred:hadoop /etc/hadoop/conf/mapred.keytab
$sudo chmod 400 /etc/hadoop/conf/hdfs.keytab
$sudo chmod 400 /etc/hadoop/conf/mapred.keytab
```

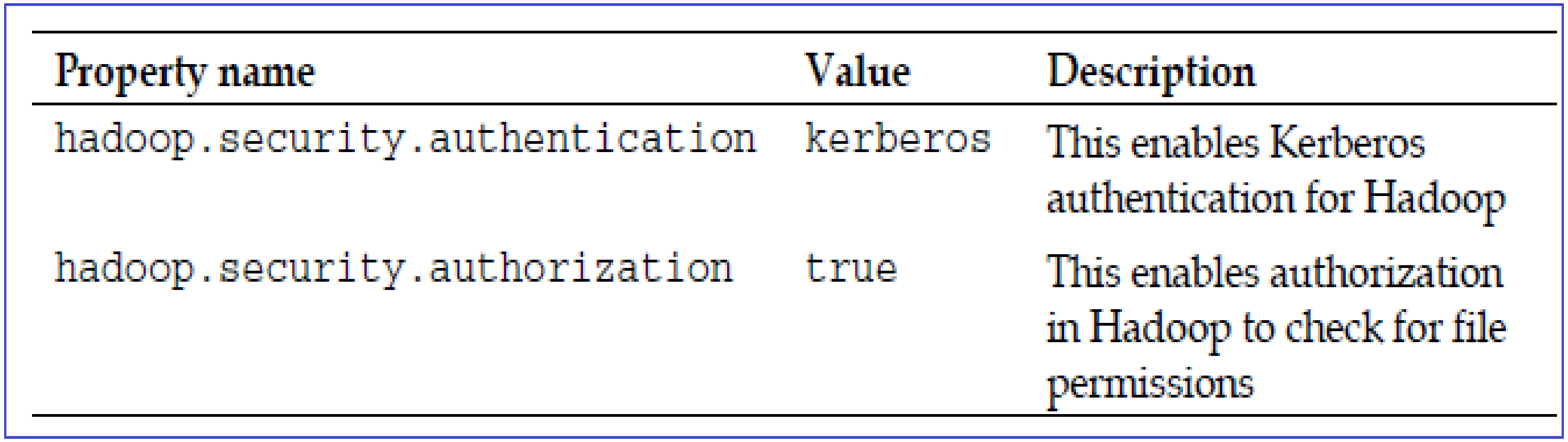
- The `keytab` file should be created specific to each node in the cluster.
- Distributing and managing the `keytab` file in a large cluster is time consuming and error prone. So it is better to use deployment tools and automate this deployment.

Setting up Hadoop configuration files

- ❑ Next, we update the Hadoop configuration files to enable Kerberos authentication.
- ❑ Before updating the Hadoop configuration file, we should shutdown the cluster.

HDFS-related configurations

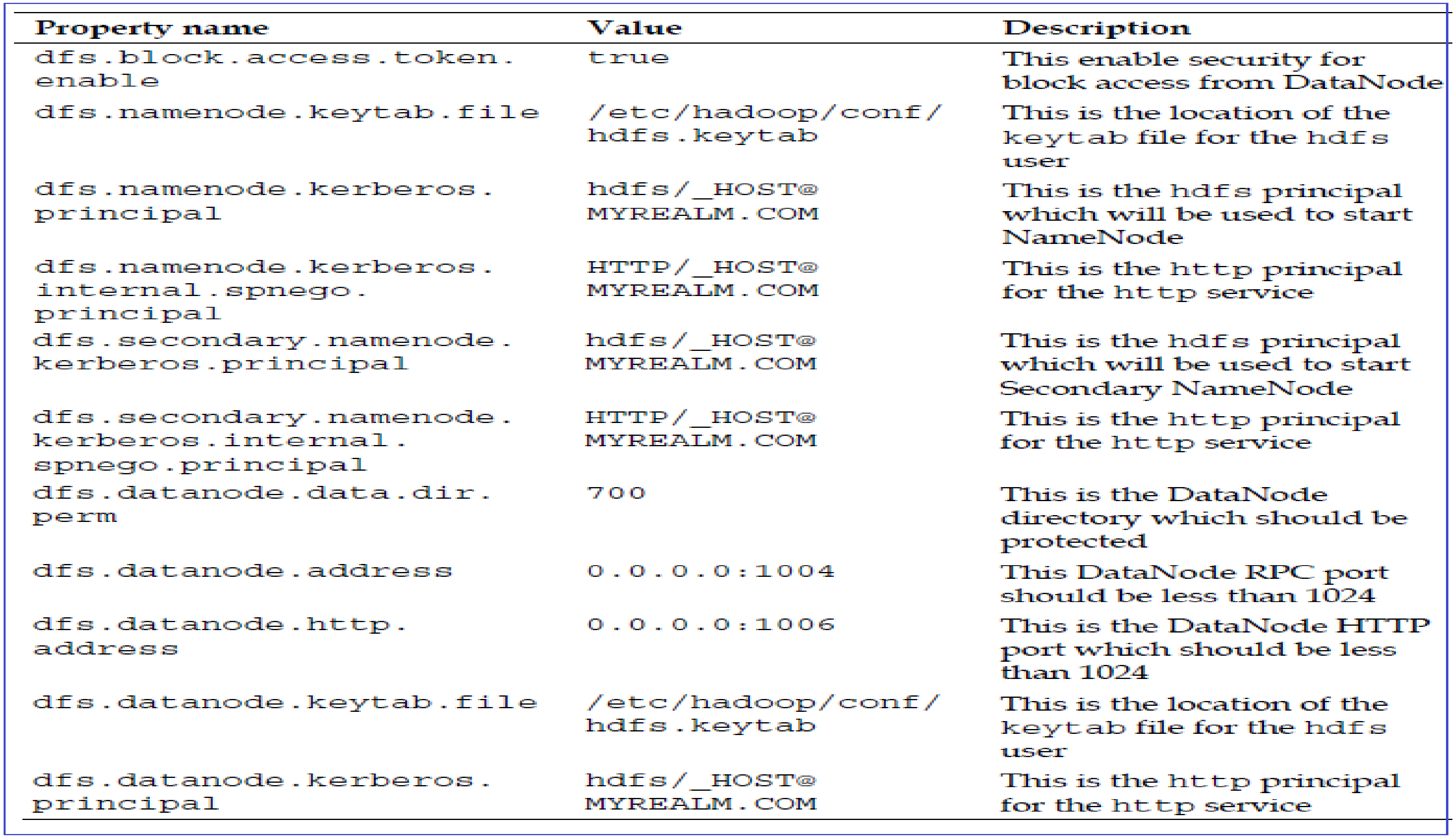
- Few properties should be updated in the `core-site.xml` file in the `/etc/hadoop/conf` folder.
- These properties enable Kerberos authentication and user authorization within the Hadoop cluster



Property name	Value	Description
`hadoop.security.authentication`	`kerberos`	This enables Kerberos authentication for Hadoop
`hadoop.security.authorization`	`true`	This enables authorization in Hadoop to check for file permissions

HDFS-related configurations

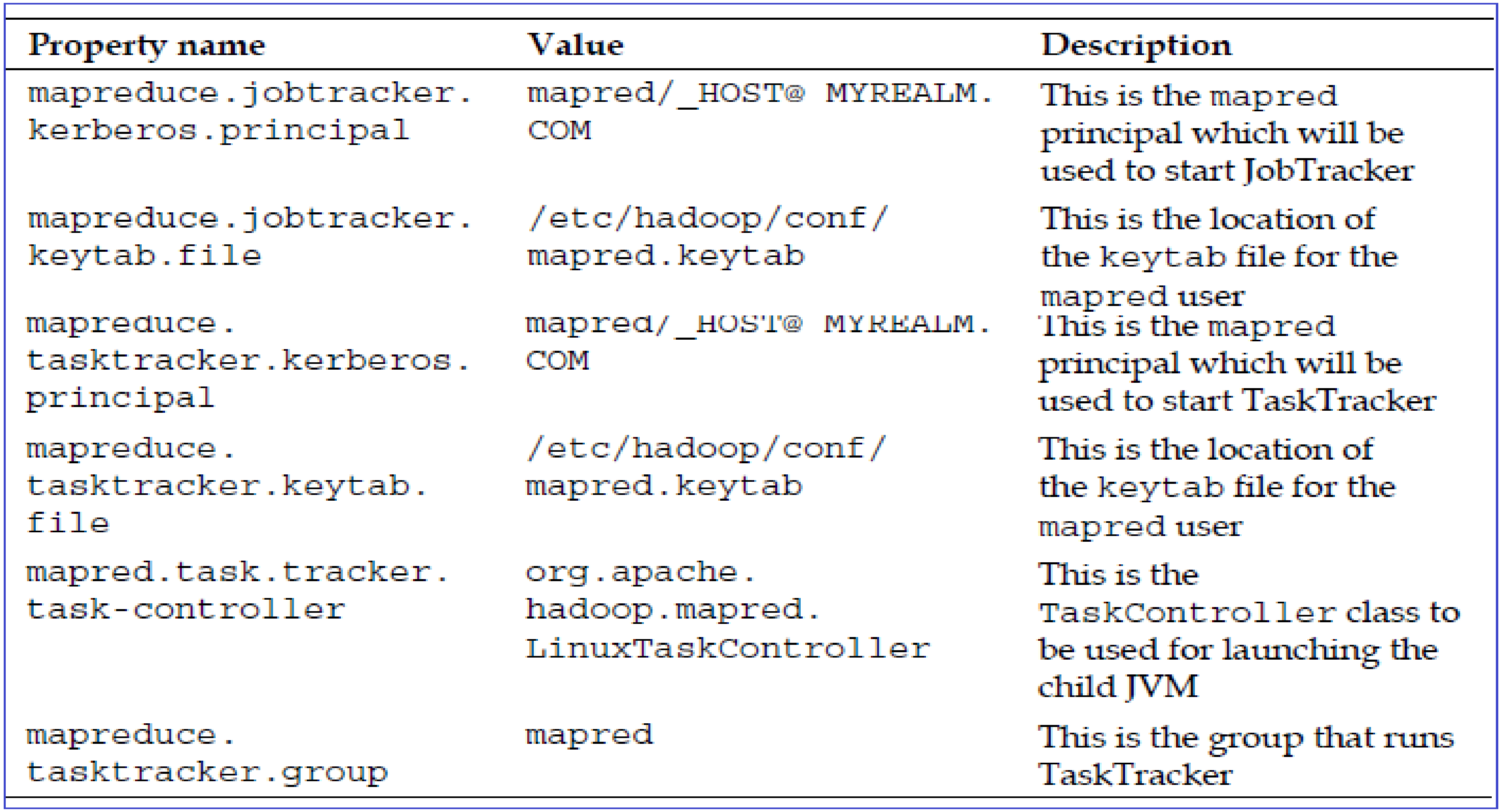
The following properties should be updated in the `hdfs-site.xml`



Property name	Value	Description
dfs.block.access.token.enable	true	This enable security for block access from DataNode
dfs.namenode.keytab.file	/etc/hadoop/conf/hdfs.keytab	This is the location of the keytab file for the hdfs user
dfs.namenode.kerberos.principal	hdfs/_HOST@MYREALM.COM	This is the hdfs principal which will be used to start NameNode
dfs.namenode.kerberos.internal.spnego.principal	HTTP/_HOST@MYREALM.COM	This is the http principal for the http service
dfs.secondary.namenode.kerberos.principal	hdfs/_HOST@MYREALM.COM	This is the hdfs principal which will be used to start Secondary NameNode
dfs.secondary.namenode.kerberos.internal.spnego.principal	HTTP/_HOST@MYREALM.COM	This is the http principal for the http service
dfs.datanode.data.dir.perm	700	This is the DataNode directory which should be protected
dfs.datanode.address	0.0.0.0:1004	This DataNode RPC port should be less than 1024
dfs.datanode.http.address	0.0.0.0:1006	This is the DataNode HTTP port which should be less than 1024
dfs.datanode.keytab.file	/etc/hadoop/conf/hdfs.keytab	This is the location of the keytab file for the hdfs user
dfs.datanode.kerberos.principal	hdfs/_HOST@MYREALM.COM	This is the http principal for the http service

MRV1-related configurations

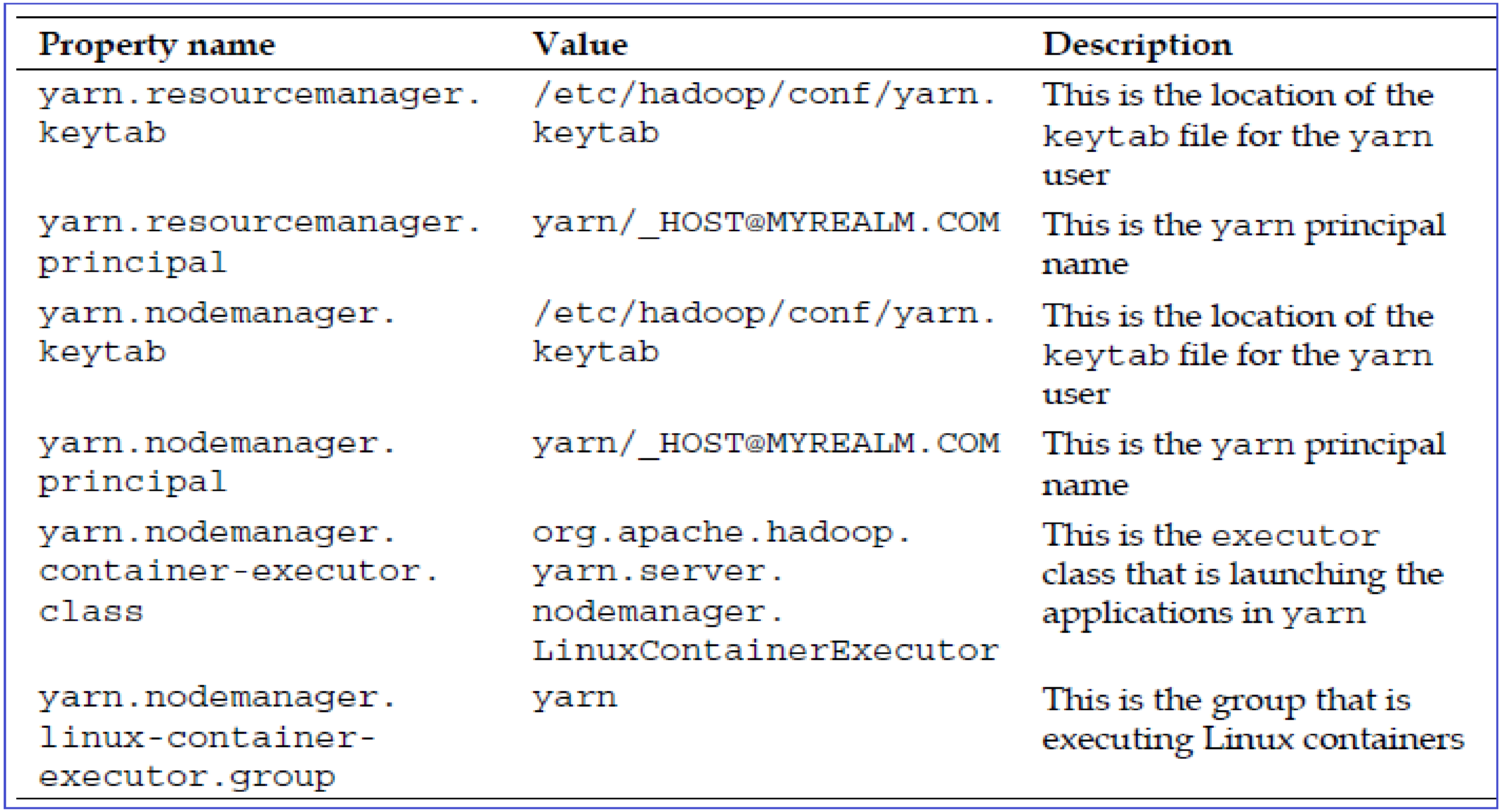
- For MRV1, `mapred-site.xml` file should be configured for securing Hadoop.



Property name	Value	Description
`mapreduce.jobtracker.kerberos.principal`	`mapred/_HOST@ MYREALM.COM`	This is the mapred principal which will be used to start JobTracker
`mapreduce.jobtracker.keytab.file`	`/etc/hadoop/conf/mapred.keytab`	This is the location of the keytab file for the mapred user
`mapreduce.tasktracker.kerberos.principal`	`mapred/_HOST@ MYREALM.COM`	This is the mapred principal which will be used to start TaskTracker
`mapreduce.tasktracker.keytab.file`	`/etc/hadoop/conf/mapred.keytab`	This is the location of the keytab file for the mapred user
`mapred.task.tracker.task-controller`	`org.apache.hadoop.mapred.LinuxTaskController`	This is the `TaskController` class to be used for launching the child JVM
`mapreduce.tasktracker.group`	`mapred`	This is the group that runs TaskTracker

MRV2-related configurations

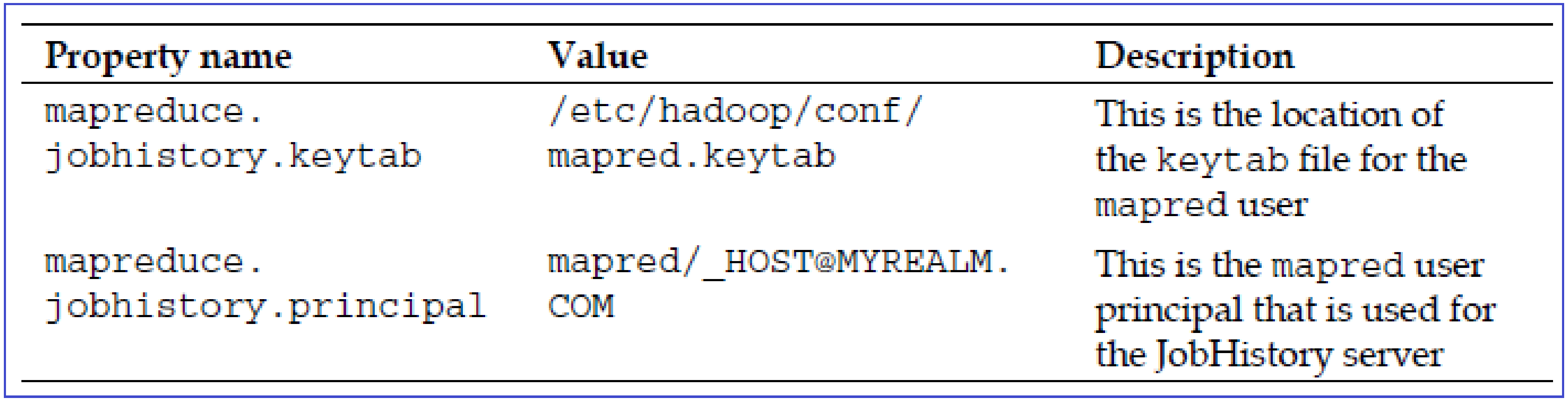
For MRV2, the `yarn-site.xml` file should be configured for specifying the location of the `keytab` file of the `yarn` user for ResourceManager and NodeManager.



Property name	Value	Description
`yarn.resourcemanager.keytab`	`/etc/hadoop/conf/yarn.keytab`	This is the location of the keytab file for the yarn user
`yarn.resourcemanager.principal`	`yarn/_HOST@MYREALM.COM`	This is the yarn principal name
`yarn.nodemanager.keytab`	`/etc/hadoop/conf/yarn.keytab`	This is the location of the keytab file for the yarn user
`yarn.nodemanager.principal`	`yarn/_HOST@MYREALM.COM`	This is the yarn principal name
`yarn.nodemanager.container-executor.class`	`org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor`	This is the executor class that is launching the applications in yarn
`yarn.nodemanager.linux-container-executor.group`	`yarn`	This is the group that is executing Linux containers

MRV2-related configurations

- ❑ The `mapred-site.xml` file should be configured with the `keytab` file location of the job history server.
- ❑ This configuration file should be present in all nodes of the cluster.
- ❑ Each user running the `yarn` jobs should be configured in each of the nodes on the cluster.



Property name	Value	Description
`mapreduce.jobhistory.keytab`	`/etc/hadoop/conf/mapred.keytab`	This is the location of the keytab file for the `mapred` user
`mapreduce.jobhistory.principal`	`mapred/_HOST@MYREALM.COM`	This is the `mapred` user principal that is used for the JobHistory server

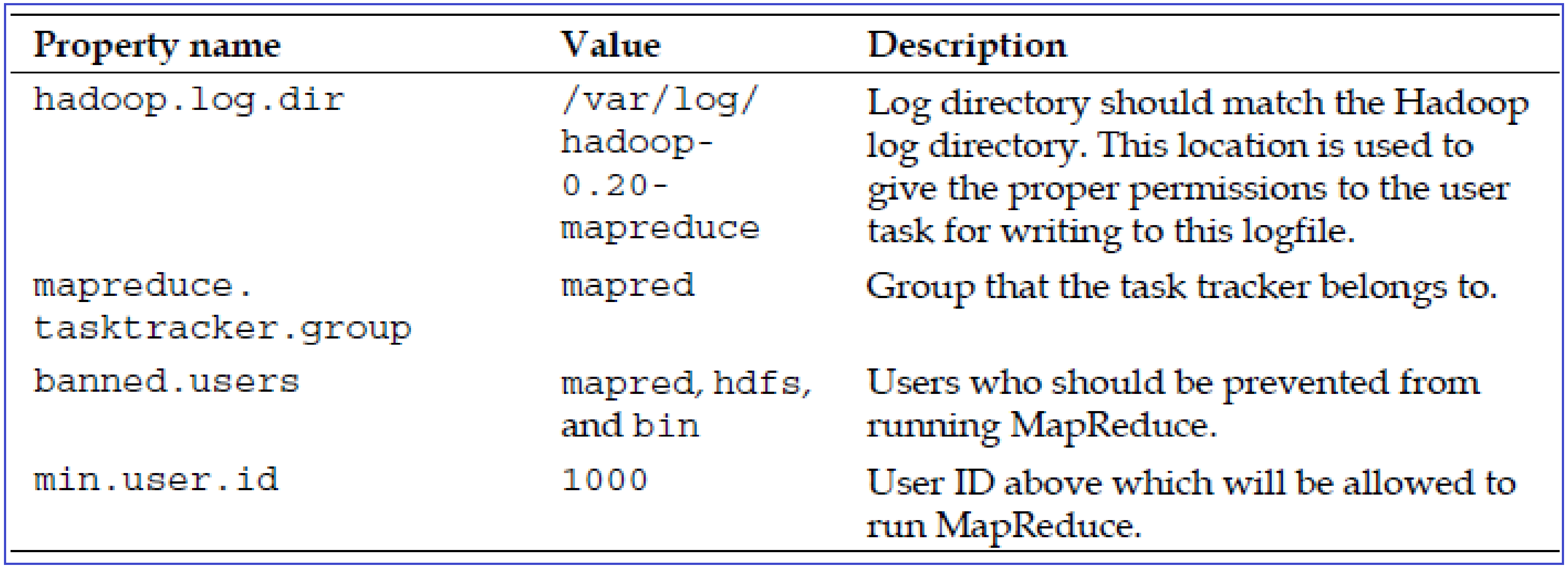
Setting up secured DataNode

- Once the Hadoop configuration files are updated, we move to DataNode and ensure it is secured.
- To do this, we need to start DataNode in a secure mode.
- **Jsvc** is a set of libraries and applications for making Java applications run on Unix more easily.
- Jsvc allows the application (for example, Tomcat) to perform some privileged operations as root and then switch to a non-privileged user.
- This program helps DataNode to bind on ports less than 1024 and then run with the `hdfs` user.
- The following configurations should be set in each of DataNodes in the `/etc/default/hadoophdfs-datanode` folder to DataNode can run in a secure mode:

```
export HADOOP_SECURE_DN_USER=hdfs
export HADOOP_SECURE_DN_PID_DIR=/var/lib/hadoop-hdfs
export HADOOP_SECURE_DN_LOG_DIR=/var/log/hadoop-hdfs
export JSVC_HOME=/usr/lib/bigtop-utils/ or /usr/libexec/bigtop-
utils
 (based on the OS the corresponding variable has to be set)
```

Setting up the TaskController class

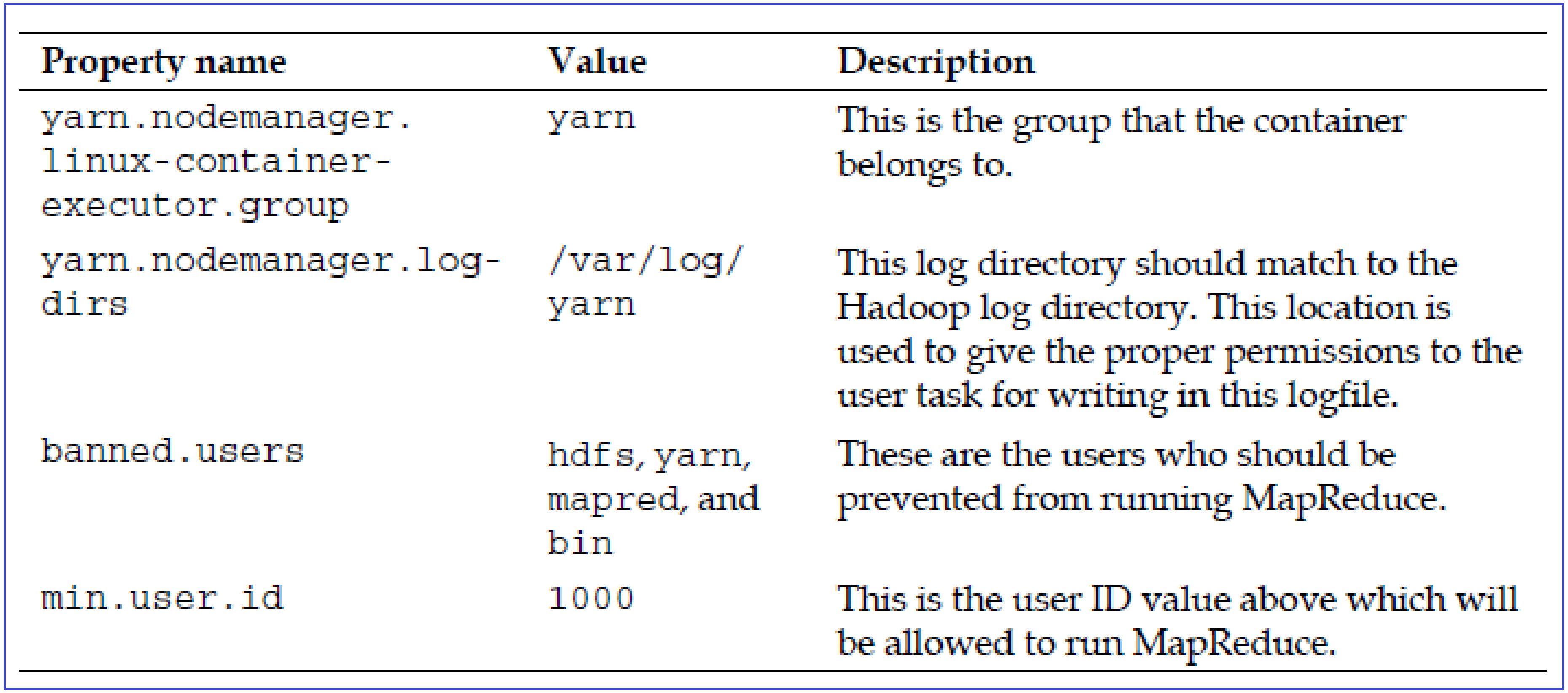
- For MRV1, the `TaskController` class in the Hadoop framework defines how users map and reduce tasks are launched and controlled.
- For a secured Hadoop cluster, we need to ensure that the user who launched the MapReduce program is running TaskNode.
- So we need all the users who run the MapReduce program to be defined on all the task nodes.
- This configuration file should have following configurations (`task-controller.cfg`):



Property name	Value	Description
`hadoop.log.dir`	`/var/log/hadoop-0.20-mapreduce`	Log directory should match the Hadoop log directory. This location is used to give the proper permissions to the user task for writing to this logfile.
`mapreduce.tasktracker.group`	`mapred`	Group that the task tracker belongs to.
`banned.users`	`mapred`, `hdfs`, and `bin`	Users who should be prevented from running MapReduce.
`min.user.id`	`1000`	User ID above which will be allowed to run MapReduce.

Setting up the TaskController class

For MRV2, similar to `task-controller.cfg`, we need to define `containerexecutor.cfg` with the following configurations:



Property name	Value	Description
`yarn.nodemanager.linux-container-executor.group`	`yarn`	This is the group that the container belongs to.
`yarn.nodemanager.log-dirs`	`/var/log/yarn`	This log directory should match to the Hadoop log directory. This location is used to give the proper permissions to the user task for writing in this logfile.
`banned.users`	`hdfs`, `yarn`, `mapred`, and `bin`	These are the users who should be prevented from running MapReduce.
`min.user.id`	`1000`	This is the user ID value above which will be allowed to run MapReduce.

Setting up the TaskController class

- We can then start all the Hadoop daemons using the following commands:

```
sudo service hadoop-hdfs-namenode start
sudo service hadoop-hdfs-datanode start
sudo service hadoop-hdfs-secondarynamenode start
```

- For MRV1:

```
sudo service hadoop-0.20-mapreduce-jobtracker start
```

- For MRV2:

```
sudo service hadoop-yarn-resourcemanager start
sudo service hadoop-yarn-nodemanager start
sudo service hadoop-mapreduce-historyserver start
```

Configuring users for Hadoop

- ❑ All users required to run MapReduce jobs on the cluster need to be set up all the nodes in the cluster.
- ❑ In a large cluster, setting up these users will be very time consuming.
- ❑ Best practice is to integrate the existing enterprise users in Active Directory or LDAP using cross-realm authentication in Kerberos.
- ❑ Users are centrally managed in Active Directory or LDAP, and we set up a one-way cross-realm trust between Active Directory/LDAP and KDC on the cluster.
- ❑ Thus, the Hadoop service principal doesn't have to be set up in Active Directory/LDAP, and they authenticate locally on the cluster with KDC.
- ❑ This also ensures that the cluster load is isolated from the rest of the enterprise.

Automation of a secured Hadoop deployment

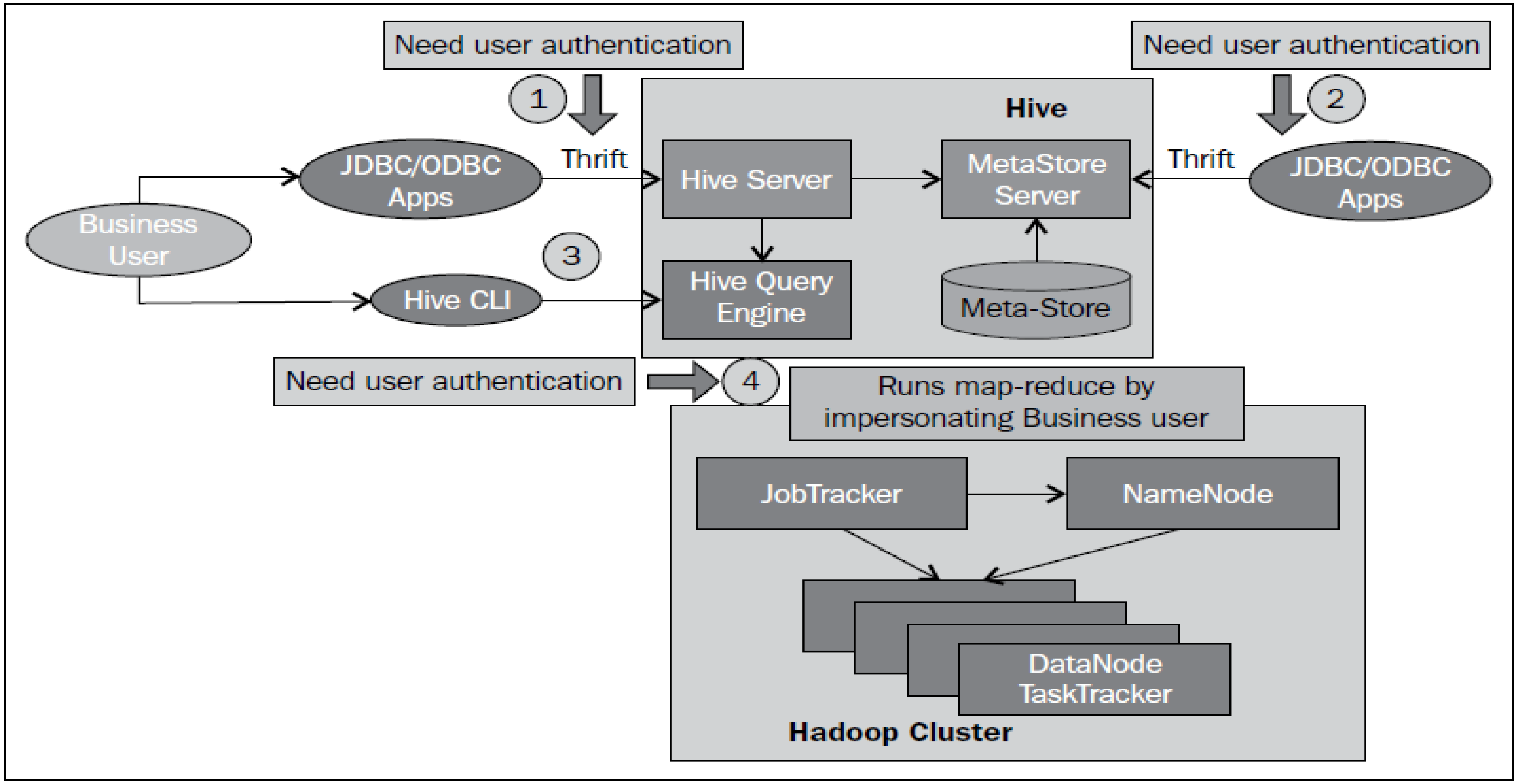
- Managing and configuring large cluster is not done manually as it is laborious and error prone.
- Traditionally, enterprises used Chef/Puppet or a similar solution for cluster configuration management and deployment.
- In this approach, organizations had to continuously update their chef recipes based on the changes in Apache Hadoop releases.
- Instead, organizations typically deploy Hadoop cluster deployment automation based on the Hadoop distribution they work with.
- For example, in a Cloudera-based Hadoop distribution, organizations leverages **Cloudera Manager** to provide cluster deployment, automation, and management capability.
- For Hortonworks-based distributions, organizations prefer Ambari.
- Similarly, Intel distribution has **Intel Manager** for Apache Hadoop.
- Each of these deployment managers support secured Hadoop deployment.
- The approach and details to configure the security remains the same; however, these tools provide the automation required for seamless deployment of the secured Hadoop cluster.

Securing the Hadoop Ecosystem

- Securing Hadoop in Big Data journey, is only half done.
- The Hadoop ecosystem consists of various components such as Hive, Oozie, and HBase.
- We need to secure all the other Hadoop ecosystem components.
- Need to understand security challenges for Hadoop components, and set up secured authentication and user authorization for each of them.
- Each ecosystem component has its own security challenges and needs to be configured uniquely based on its architecture to secure them.
- Each of these ecosystem components has end users directly accessing the component or a backend service accessing the Hadoop core components (HDFS and MapReduce).

Securing Hive

- Hive provides the ability to run SQL queries over the data stored in the HDFS.
- Hive provides the Hive query engine that converts Hive queries provided by the user to a pipeline of MapReduce
- Results of MapReduce executions are then presented back to the user or stored in HDFS.



Securing Hive

To secure Hive in Hadoop ecosystem, following interactions should be secured:

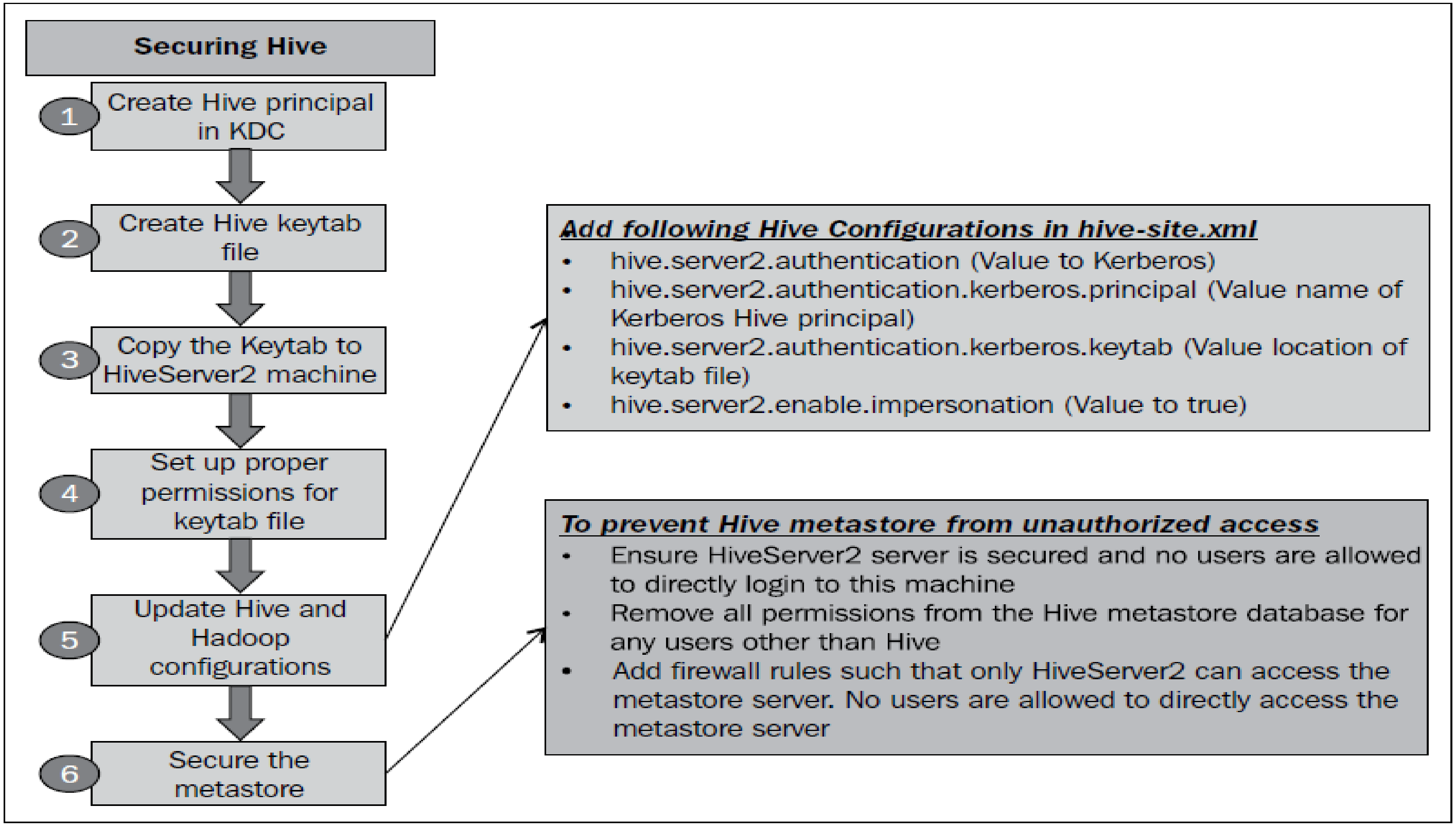
- ❑ User interaction with Hive CLI or HiveServer
- ❑ User roles and privileges needs to be enforced to ensure users have access to only authorized data
- ❑ The interaction between Hive and Hadoop (JobTracker or ResourceManager) has to be secured and the user roles and privileges should be propagated to Hadoop jobs

Securing Hive

- When we use Kerberos authentication for Hive users with HiveServer2, the same user is impersonated to execute MapReduce on the Hadoop cluster.
- So it is recommended that in a production environment, we configure HiveServer2 with Kerberos to have a seamless authentication and access control for the users submitting Hive queries.
- The credential store for Kerberos KDC can be configured to be LDAP so that we can centrally manage the user credentials of the end users.

Securing Hive

❑To set up a secured Hive interactions, we need to do following



Securing Hive
1 Create Hive principal in KDC
2 Create Hive keytab file
3 Copy the Keytab to HiveServer2 machine
4 Set up proper permissions for keytab file
5 Update Hive and Hadoop configurations
6 Secure the metastore

Add following Hive Configurations in hive-site.xml

- hive.server2.authentication (Value to Kerberos)
- hive.server2.authentication.kerberos.principal (Value name of Kerberos Hive principal)
- hive.server2.authentication.kerberos.keytab (Value location of keytab file)
- hive.server2.enable.impersonation (Value to true)

To prevent Hive metastore from unauthorized access

- Ensure HiveServer2 server is secured and no users are allowed to directly login to this machine
- Remove all permissions from the Hive metastore database for any users other than Hive
- Add firewall rules such that only HiveServer2 can access the metastore server. No users are allowed to directly access the metastore server

Securing Hive

- One of the key steps in securing Hive interaction is to ensure that the Hive user is impersonated in Hadoop
- To achieve this goal, we need to add the `hive.server2.enable.impersonation` configuration in `hive-site.xml`, and `hadoop.proxyuser.hive.hosts` and `hadoop.proxyuser.hive.groups` in `core-site.xml`.

```
<property>
<name>hive.server2.authentication</name>
<value>KERBEROS</value>
</property>
<property>
<name>hive.server2.authentication.kerberos.principal</name>
<value>hive/_HOST@YOUR-REALM.COM</value>
</property>
<property>
<name>hive.server2.authentication.kerberos.keytab</name>
<value>/etc/hive/conf/hive.keytab</value>
</property>
<property>
<name>hive.server2.enable.impersonation</name>
<description>Enable user impersonation for
HiveServer2</description>
<value>true</value>
</property>
```

Securing Hive using Sentry

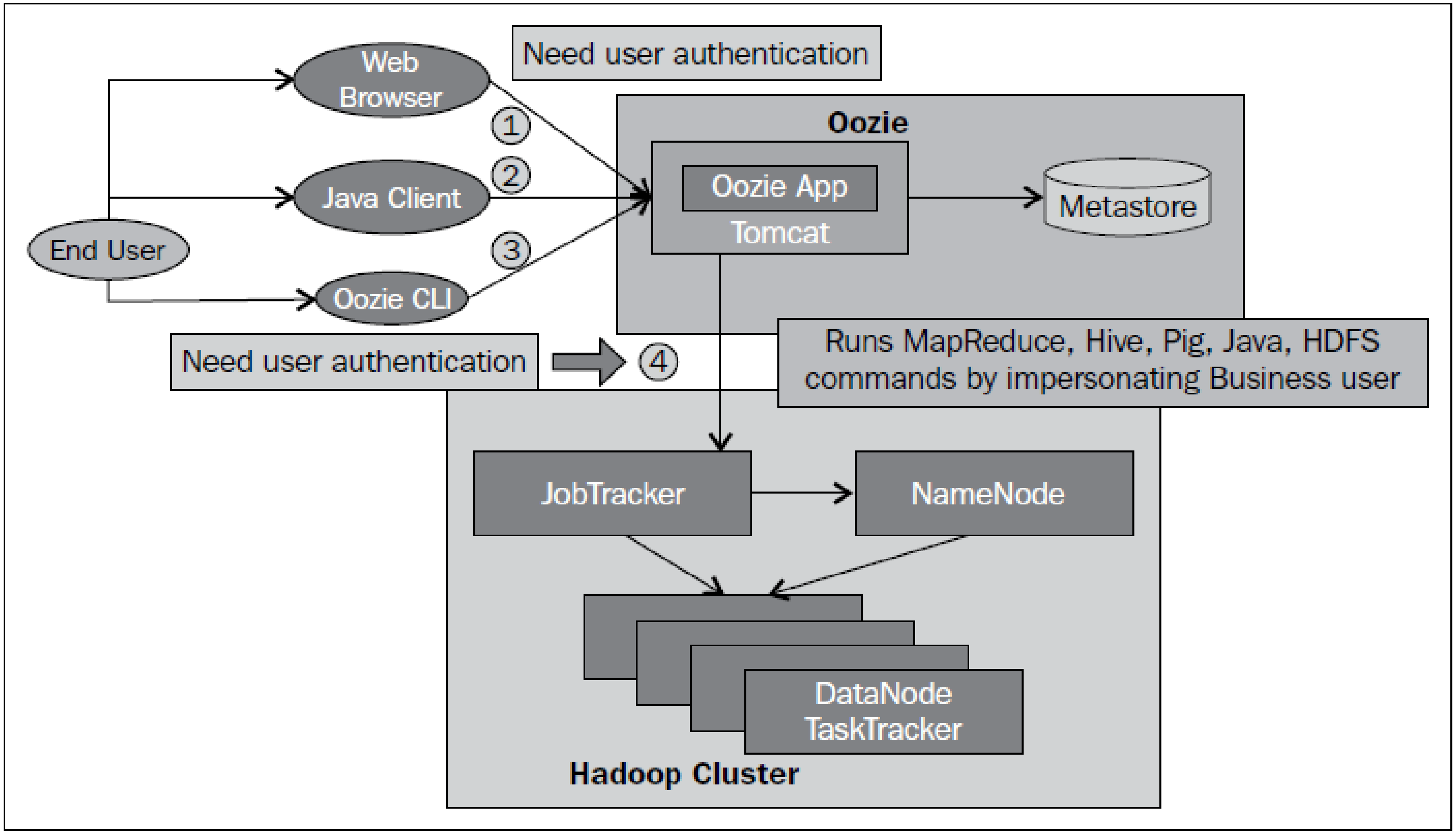
- **Sentry** is the one of the latest entrant in the Hadoop ecosystem that provides finegrained user authorization for the data that is stored in Hive.
- Sentry provides finegrained, role-based authorization to Hive and **Impala**.
- Sentry uses HiveServer2 and metastore server to execute the queries on the Hadoop platform.
- However, the user impersonation is turned off in HiveServer2 when Sentry is used.
- Sentry enforces user privileges on the Hadoop data using the Hive metastore.
- Sentry supports authorization policies per database/schema.

Securing Oozie

- ❑Usually in a production system, there is a need to run a series of heterogeneous tasks consisting of Pig, Hive, MapReduce, and so on in some predefined sequence.
- ❑ To achieve this we use Oozie.
- ❑ Oozie is a workflow orchestrator and scheduler for the Hadoop ecosystem.
- ❑Oozie takes an input XML configuration file that contains information about the sequence and dependency from the user, and executes a workflow of jobs that may contain Hive, Pig, MapReduce, Java, or shell scripts.

Securing Oozie

Figure shows interaction of a user executing tasks on Oozie:

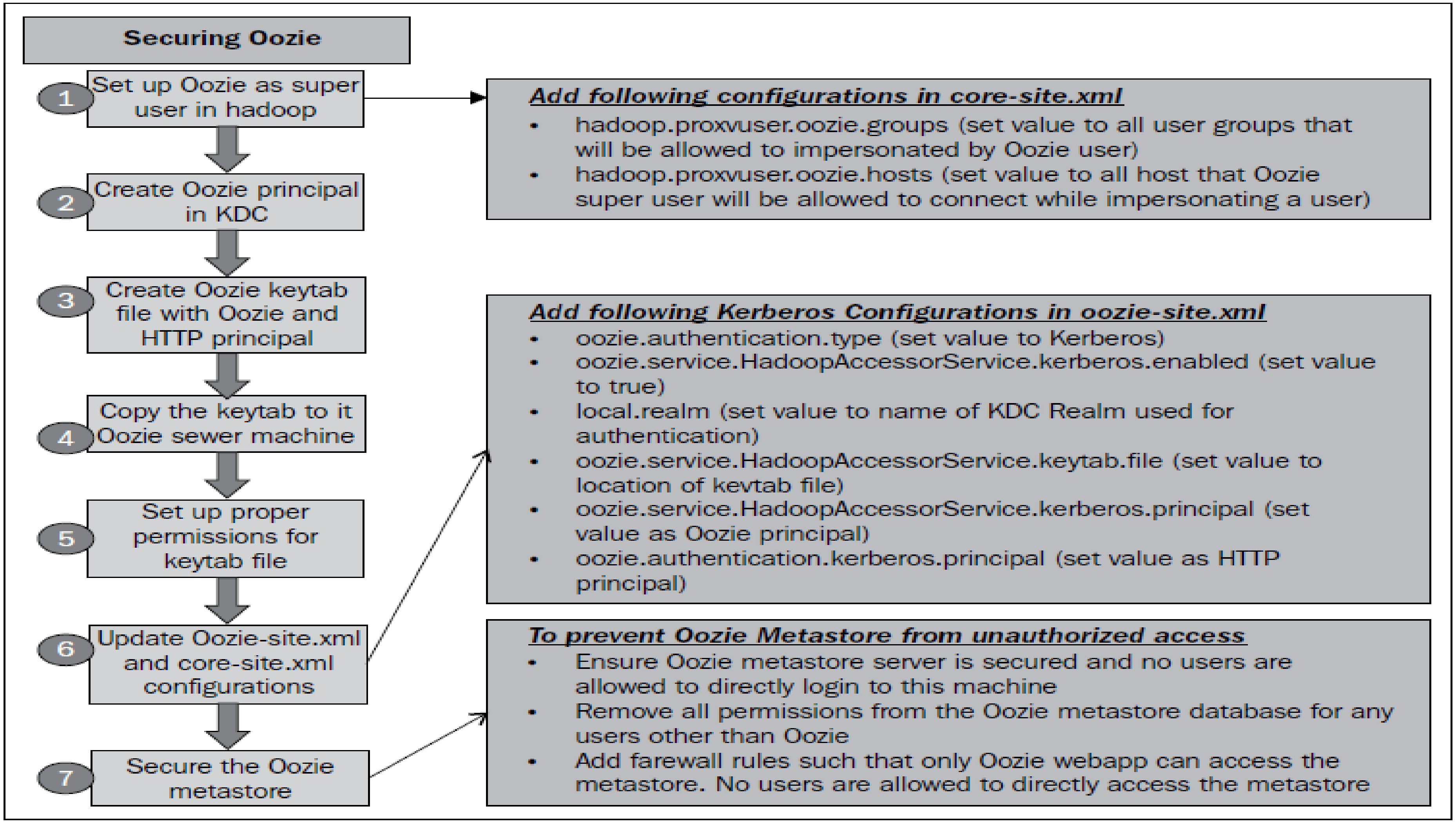


Securing Oozie

- To secure Oozie, we need to secure two different interactions.
- First we look at the approach to authenticate end users while accessing Oozie securely.
- Next we look at how the Oozie server is going to run jobs on a secured Hadoop cluster.
- To ensure secure end users are accessing Oozie web applications, Oozie provides user authentication to the Oozie web services. Oozie supports HTTPS (SSL)-base encryption between Oozie server and browser.
- Oozie also provides Kerberos HTTP **Simple and Protected GSSAPI Negotiation Mechanism** (**SPNEGO**) authentication for web clients. SPNEGO protocol is used when a client application wants to authenticate to a remote server.
- For Oozie web clients, we need to set up SPNEGO authentication on the user browser.

Securing Oozie

Following figure steps are followed to set up a secured Oozie in the Hadoop cluster


Securing Oozie
1 Set up Oozie as super user in hadoop
2 Create Oozie principal in KDC
3 Create Oozie keytab file with Oozie and HTTP principal
4 Copy the keytab to it Oozie sewer machine
5 Set up proper permissions for keytab file
6 Update Oozie-site.xml and core-site.xml configurations
7 Secure the Oozie metastore

Add following configurations in core-site.xml

- hadoop.proxvuser.oozie.groups (set value to all user groups that will be allowed to impersonated by Oozie user)
- hadoop.proxvuser.oozie.hosts (set value to all host that Oozie super user will be allowed to connect while impersonating a user)

Add following Kerberos Configurations in oozie-site.xml

- oozie.authentication.type (set value to Kerberos)
- oozie.service.HadoopAccessorService.kerberos.enabled (set value to true)
- local.realm (set value to name of KDC Realm used for authentication)
- oozie.service.HadoopAccessorService.keytab.file (set value to location of kevtab file)
- oozie.service.HadoopAccessorService.kerberos.principal (set value as Oozie principal)
- oozie.authentication.kerberos.principal (set value as HTTP principal)

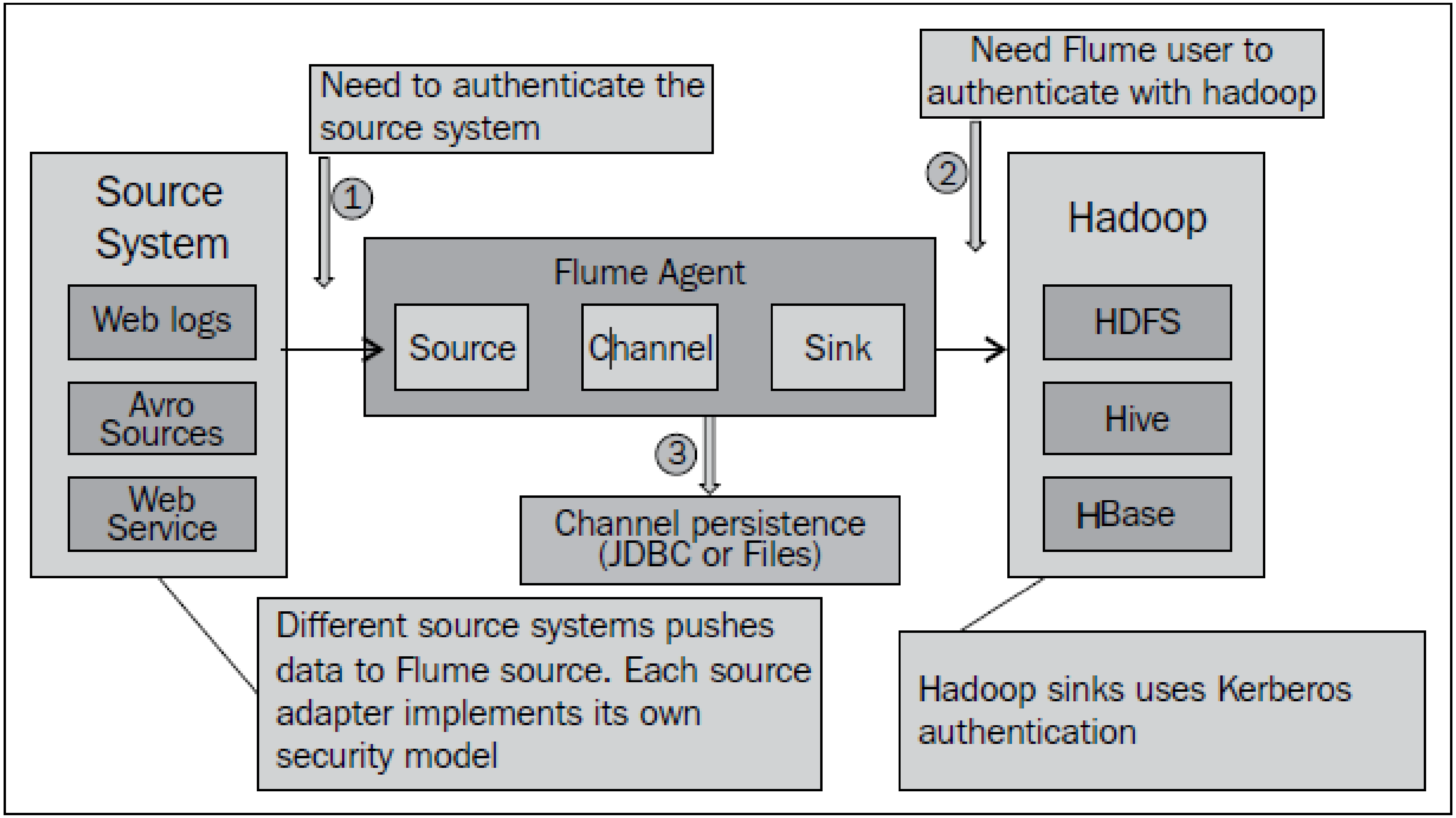
To prevent Oozie Metastore from unauthorized access

- Ensure Oozie metastore server is secured and no users are allowed to directly login to this machine
- Remove all permissions from the Oozie metastore database for any users other than Oozie
- Add farewall rules such that only Oozie webapp can access the metastore. No users are allowed to directly access the metastore

Securing Flume

- Flume is a distributed, reliable, near real-time ingestion component in the Hadoop ecosystem.
- Flume-ng is the latest version of Flume
- Flume has the concepts of **Source**, **Channel**, and **Sink** to ensure reliable communication.
- All these are embedded in a single **Flume Agent**.
- Source collects the events from the various source systems and pushes it to the channel.
- The channel is usually implemented using file, RDBMS, or memory.
- **Sink** pushes data to the target system.
- **Channel** removes the data once the sink is able to successfully deliver the event to the target system.
- Thus, there are multiple handshakes as data is ingested from multiple **Source System** through Flume.

Securing Flume

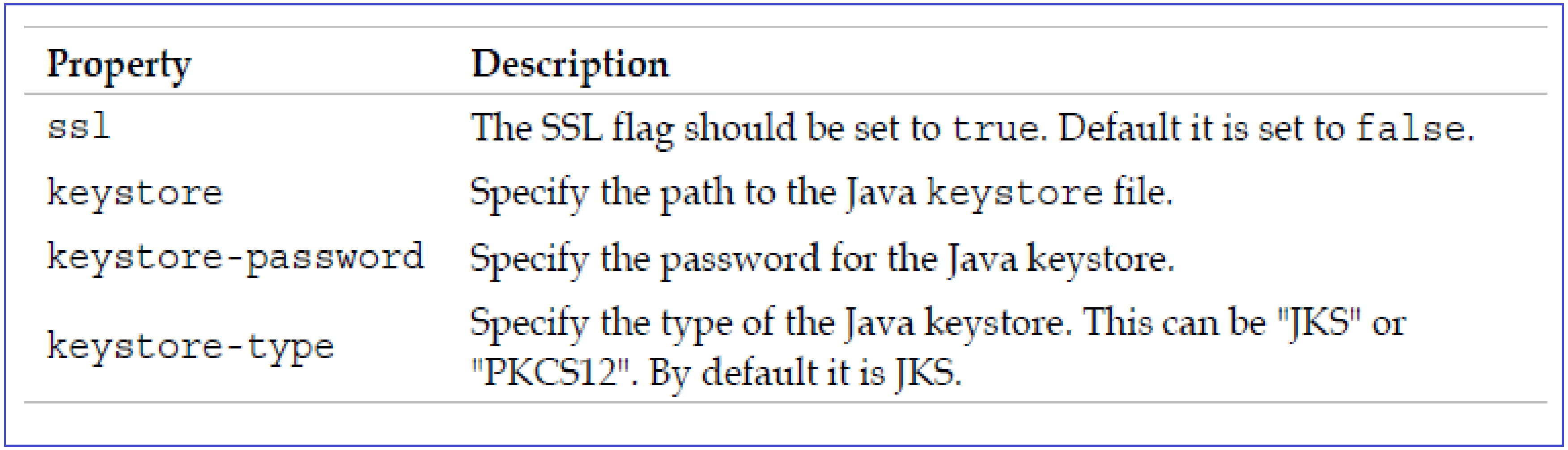

Need to authenticate the source system
Need Flume user to authenticate with hadoop
Source System
Web logs
Avro Sources
Web Service
1
2
3
Flume Agent
Source
Channel
Sink
Hadoop
HDFS
Hive
HBase
Channel persistence (JDBC or Files)
Different source systems pushes data to Flume source. Each source adapter implements its own security model
Hadoop sinks uses Kerberos authentication

Securing Flume

- ❑ To establish a secured Flume interaction with Hadoop, we need to
  - ➢ ensure that only authentic **Source System** are able to ingest data through Flume.
  - ➢ We also need to establish the trust between the **Flume Agent** and **Hadoop** to ensure that Flume is able to ingest the data into Hadoop on behalf of the source system process.
- ❑ Single **Flume Agent** will need to handle multiple source systems
- ❑ Need to secure intermediate data persisted in the **Channel persistence** layer
- ❑ There are usually three levels of security controls that need to be implemented.
  1. First, Flume should ensure that the source system processes are authenticated before they ingest data through Flume.
  2. Secondly, the communication between **Sink** and **Hadoop** is secured.
  3. Finally, we need to ensure that the intermediate data persisted in **Channel** is secured by enforcing proper authentication, authorization, and encryption of the data.

Securing Flume sources

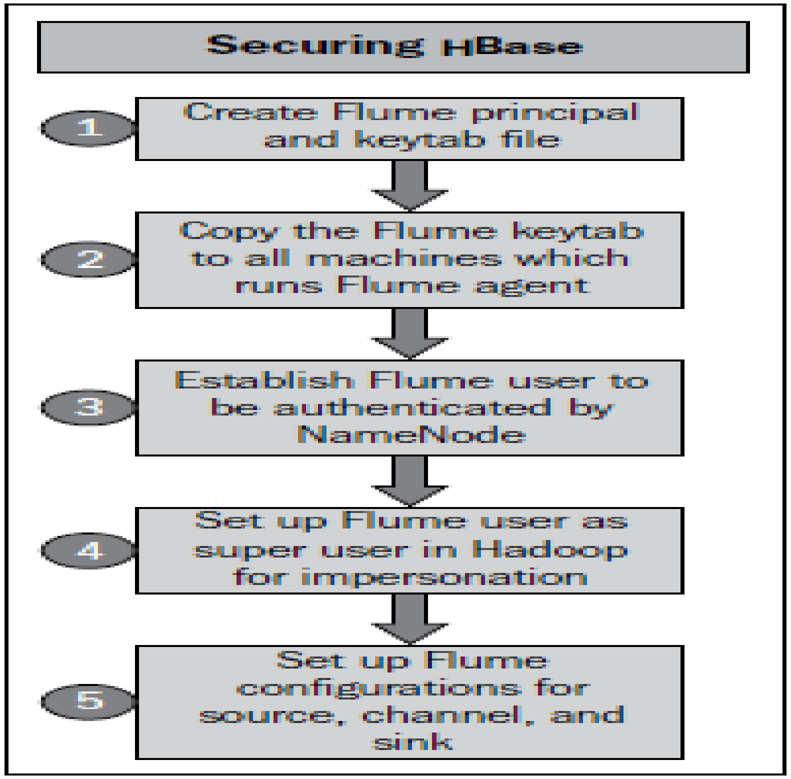
- ❑ Each of the Flume sources provide their own authentication and authorization mechanism
- ❑ Can be configured based on the type of the Flume source used.
- ❑ For example, JMS source provides the proper username and password file
- ❑ Similarly, Avro source supports SSL-based encryption where the credentials are managed in the Java keystore file.
- ❑ The following properties are used to configure the security for Avro source:



Property	Description
`ssl`	The SSL flag should be set to `true`. Default it is set to `false`.
`keystore`	Specify the path to the Java `keystore` file.
`keystore-password`	Specify the password for the Java keystore.
`keystore-type`	Specify the type of the Java keystore. This can be "JKS" or "PKCS12". By default it is JKS.

Securing Hadoop sink

- Sinks also allow authentication and authorization based on the type of sink.
- From a Hadoop perspective, HDFS and Hbase sink are most commonly used sinks.
- To configure a secured Flume agent for HDFS and HBase sink, a Flume principal is set up in the KDC, and the `keytab` file for this principal is used for authentication.
- The Flume user is established to be a superuser in Hadoop and allows impersonation of the various sources that are going to ingest data into Hadoop.


Securing HBase
1
Create Flume principal and keytab file
2
Copy the Flume keytab to all machines which runs Flume agent
3
Establish Flume user to be authenticated by NameNode
4
Set up Flume user as super user in Hadoop for impersonation
5
Set up Flume configurations for source, channel, and sink

Securing Hadoop sink

- In Hadoop, `core-site.xml` specifies Flume as a superuser and establishes the group of users that will be impersonated by Flume, with the help of the following configuration:

```
<property>
<name>hadoop.proxyuser.flume.groups</name>
<value>*</value>
</property>
<property>
<name>hadoop.proxyuser.flume.hosts</name>
<value>*</value>
</property>
```

Securing a Flume channel

- Flume supports memory, database, and file channels.
- Data security is a concern only for database and file channels as the data is persisted external to the process running the Flume agent.
- Database channel provides user authentication and authorization controls using the standard database security mechanism and the username and password can be provided in `connection.properties.file`.
- File channels provides security using encryption of the data stored in the file.
- File channels support data encryption using the Java keystores.

Securing Pig

- ❑ Similar to Hive, Pig provides user the ability to write the logic in procedural way in language known as Pig Latin, and then Pig converts this logic to a pipeline of MapReduce jobs and executes on the Hadoop cluster.
- ❑ Pig uses the user credentials to submit the job to Hadoop.
- ❑ So there is no need of any additional Kerberos security authentication.
- ❑ Pig users need to have a valid Kerberos ticket which is obtained by running kinit.
- ❑ So before starting Pig, the user should authenticate with KDC and obtain a valid Kerberos ticket.
- ❑ The following are the steps to access Pig in a secured Hadoop cluster:
  1. User logs into the machine where Pig is installed with the user credentials.
  2. User performs a kinit operation and gets the Kerberos **ticket-granting ticket** (**TGT**) for the user.
  3. When the user invokes Pig Grunt or runs the Pig script, Hadoop fetches the ticket from the ticket cache and uses it for authentication.

Best practices for securing the Hadoop ecosystem components

- ❑ All services that are running within the Hadoop ecosystem need to be authenticated with KDC.
- ❑ It is a best practice to store the KDC credentials in an LDAP store, so that the credentials and authorizations can be centrally managed.
- ❑ The `keytab` file needs to be secured, and only the user for whom the file is created should be provided with read access to the file.
- ❑ Whenever a Java client is accessing the service, client authentication should be done by the service using RPC authentication mechanism.
- ❑ Whenever user impersonation is used to impersonate an end user by the service user, the service process has to be fully secured by Kerberos and also the host running the service should have limited user access.
- ❑ Whenever Hadoop services are acting on end user's behalf, it is better to run the operation as the request user by impersonation.
- ❑ To run secured authentication in JVM, we can use the JAAS file to perform the authentication.
- ❑ Enforce strict firewall rules on the cluster to ensure that there is no unauthorized access.

Thank You !!!