Protegrity Big Data Protector Guide 9.1.0.0

Created on: Nov 19, 2024

Notice

Copyright

Table of Contents

Chapter 1

Introduction to This Guide

This guide provides information about configuring and using the Protegrity Big Data Protector (BDP) for Hadoop.

This guide should be used along with the *Protegrity Enterprise Security Administrator Guide 9.1.0.0*, which explains the mechanism of managing the data security policy.

It is recommended that you first read the sections explaining the basics of Big Data Protector in this guide.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

1.1 Sections contained in this Guide

This section provides a short description about the sections contained in this guide.

The guide is broadly divided into the following sections:

- Section *Introduction to This Guide* defines the purpose and scope for this guide. In addition, it explains how information is organized in this guide.
- Section *Overview of the Big Data Protector* provides a general idea of Hadoop and how it has been integrated with the Big Data Protector. In addition, it describes the protection coverage of various Hadoop ecosystem applications, such as MapReduce, Hive and Pig, and information about HDFS File Protection (HDFSFP).
- Section *Hadoop Application Protector* provides information about Hadoop Application Protector.
- Section *HDFS File Protector (HDFSFP) (Deprecated from Big Data Protector 7.2.0)* provides information about the protection of files stored in HDFSFP.
- Section *HBase* provides information about the Protegrity HBase protector.
- Section *Impala* provides information about the Protegrity Impala protector.
- Section *Spark* provides information about the Protegrity Spark Java and Spark SQL protectors. In addition, it provides information about Spark Scala.
- Section *Appendix: Return Codes* provides information about all possible error codes and error descriptions for Big Data Protector.
- Section *Appendix: Using Hive with HDFSFP (Deprecated from Big Data Protector 7.2.0)* provides information about using Hive with HDFSFP.
- Section *Appendix: Configuring Talend with HDFSFP (Deprecated from Big Data Protector 7.2.0)* provides the procedures for configuring Talend with HDFSFP.

- Section *Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database* describes procedures for migrating tokenized Unicode data from and to a Teradata database.
- Section *Appendix: Using Sample Data* provides examples of sample data that you must create for testing the Big Data protector.

1.2 Accessing the Protegrity documentation suite

This section describes the methods to access the *Protegrity Documentation Suite* using the *My.Protegrity* portal.

Chapter 2

Overview of the Big Data Protector

The Protegrity Big Data Protector, which was released over four years back, was first to support fine grained data protection on Hadoop and enterprise-ready Hadoop native encryption.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

Protegrity has regularly updated Big Data Protector to include support for MapReduce, Hive, Pig, HBase, Impala, and Spark.

Starting from the Big Data Protector, version 7.0, which released in 2017, support for native installers and rolling restarts for the Cloudera and Ambari environments is added.

Protegrity also supports Spark SQL, Spark streaming, Kafka, and Flume type of real-time ingestion tools. You can achieve both, batch and real time data protection, using Protegrity Big Data Protector.

The following are some of the use cases that the Protegrity Big Data Protector satisfies:

- Data protection at source applications: In this case, the sensitive data is fully protected wherever it flows, including the Hadoop ecosystem.

  In addition, it ensures that the Hadoop system, which stores the protected data, is not brought into scope for PCI, PII, GDPR, HIPPA, and other compliance policies.

- Data protection during import into landing zones (ETL process): In this case, the sensitive data is protected before it lands into the Hadoop ecosystem.

This option does not require interfacing at the source applications while maintaining the Hadoop system's status as outside the scope of compliance requirements.

- Coarse-grained data encryption within Hadoop: Protegrity HDFSFP also offers the option of encryption at the file or volume level, which is equivalent of Hadoop native encryption.

  It provides security for unstructured or sensitive data that is stored in Hadoop. Protegrity Big Data Protector provides coarse-grained encryption that uses the same Key management infrastructure, which is used for other Protegrity protectors driven by ESA to ensure that one key or certificate management solution protects the sensitive data fields and the physical disks in case of a loss.

In the Protegrity Big Data Protector for Apache Hadoop, the data is split and shared with all the data nodes in the Hadoop cluster. The Big Data Protector is deployed on each of these nodes and the PEP Server, where the protection enforcement policies are shared.

The Protegrity Big Data Protector is scalable and new nodes can be added as required. It is cost effective since massively parallel computing is done on commodity servers, and it is flexible as it can work with data from any number of sources. The Big Data Protector is fault tolerant as the system redirects the work to another node if a node is lost. It can handle all types of data, such as structured and unstructured data, irrespective of their native formats.

The Big Data Protector protects data, which is handled by various Hadoop applications and protects files stored in the cluster. MapReduce, Hive, Pig, HBase, and Impala can use Protegrity protection interfaces to protect data as it is stored or retrieved from the Hadoop cluster. All standard protection techniques offered by Protegrity are applicable to Big Data Protector.

For more information about the available protection options, such as data types, Tokenization or Encryption types, or length preserving and non-preserving tokens, refer to *Protection Methods Reference Guide 9.1.0.0*.

2.1 Components of Hadoop

The Big Data Protector works on the Hadoop framework as shown in the following figure.

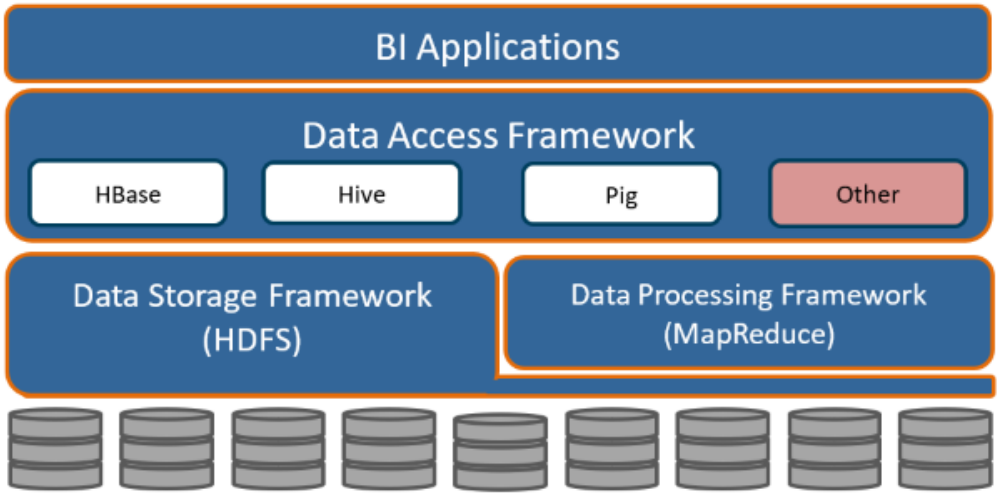


Figure 2-1: Hadoop Components

> **Note:**
> The illustration of Hadoop components is an example.
>
> Based on requirements, the components of Hadoop might be different.

Hadoop interfaces have been used extensively to develop the Big Data Protector. It is a common deployment practice to utilize Hadoop Distributed File System (HDFS) to store the data, and let MapReduce process the data and store the result back in HDFS.

2.1.1 Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) spans across all nodes in a Hadoop cluster for data storage. It links together the file systems on many nodes to make them into one big file system. HDFS assumes that nodes will fail, so data is replicated across multiple nodes to achieve reliability.

2.1.2 MapReduce

The MapReduce framework assigns work to every node in large clusters of commodity machines. MapReduce programs are sets of instructions to parse the data, create a map or index, and aggregate the results. Since data is distributed across multiple nodes, MapReduce programs run in parallel, working on smaller sets of data.

A MapReduce job is executed by splitting each job into small Map tasks, and these tasks are executed on the node where a portion of the data is stored. If a node containing the required data is saturated and not able to execute a task, then MapReduce shifts the task to the least busy node by replicating the data to that node. A Reduce task combines results from multiple Map tasks, and store all of them back to the HDFS.

2.1.3 Hive

The Hive framework resides above Hadoop to enable ad hoc queries on the data in Hadoop. Hive supports HiveQL, which is similar to SQL. Hive translates a HiveQL query into a MapReduce program and then sends it to the Hadoop cluster.

2.1.4 Pig

Pig is a high-level platform for creating MapReduce programs used with Hadoop.

2.1.5 HBase

HBase is a column-oriented datastore, meaning it stores data by columns rather than by rows. This makes certain data access patterns much less expensive than with traditional row-oriented relational database systems. The data in HBase is protected transparently using Protegrity HBase coprocessors.

2.1.6 Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

The Impala daemon runs on each node in the cluster, reading and writing to data in the files, and accepts queries from the Impala shell command. The following are the core components of Impala:

- Impala daemon (*impalad*) – This component is the Impala daemon which runs on each node in the cluster. It reads and writes the data in the files and accepts queries from the Impala shell command.
- Impala Statestore (*statestored*) – This component checks the health of the Impala daemons on all the nodes contained in the cluster. If a node is unavailable due to any error or failure, then the Impala *statestore* component informs all other nodes about the failed node to ensure that new queries are not sent to the failed node.
- Impala Catalog (*catalogd*) – This component is responsible for communicating any changes in the metadata received from the Impala SQL statements to all the nodes in the cluster.

2.1.7 Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time.

Spark leverages the physical memory of the Hadoop system and utilizes Resilient Distributed Datasets (RDDs) to store the data in-memory and lowers latency, if the data fits in the memory size. The data is saved on the hard drive only if required.

2.2 Features of Protegrity Big Data Protector

The Protegrity Big Data Protector (Big Data Protector) uses patent-pending vaultless tokenization and central policy control for access management and secures sensitive data at rest in the following areas:

- Data in HDFS
- Data used during MapReduce, Hive and Pig processing, and with HBase, Impala, and Spark
- Data traversing enterprise data systems

The data is protected from internal and external threats, and users and business processes can continue to utilize the secured data.

Data protection may be by encryption or tokenization. In tokenization, data is converted to similar looking inert data known as tokens where the data format and type can be preserved. These tokens can be detokenized back to the original values when it is required.

Protegrity secures files with volume encryption and also protects data inside files using tokenization and strong encryption protection methods. Depending on the user access rights and the policies set using Policy management in ESA, this data is unprotected.

The Protegrity Hadoop Big Data Protector provides the following features:

- Provides fine grained field-level protection within the MapReduce, Hive, Pig, HBase, and Spark frameworks.
- Provides directory and file level protection (encryption).
- Provides Protegrity Format Preserving Encryption (FPE) method for structured data. The following data types are supported:
  - Numeric (0-9)
  - Alpha (a-z, A-Z)
  - Alpha-Numeric (0-9, a-z, A-Z)
  - Credit Card (0-9)
  - Unicode Basic Latin and Latin-1 Supplement Alpha
  - Unicode Basic Latin and Latin-1 Supplement Alpha-Numeric

  For more information about FPE, refer to *Protection Methods Reference Guide 9.0.0.0*.
- Retains distributed processing capability as field-level protection is applied to the data.
- Protects data in the Hadoop cluster using role-based administration with a centralized security policy.
- Starting from the Big Data Protector, version 7.0 release, native installers for the Cloudera and Ambari environments are being provided. These new installers simplify the task of installing, configuring, and managing Big Data Protector using Cloudera Manager or the Ambari UI.
- Simplified installation, administration, and management of Big Data Protector using the following components:
  - Parcels: In Cloudera Manager, a Big Data Protector Parcel, which is a single consolidated file, contains all the required files for installing and using Big Data Protector on a cluster and the metadata used by Cloudera Manager.
  - Custom Service Descriptors (CSDs): In Cloudera Manager, a CSD contains all the configurations required to describe and manage the Big Data Protector services. The CSDs are provided as Jar files.

  - Management Packs: In Ambari, a Big Data Protector management pack, which is a single consolidated file, contains all the required files for installing and using Big Data Protector on a cluster and the metadata used by the Ambari UI.
- Easy monitoring of the Big Data Protector services, such as, BDP PEP, using the Cloudera Manager UI instead of the CLI.
- Easy monitoring of the Big Data Protector services, such as BDP PEP and BDP HDFSFP, using Ambari instead of the CLI.
- Automatic deactivation of older Big Data Protector parcels in Cloudera Manager on update.
- Provides logging and viewing data access activities and real-time alerts with a centralized monitoring system.
- Ensures minimal overhead for processing secured data, with minimal consumption of resources, threads and processes, and network bandwidth.
- Provides transparent data protection with Protegrity HBase protectors.
- Transparently protects files processed by MapReduce and Hive in HDFS using HDFSFP.

Note:
The following figure illustrates the various components in an Enterprise Hadoop ecosystem.

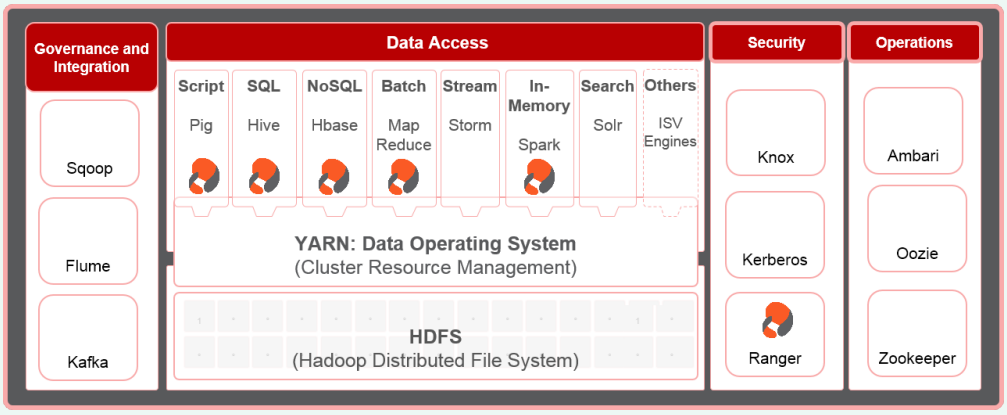


Figure 2-2: Enterprise Hadoop Components

Currently, Protegrity supports MapReduce, Hive, Pig, and HBase which utilize HDFS as the data storage layer. The following points can be referred to as general guidelines:

Note:
Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 8.1.0.0.

- Sqoop: Sqoop can be used for ingestion into HDFSFP protected zone (For Hortonworks, Cloudera, and Pivotal HD).
- Beeline, Beeswax, and Hue on Cloudera: Beeline, Beeswax, and Hue are certified with Hive protector and Hive with HDFSFP integrations.
- Beeline, Beeswax, and Hue on Hortonworks & Pivotal HD: Beeline, Beeswax, and Hue are certified with Hive protector and Hive with HDFSFP integrations.
- Ranger (Hortonworks): Ranger is certified to work with the Hive protector and Hive with HDFSFP integrations only.
- Sentry (Cloudera): Sentry is certified with Hive protector, Hive with HDFSFP integrations, and Impala protector only.
- MapReduce and HDFSFP integration is certified with *TEXTFILE* format only.
- Hive and HDFSFP integration is certified with *TEXTFILE* format only.
- Pig and HDFSFP integration is certified with *TEXTFILE* format only.

We neither support nor have certified other components in the Hadoop stack. We strongly recommend consulting Protegrity, before using any unsupported components from the Hadoop ecosystem with our products.

2.3 Using Protegrity Data Security Platform with Hadoop

To protect data, the components of the Protegrity Data Security Platform are integrated into the Hadoop cluster as shown in the following figure.

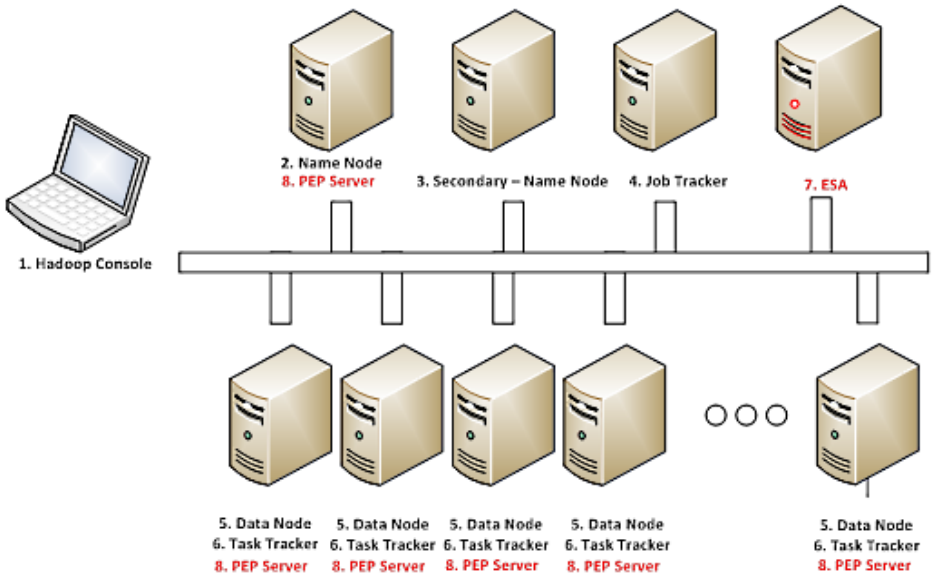


Figure 2-3: Protegrity Data Security Platform with Hadoop

The Enterprise Security Administrator (ESA) is a soft appliance that needs to be pre-installed on a separate server, which is used to create and manage policies.

For more information about installing the ESA, and creating and managing policies, refer to *Installation Guide 9.1.0.0* and *Policy Management Guide 9.1.0.0* respectively.

To achieve a parallel nature for the system, a PEP Server is installed on every data node. It is synchronized with the connection properties of ESA.

Each task runs on a node under the same Hadoop user. Every user has a policy deployed for running their jobs on this system. Hadoop manages the accounts and users. You can get the Hadoop user information from the actual job configuration.

HDFS implements a permission model for files and directories, based on the Portable Operating System Interface (POSIX) for Unix model. Each file and directory is associated with an owner and a group. Depending on the permissions granted, users for the file and directory can be classified into one of these three groups:

- Owner
- Other users of the group
- All other users

2.4 Overview of Hadoop Application Protection

This section describes the various levels of protection provided by Hadoop Application Protection.

2.4.1 Protection in MapReduce Jobs

A MapReduce job in the Hadoop cluster involves sensitive data. You can use Protegrity interfaces to protect data when it is saved or retrieved from a protected source. The output data written by the job can be encrypted or tokenized. The protected data can be subsequently used by other jobs in the cluster in a secured manner. Field level data can be secured and ingested into HDFS by independent Hadoop jobs or other ETL tools.

For more information about secure ingestion of data in Hadoop, refer to section *Ingesting Files Using Hive Staging*.

For more information on the list of available APIs, refer to section *MapReduce APIs*.

If Hive queries are created to operate on sensitive data, then you can use Protegrity Hive UDFs for securing data. While inserting data to Hive tables, or retrieving data from protected Hive table columns, you can call Protegrity UDFs loaded into Hive during installation. The UDFs protect data based on the input parameters provided.

Secure ingestion of data into HDFS to operate Hive queries can be achieved by independent Hadoop jobs or other ETL tools.

For more information about securely ingesting data in Hadoop, refer to section *Ingesting Data Securely*.

2.4.2 Protection in Hive Queries

Protection in Hive queries is done by Protegrity Hive UDFs, which translates a HiveQL query into a MapReduce program and then sends it to the Hadoop cluster.

For more information on the list of available UDFs, refer to section *Hive UDFs*.

2.4.3 Protection in Pig Jobs

Protection in Pig jobs is done by Protegrity Pig UDFs, which are similar in function to the Protegrity UDFs in Hive.

For more information on the list of available UDFs, refer to section *Pig UDFs*.

2.4.4 Protection in HBase

HBase is a database which provides random read and write access to tables, consisting of rows and columns, in real-time. HBase is designed to run on commodity servers, to automatically scale as more servers are added, and is fault tolerant as data is divided across servers in the cluster. HBase tables are partitioned into multiple regions. Each region stores a range of rows in the table. Regions contain a datastore in memory and a persistent datastore(HFile). The Name node assigns multiple regions to a region server. The Name node manages the cluster and the region servers store portions of the HBase tables and perform the work on the data.

The Protegrity HBase protector extends the functionality of the data storage framework and provides transparent data protection and unprotection using coprocessors, which provide the functionality to run code directly on region servers. The Protegrity coprocessor for HBase runs on the region servers and protects the data stored in the servers. All clients which work with HBase are supported.

The data is transparently protected or unprotected, as required, utilizing the coprocessor framework.

For more information about HBase, refer to section *HBase*.

2.4.5 Protection in Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

The Protegrity Impala protector extends the functionality of the Impala query engine and provides UDFs which protect or unprotect the data as it is stored or retrieved.

For more information about the Impala protector, refer to section *Impala*.

2.4.6 Protection in Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time. You can also utilise Spark Streaming to process live data streams and store the processed data in Hadoop.

The Protegrity Spark Java protector extends the functionality of the Spark engine and provides Java APIs that protect, unprotect, or reprotect the data as it is stored or retrieved.

For more information about the Spark Java and SQL protectors, refer to section *Spark*.

The Protegrity Spark Java protector extends the functionality of the Spark engine and provides Java APIs that protect, unprotect, or reprotect the data as it is stored or retrieved.

The Protegrity Spark SQL protector provides native UDFs that can be utilized with Spark Scala to protect, unprotect, or reprotect the data as it is stored or retrieved.

You can create and submit Spark jobs using the methods listed in the following table.

Table 2-1: Creating and Submitting Spark Jobs

To create and submit Spark jobs with…	Refer to section…
Spark Java APIs	*Spark Java*
Spark SQL UDFs	*Spark SQL*
Spark Scala	*Spark Scala*

2.5 HDFS File Protection (HDFSFP) (Deprecated from Big Data Protector 7.2.0)

Files are stored and retrieved by Hadoop system elements, such as file shell commands, MapReduce, Hive, Pig, HBase and so on. The stored files reside in HDFS and span multiple cluster nodes.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

Most of the files in HDFS are plain text files and stored in the clear, with access control like a POSIX file system. These files contain sensitive data, making it vulnerable with exposure to unwanted users. These files are transparently protected as they are stored in HDFS. In addition, the content is exposed only to authorized users. The content in the files is unprotected transparently to processes or users, authorized to view and process the files. The user is automatically detected from the job information

provided by HDFSFP. The job accessing secured files must be initialized by an authorized user having the required privileges in ACL. The files encrypted by HDFSFP are suitable for distributed processing by Hadoop distributed jobs like MapReduce.

HDFSFP protects individual files or files stored in a directory. The access control is governed by the security policy and ACL supplied by the security officer. The access control and security policy is controlled through ESA interfaces. Command line and UI options are available to control ACL entries for file paths and directories.

2.6 Ingesting Data Securely

This section describes the ways in which data can be secured and ingested by various jobs in Hadoop at a field or file level.

2.6.1 Ingesting Files Using Hive Staging

Semi-structured data files can be loaded into a Hive staging table for ingestion into a Hive table with Hive queries and Protegrity UDFs. After loading data in the table, the data will be stored in protected form.

2.6.2 Ingesting Files into HDFS by HDFSFP (Deprecated from Big Data Protector 7.2.0)

The HDFSFP component of Big Data Protector can be used for ingesting files securely in HDFS. It provides granular access control for the files in HDFS. You can ingest files using the command shell and Java API in HDFSFP.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

For more information about using HDFSFP, refer to section *HDFS File Protector (HDFSFP) (Deprecated from Big Data Protector 7.2.0)*.

2.7 Data Security Policy and Protection Methods

A data security policy establishes processes to ensure the security and confidentiality of sensitive information. In addition, the data security policy establishes administrative and technical safeguards against unauthorized access or use of the sensitive information.

Depending on the requirements, the data security policy typically performs the following functions:

- Classifies the data that is sensitive for the organization.
- Defines the methods to protect sensitive data, such as encryption and tokenization.
- Defines the methods to present the sensitive data, such as masking the display of sensitive information.
- Defines the access privileges of the users that would be able to access the dasta.
- Defines the time frame for privileged users to access the sensitive data.
- Enforces the security policies at the location where sensitive data is stored.
- Provides a means of auditing authorized and unauthorized accesses to the sensitive data. In addition, it can also provide a means of auditing operations to protect and unprotect the sensitive data.

The data security policy contains a number of components, such as, data elements, datastores, member sources, masks, and roles. The following list describes the functions of each of these entities:

- **Data elements** define the data protection properties for protecting sensitive data, consisting of the data securing method, data element type and its description. In addition, Data elements describe the tokenization or encryption properties, which can be associated with roles.
- **Datastores** consist of enterprise systems, which might contain the data that needs to be processed, where the policy is deployed and the data protection function is utilized.
- **Member sources** are the external sources from which users (or members) and groups of users are accessed. Examples are a file, database, LDAP, and Active Directory.
- **Masks** are a pattern of symbols and characters, that when imposed on a data field, obscures its actual value to the user. Masks effectively aid in hiding sensitive data.
- **Roles** define the levels of member access that are appropriate for various types of information. Combined with a data element, roles determine and define the unique data access privileges for each member.

For more information about the data security policies, protection methods, and the data elements supported by the components of the Big Data Protector, refer to *Protection Methods Reference Guide 9.1.0.0*.

2.8 Installing and Uninstalling Big Data Protector

For information about installing and uninstalling the Big Data Protector, refer to *Installation Guide 9.1.0.0*.

2.9 Working with the Log Forwarder

The Log Forwarder is a log processor tool running along with the PEP server on a cluster node. It serves the purpose of collecting, aggregating, caching, and moving the logs from the PEP server (Application log) and the Big Data Protector (Audit log) to the Audit Store on the ESA and PSU.

The Log Forwarder uses the *15780* port, which is configurable, to receive Audit Log and Protection Log on each cluster node and send the logs to the Appliance running the Audit Store. The appliance can be the ESA or the Protegrity Storage Unit (PSU). The Appliance communicates with the Log Forwarder using the port *9200*.

For more information about the Protegrity Storage Unit (PSU), refer the *Protegrity Storage Unit Guide 9.1.0.0*.

For more information about logging, refer to the *Protegrity Log Management Guide 9.1.0.0*.

For more information about the Audit Store, refer to the *Audit Store Guide 9.1.0.0*.

The Log Forwarder aggregates logs at 10 second intervals. For the Big Data Protector, the components of Log Forwarder are configured in the *pepserver.cfg* file.

For more information about the Log Forwarder related configurations, refer to the section *Appendix: PEP Server Configuration File* in the *Protegrity Installation Guide 9.1.0.0*.

2.9.1 Logging Architecture

Depending on the proxy configuration of the Big Data Protector, the logging architecture can be specified as following types:

- Logging Architecture of the Big Data Protector cluster without the DPS Proxy
- Logging Architecture of the Big Data Protector cluster with the DPS Proxy

2.9.2 Logging Architecture of the Big Data Protector Cluster without the DPS Proxy

This section explains the logging architecture of the Big Data Protector cluster without the DPS Proxy.

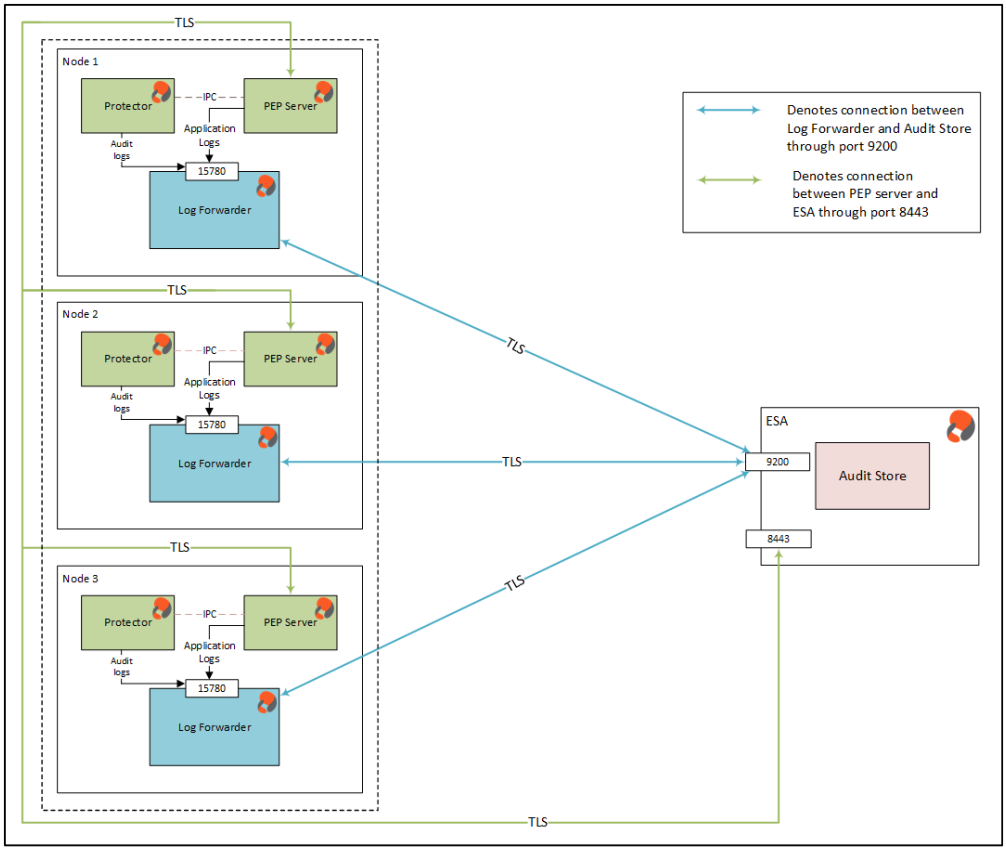


Figure 2-4: Big Data Protector Cluster Logging Architecture without DPS Proxy

In a multi-node architecture, the Log Forwarder on each cluster node collects the Application logs and Audit logs from the PEP server and the Big Data Protector and sends the logs to the to the Appliance running the Audit Store. The appliance can be the ESA or the Protegrity Storage Unit (PSU).

2.9.3 Logging Architecture of the Big Data Protector Cluster with the DPS Proxy

This section explains the logging architecture of the Big Data Protector cluster with the DPS Proxy.

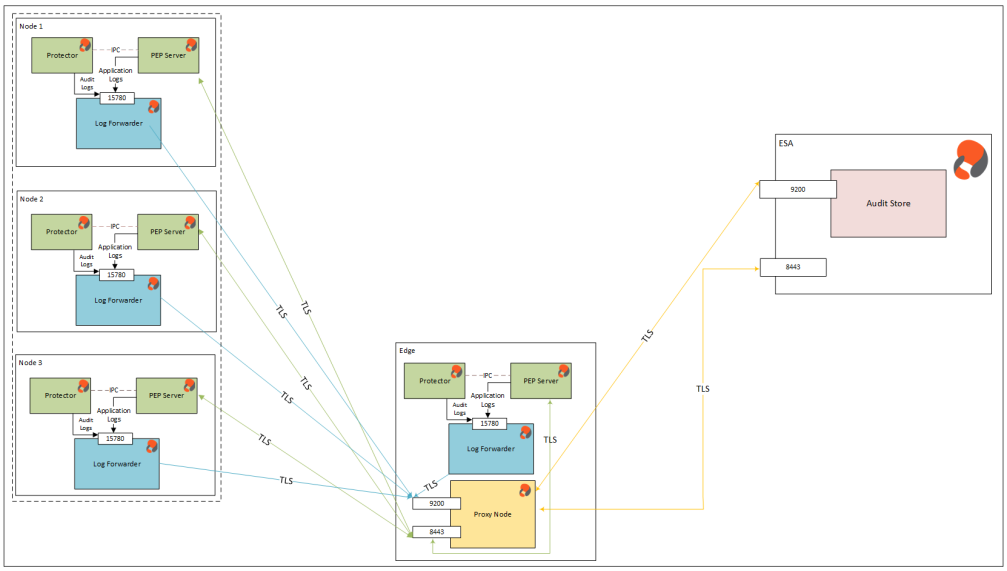


Figure 2-5: Big Data Protector Cluster Logging Architecture with DPS Proxy

In this scenario, the Log Forwarder on each cluster node collects the Application logs and Audit logs from the PEP server and the Big Data Protector and sends the logs to the DPS Proxy. The DPS Proxy sends the logs to the Appliance running the Audit Store. The appliance can be the ESA or the Protegrity Storage Unit (PSU).

2.10 Using Health Check

The Big Data Protector has introduced the health check feature from the Big Data Protector 8.1.0.0 release onwards. Before the Big Data Protector 8.1.0.0 release, if a job is scheduled on a cluster, and the PEP server goes down on a single node, then the job fails. The Health check feature helps you to overcome this issue by identifying that single node as unhealthy, so that a new job is not scheduled on that node.

Caution: Currently, the health check feature in Big Data Protector 9.1.0.0 release is only supported for YARN-based services, such as, MapReduce, Hive, and Spark.

The Big Data Protector supports health check for the following distributions:

- CDH
- CDP
- HDP

2.10.1 Using Health Check on an HDP Distribution

The Cloudera Manager enables you to create and register a custom alert in the Ambari Web interface. You can configure this alert to trigger an email when the PEP server fails on any node in the cluster.

2.10.1.1 Enabling the Health Check for an HDP Distribution

This section describes the procedure to enable the health check in the Big Data Protector running on an HDP distribution.

Depending on the requirement, the health check in the Big Data Protector can be enabled during installation or after installation.

2.10.1.1.1 Enabling the Health Check During Installation of the Big Data Protector

The health check is disabled by default. During installation of the Big Data Protector, the health check can be enabled while running the configurator script.

Note: For more information about enabling the health check during installation, refer to the *Installation Guide 9.1.0.0*.

2.10.1.1.2 Enabling the Health Check after Installing the Big Data Protector

If you fail to enable the health check feature during the installation of the Big Data Protector, then you can enable the health check feature manually after the installation is complete.

➤ To enable the health check feature:

1. On the Ambari Home screen, click the **YARN** service.
   The **YARN** screen appears.

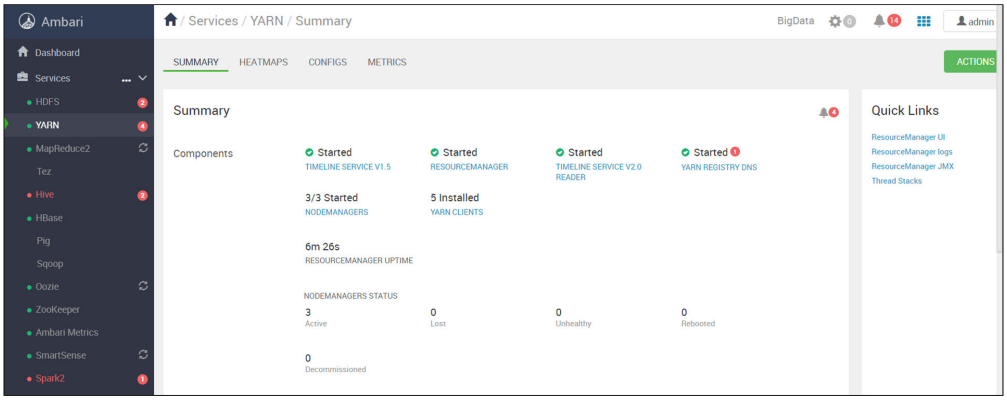


Figure 2-6: YARN Screen

2. Click the **CONFIGS** tab.
The **CONFIGS** tab appears.

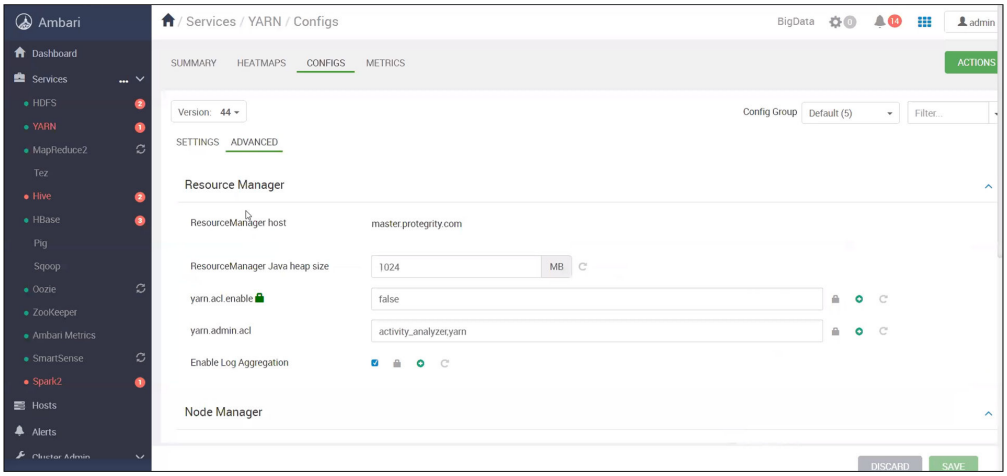


Figure 2-7: CONFIG Tab

3. Click the **ADVANCED** tab.
The **ADVANCED** setting page appears.
4. Scroll to the bottom of the page and click **Custom yarn-site**.
The following parameters appear.

Figure 2-8: YARN Configuration Parameters

5. Click **Add Property**.
The **Add Property** page appears.

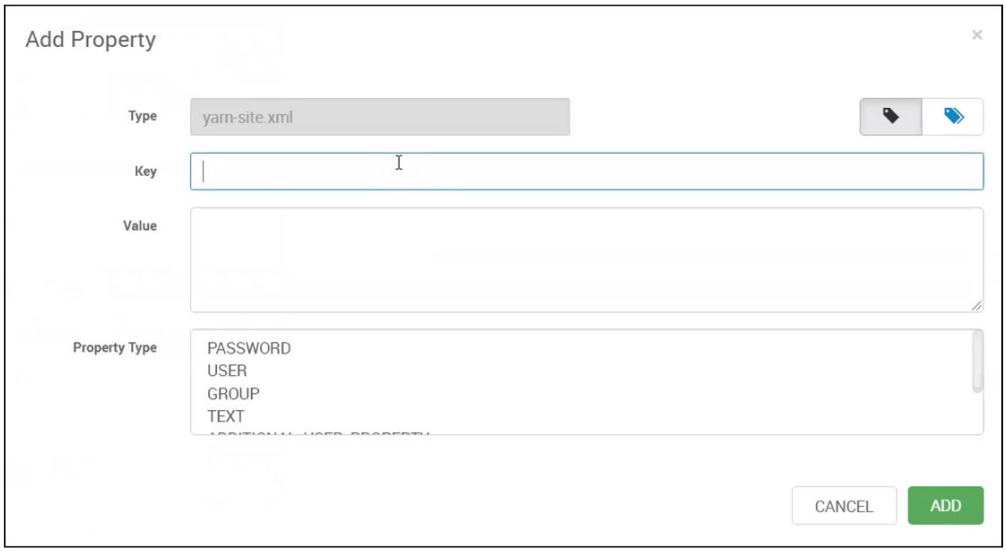


Figure 2-9: Add Property Page

6. In the **Key** box, type *yarn.nodemanager.health-checker.script.path*.
7. In the **Value** box, type the path of the healthchecker script.
   For example, */opt/protegrity/9.1.0.0.x/pephealthcheck/pgreppep.sh*.

   where,

   - */opt/protegrity/9.1.0.0.x/* - specifies the installation directory for Protegrity products
   - *pephealthcheck/pgreppep.sh* - specifies the location of the healthchecker script
8. To add the *yarn.nodemanager.health-checker.script.path* key, click **ADD**.
9. To save the *yarn.nodemanager.health-checker.script.path* key, click **SAVE**.
10. On the Ambari Home screen, navigate to the **SUMMARY** tab.
11. Click **Restart**.
    The **Restart** menu appears.

Figure 2-10: Restart Drop Down Menu

12. To restart the YARN services, select **Restart All Affected**.
    The **Confirmation** page appears.

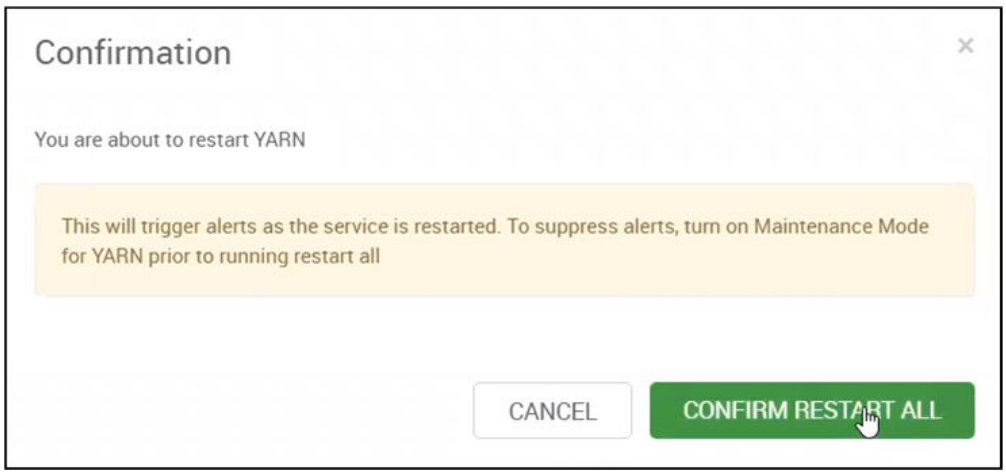


Figure 2-11: Confirmation Page

13. Click **CONFIRM RESTART ALL**.
    The **Background Operations** screen appears and the status of all YARN services are displayed.

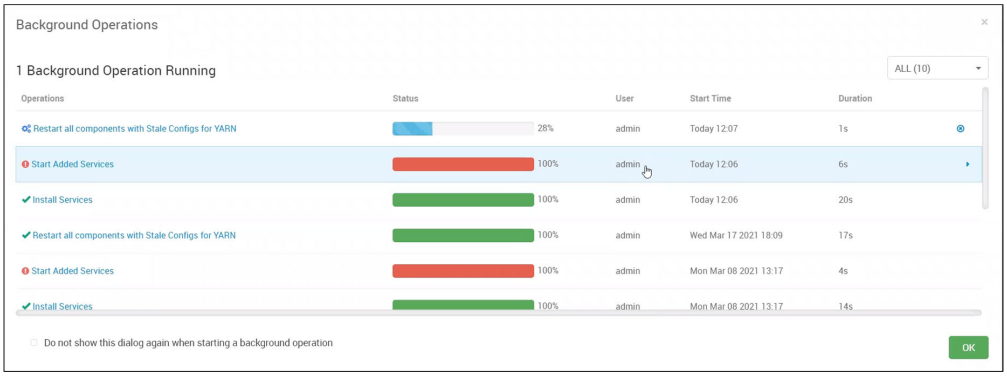


Figure 2-12: Background Operations Screen

14. If all the YARN service operations are complete, then click **OK**.

2.10.1.2 Disabling the Health Check for an HDP Distribution

This section describes the procedure to disable the health check feature in the Big Data Protector running on an HDP distribution.

To disable the health check:

1. On the Ambari Home screen, click the **YARN** service.
   The **YARN** screen appears.

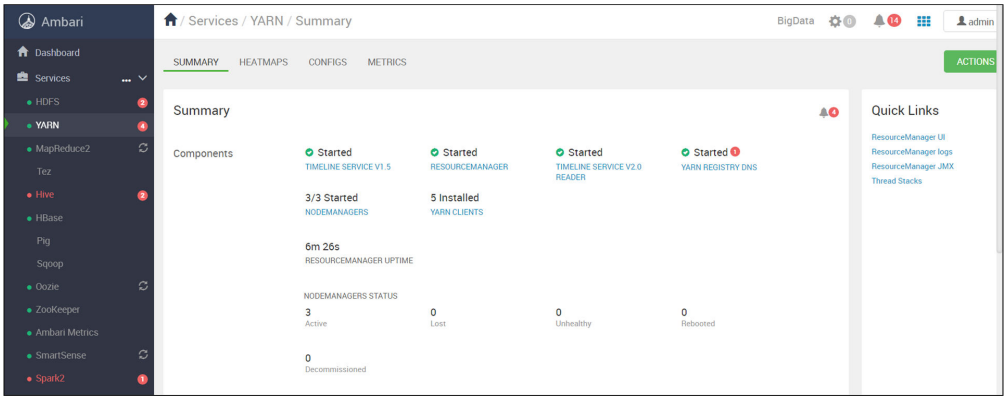


Figure 2-13: YARN Screen

2. Click the **CONFIGS** tab.
   The **CONFIGS** tab appears.

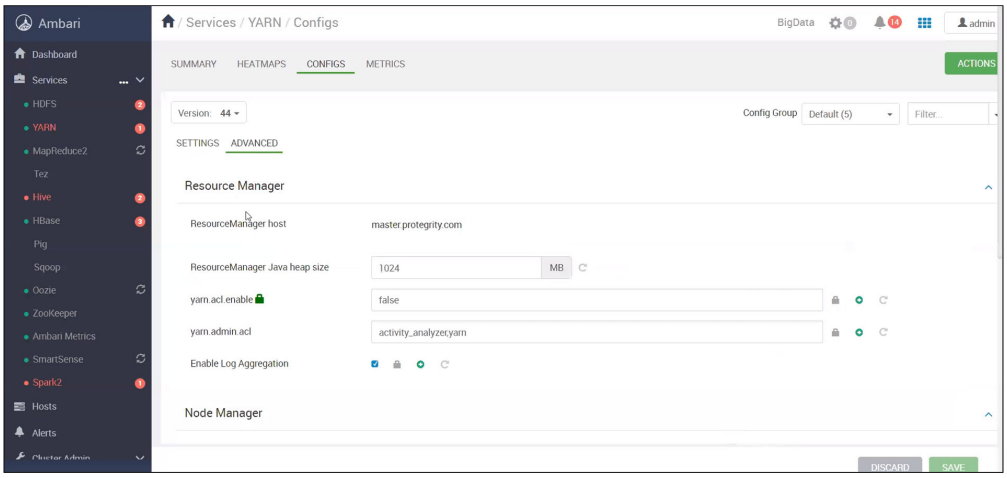


Figure 2-14: CONFIG Tab

3. Click the **ADVANCED** tab.
   The **ADVANCED** setting screen appears.
4. Scroll to the bottom of the page and click **Custom yarn-site**.
   The following parameters appear.

Figure 2-15: YARN Configuration Parameters

5. Besides the *yarn.nodemanager.health-checker.script.path* box, click  .
   The **Configurations** screen appears.

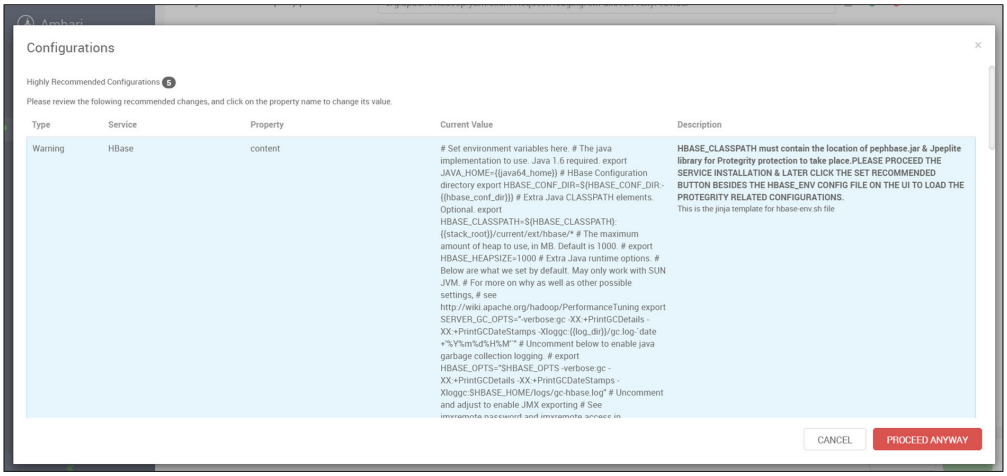


Figure 2-16: Configurations Screen

6. Click **PROCEED ANYWAY**.
7. Click **SAVE**.
   The **Save Configuration Changes** page appears.

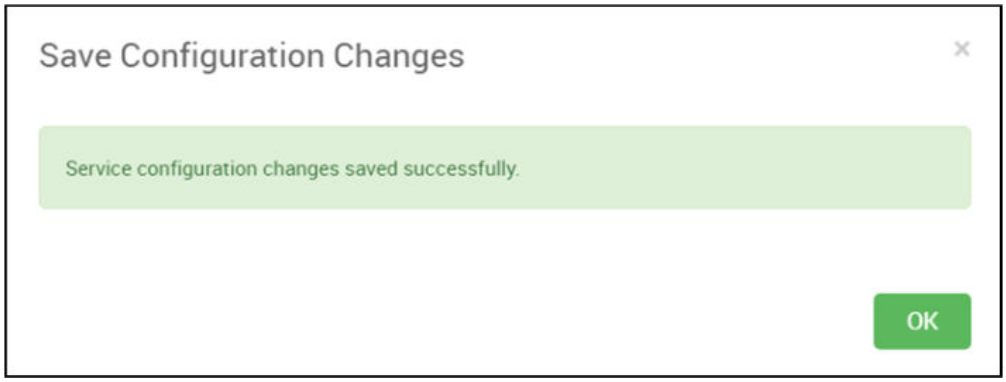


Figure 2-17: Save Configuration Changes Box

8. Click **OK**.
9. On the Ambari Home screen, navigate to the **SUMMARY** tab.
10. Click **Restart**.
    The **Restart** menu appears.

Figure 2-18: Restart Drop Down Menu

11. To restart YARN services, select **Restart All Affected**.
    The prompt to confirm the action appears.

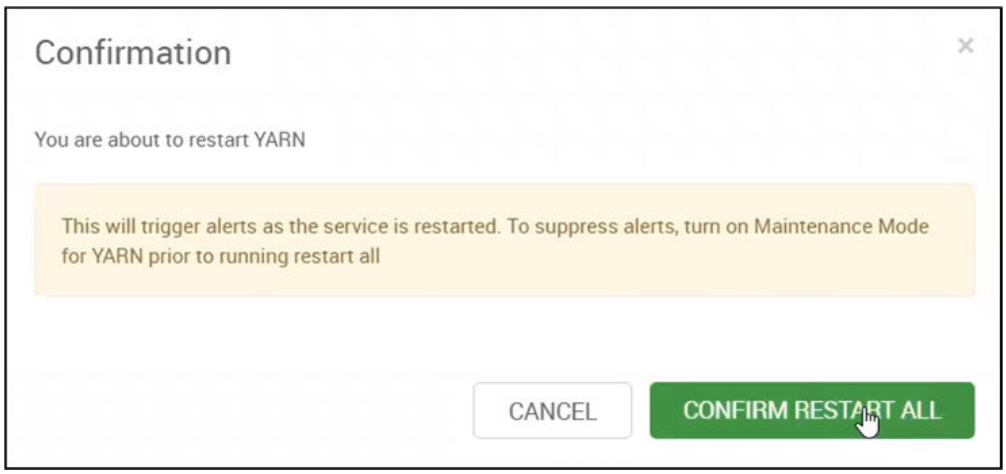


Figure 2-19: Confirmation Box

12. Click **CONFIRM RESTART ALL**.
The **Background Operations** box appears and the status of all YARN services are displayed.

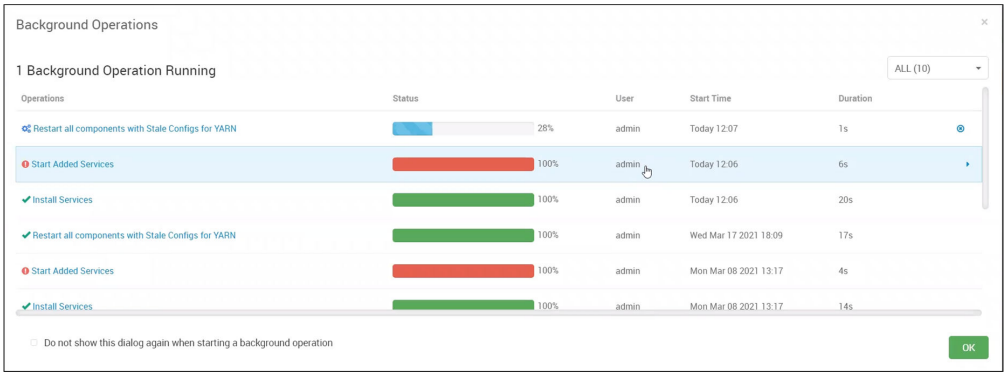


Figure 2-20: Background Operations Box

13. If all the YARN service operations are complete, then click **OK**.

2.10.1.3 Configuring Alerts in Ambari

Ambari has introduced a feature that enables you to create and register a custom alert using the Ambari Web interface. This mechanism in Ambari triggers an action based on a specified condition. For example, you can create an alert that will trigger an email or display an alert on the dashboard if the PEP server fails on any node in the cluster.

➤ To display the status of BDP service in the Ambari alerts dashboard:

1. To determine the status of the service, create the following files:
   - *alerts.json*

     Note: For more information about the contents of the *alerts.json* file, refer to the section *Sample Files for Ambari Alerts*.

   - *checkpep.py*

     Note: For more information about the contents of the *checkpep.py* file, refer to the section *Sample Files for Ambari Alerts*.

2. On the Ambari server, copy the Python script, *checkpep.py*, to the following directory.

```
/var/lib/ambari-server/resources/host_scripts
```

3. To enable the agents to fetch the script, from the Ambari server, on each node in the cluster, restart the Ambari agent on each node.
   The Ambari agents will fetch the Python scripts and download them to the nodes in the */ambari-agent/cache/* directory. For example,

   ```
   /var/lib/ambari-agent/cache/host_scripts
   ```

   > **Note:** If the Ambari agents are unable to fetch the Python script, then you must manually copy them to the location specified in the example.

4. To copy the Python script to the master node, run the following command on the Master node.

   ```
   cp -r /var/lib/ambari-server/resources/host_scripts/checkpep.py /var/lib/ambari-agent/
   cache/host_scripts/
   ```

5. To install the custom alert, run the following command.

   ```
   curl -u admin:admin -i -H 'X-Requested-By:ambari' -X POST -d @alerts.json http://
   <Ambari_server_host>:8080/api/v1/clusters/<cluster_name>/alert_definitions
   ```

   The command sends the curl request to the Ambari server host.

6. To verify whether the alert is registered, run the following command.

   ```
   curl -u admin:admin -i -H 'X-Requested-By:ambari' -X GET http://
   <Ambari_server_host>:8080/api/v1/clusters/<cluster_name>/alert_definitions
   ```

7. To delete the alert, run the following command.

   ```
   curl -u admin:admin -i -H 'X-Requested-By:ambari' -X DELETE http://
   <Ambari_server_host>:8080/api/v1/clusters/<cluster_name>/alert_definitions/151
   ```

   where,

   151 - is the alert number. You can retrieve this number either by listing the alert or by locating the alert in the register.

   > **Note:** For more information about alerts, please refer to the Cloudera Community Article on how to create and register custom Ambari alerts.

2.10.2 Using Health Check on a CDP Cloud Platform

The Cloudera Manager provides the **Create Trigger** option within a service to monitors the health. For example, you can use the **Create Trigger** option in the BDP PEP service to monitor the health of the PEP servers in a cluster based on the specified conditions or triggers. If the PEP server is in a bad state in a cluster, then the **Create Trigger** option will return the number of the PEP servers in bad health. You can also use the predefined conditions or triggers for alerting purposes.

> **Note:** For more information about configuring the alert delivery, refer to *https://docs.cloudera.com/cdp-private-cloud-base/7.1.6/monitoring-and-diagnostics/topics/cm-configuring-alert-delivery.html*.

2.10.2.1 Enabling the Health Check for a CDP Cloud Platform

This section describes the procedure to enable the health check feature in the Big Data Protector running on a CDP cloud platform. You must manually add the healthchecker script and restart the YARN service to enable the health check feature.

➤ To enable the health check:

1. Log in to the Cloudera Manager Web UI.
2. Navigate to **Clusters** > **YARN**.

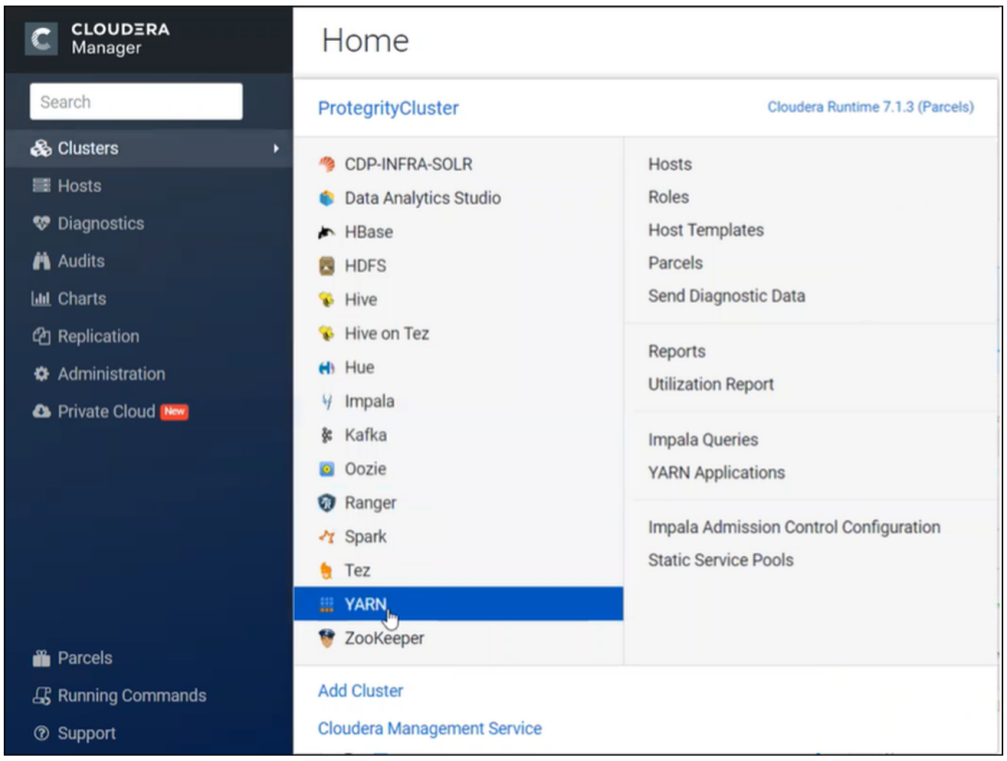


Figure 2-21: YARN Screen

3. Click the **Configuration** tab.
   The **Configuration** screen appears.

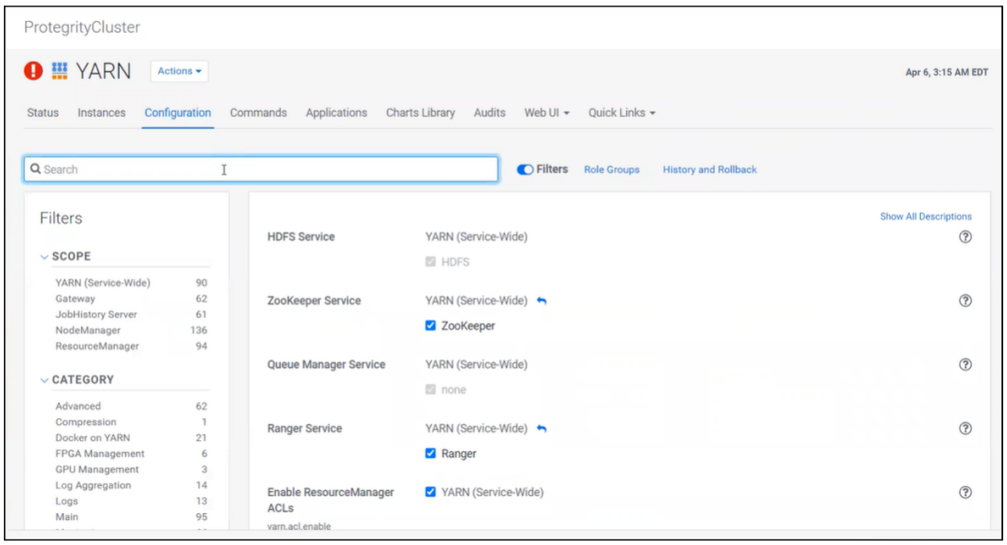


Figure 2-22: Configuration Screen

4. Navigate to the **Healthchecker Script Path** option.
   Alternatively, you can search for the health-check term in the search box.
5. In the **Healthchecker Script Path** box, type the location of the *yarn.nodemanager.health-checker.script.path* script.
   For example, */opt/cloudera/parcels/PTY_BDP/pephealthcheck/pgreppep.sh*.

   where,

- */opt/cloudera/parcels/* - specifies the Cloudera Parcel directory
- *PTY_BDP/pephealthcheck/pgreppep.sh* - specifies the path of the healthchecker script

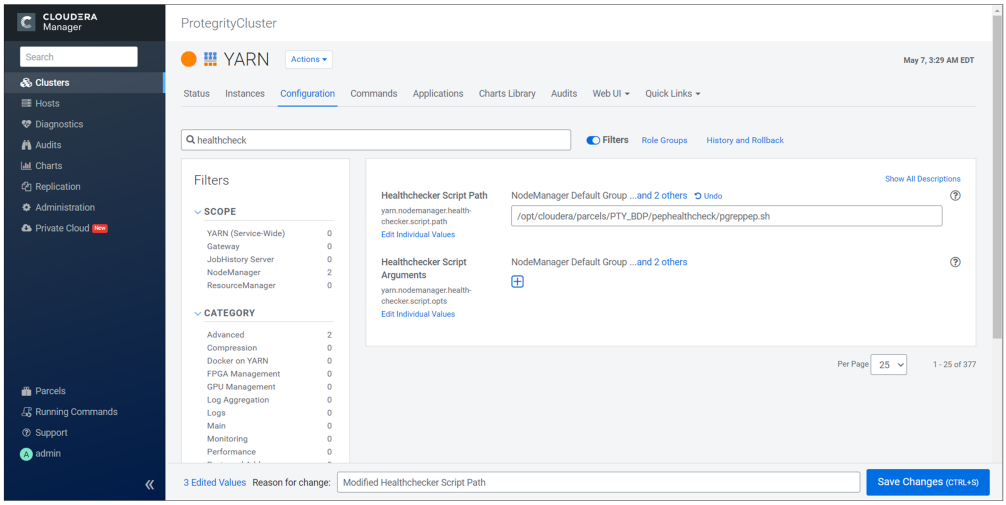


Figure 2-23: YARN Healthchecker Script path

6. Click **Save Changes**.
7. To restart the Stale Configurations, click  .
   The **Stale Configurations** screen appears.

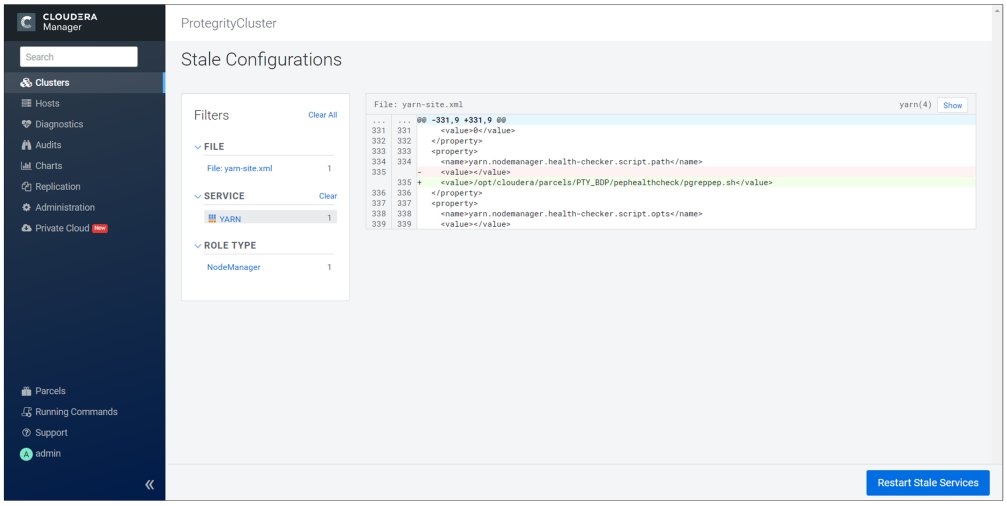


Figure 2-24: Stale Configurations Screen

8. To reload the configurations and restart the services, click **Restart Stale Services**.
   The **Review Changes** screen appears.

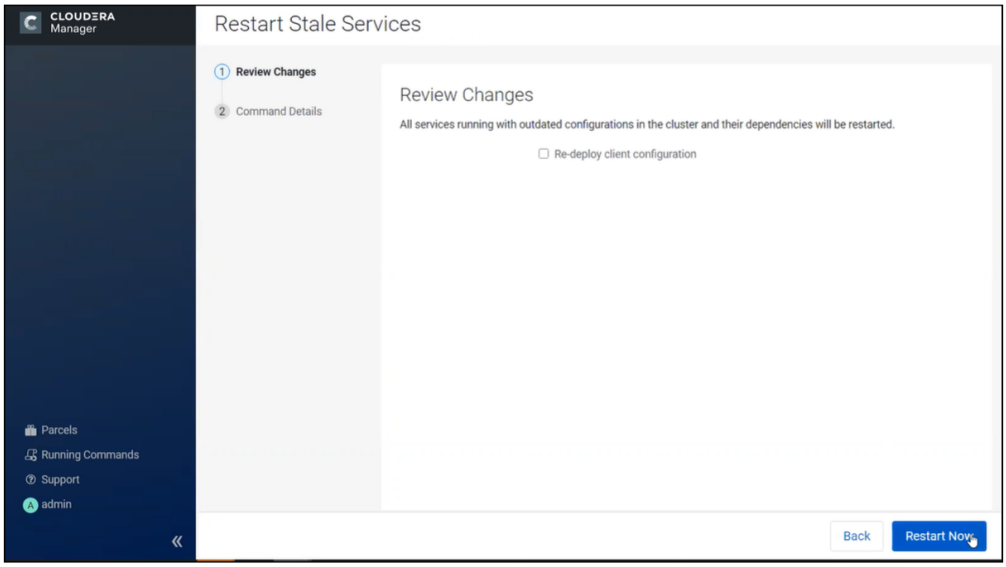


Figure 2-25: Review Changes Screen

9. Click **Restart Now**.
   The **Restart Awaiting Staleness Computation Command** screen appears.

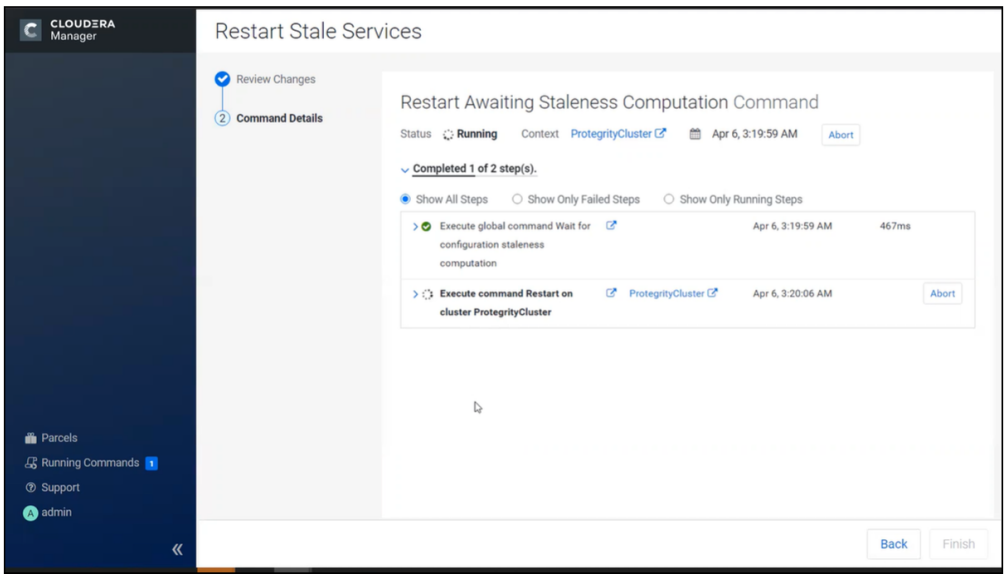


Figure 2-26: Restart Awaiting Staleness Computation Command Screen

10. Click **Finish**.

2.10.2.2 Disabling the Health Check for a CDP Cloud Platform

This section describes the procedure to disable the health check in the Big Data Protector running on a CDP cloud platform.

➤ To disable the health check:

1. Log in to the Cloudera Manager Web UI.
2. Navigate to **Clusters** > **YARN**.

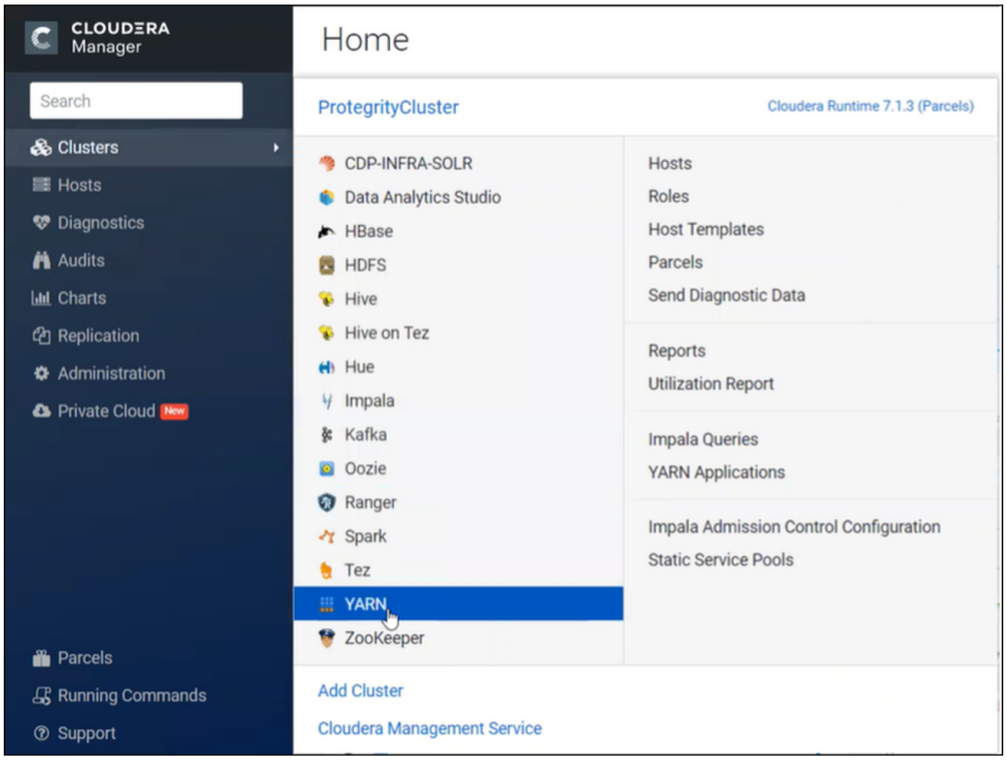


Figure 2-27: YARN Screen

The **YARN** screen appears.

3. Click the **Configuration** tab.
   The **Configuration** tab appears.

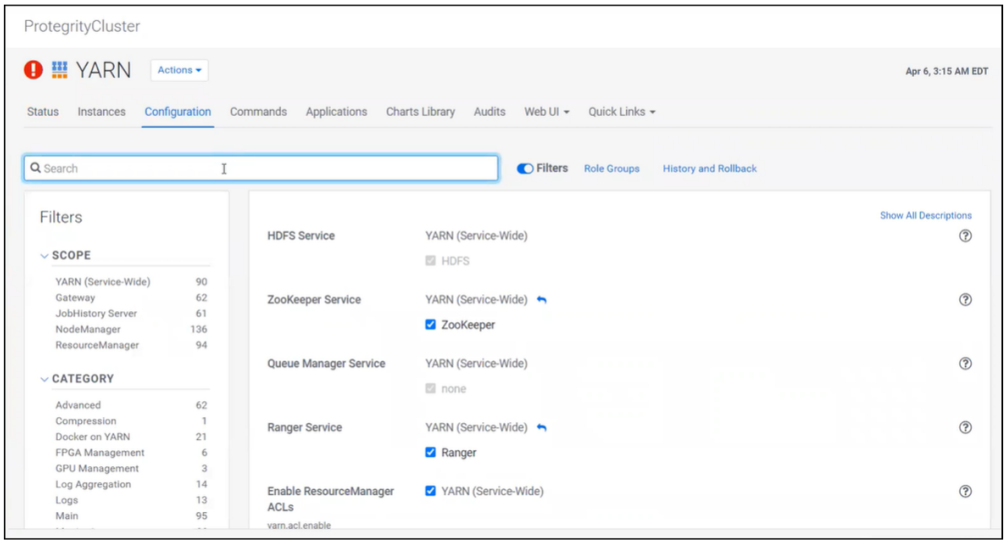


Figure 2-28: Configuration Tab

4. Navigate to the **Healthchecker Script Path** option.
   Alternatively, you can search for the health-check term in the search box.
5. From the **Healthchecker Script path** box, remove the *yarn.nodemanager.health-checker.script.path* value.
   For example, */opt/cloudera/parcels/PTY_BDP/pephealthcheck/pgreppep.sh*.

   where,

   - */opt/cloudera/parcels/* - specifies the Cloudera Parcel directory

- *PTY_BDP/pephealthcheck/pgreppep.sh* - specifies the path of the healthchecker script

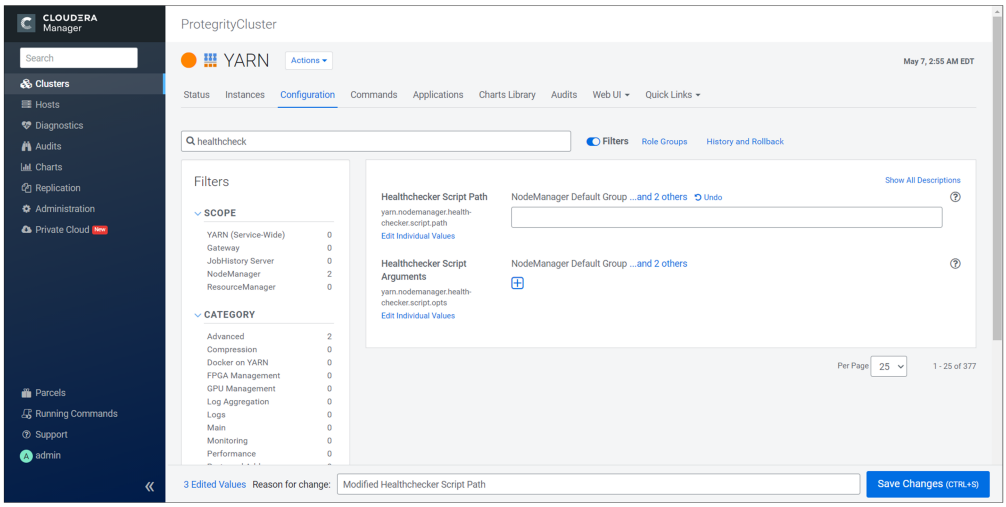


Figure 2-29: YARN Healthchecker Script path

6. Click **Save Changes**.
7. To restart the Stale Configurations, click  .
   The **Stale Configurations** screen appears.

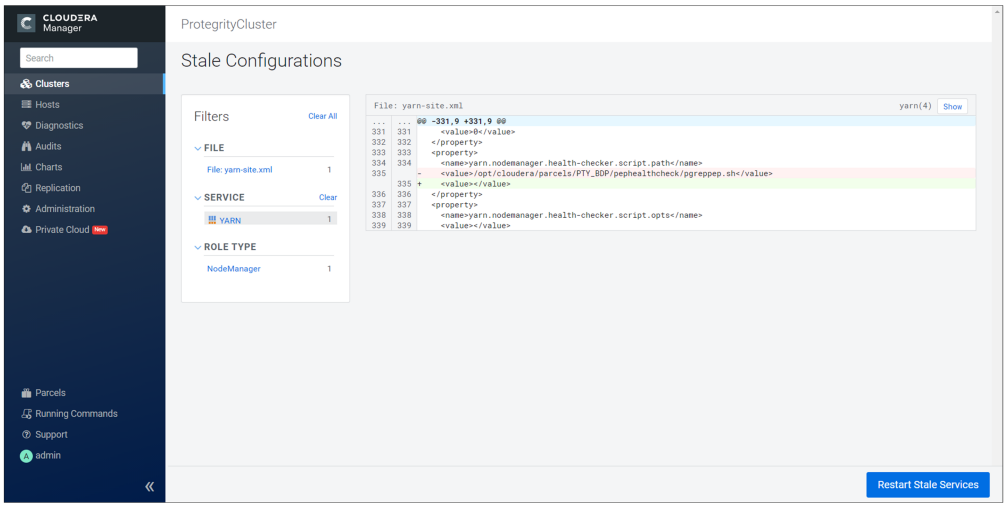


Figure 2-30: Stale Configurations Screen

8. Click **Restart Stale Services**.
   The **Review Changes** screen appears.

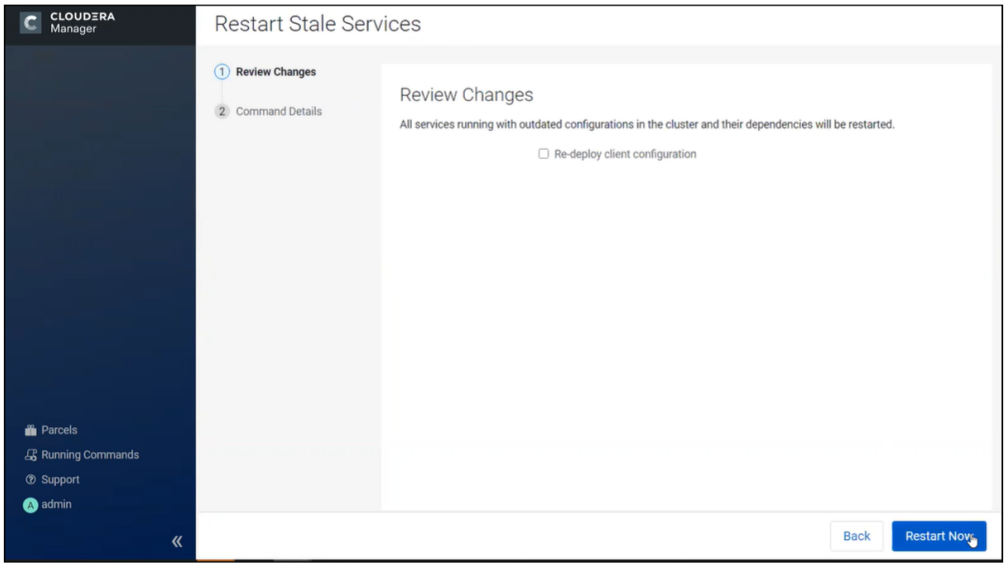


Figure 2-31: Review Changes Screen

9. Click **Restart Now**.
   The **Restart Awaiting Staleness Computation Command** screen appears.

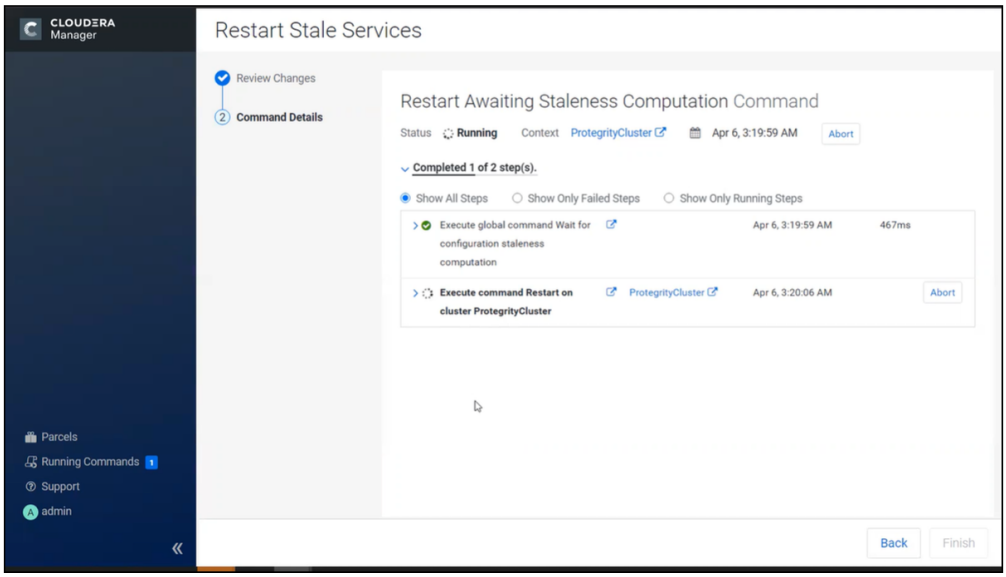


Figure 2-32: Restart Awaiting Staleness Computation Command Screen

10. Click **Finish**.

2.10.2.3 Creating a Trigger

The **Create Trigger** option in the *BDP PEP* service monitors the health condition of the PEP servers in a cluster based on the specified conditions or triggers. If the PEP server is in a bad state in a cluster, then the **Create Trigger** option returns the number of the PEP servers in bad health. You can also use the specified condition or trigger for alerting purposes.

> **Note:** For more information about the configuring alert delivery, refer to *https://docs.cloudera.com/cdp-private-cloud-base/7.1.6/monitoring-and-diagnostics/topics/cm-configuring-alert-delivery.html*.

➤ To create a trigger:

1. Navigate to **Clusters** > **BDP PEP** > **Status**.

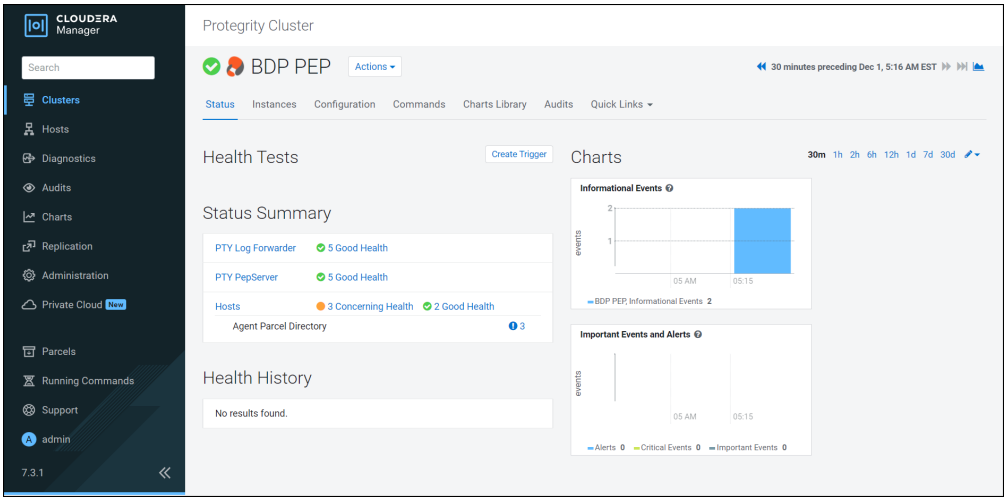


Figure 2-33: The BDP PEP Status Page

2. Click **Create Trigger**.
   The **Create New Trigger** page appears.

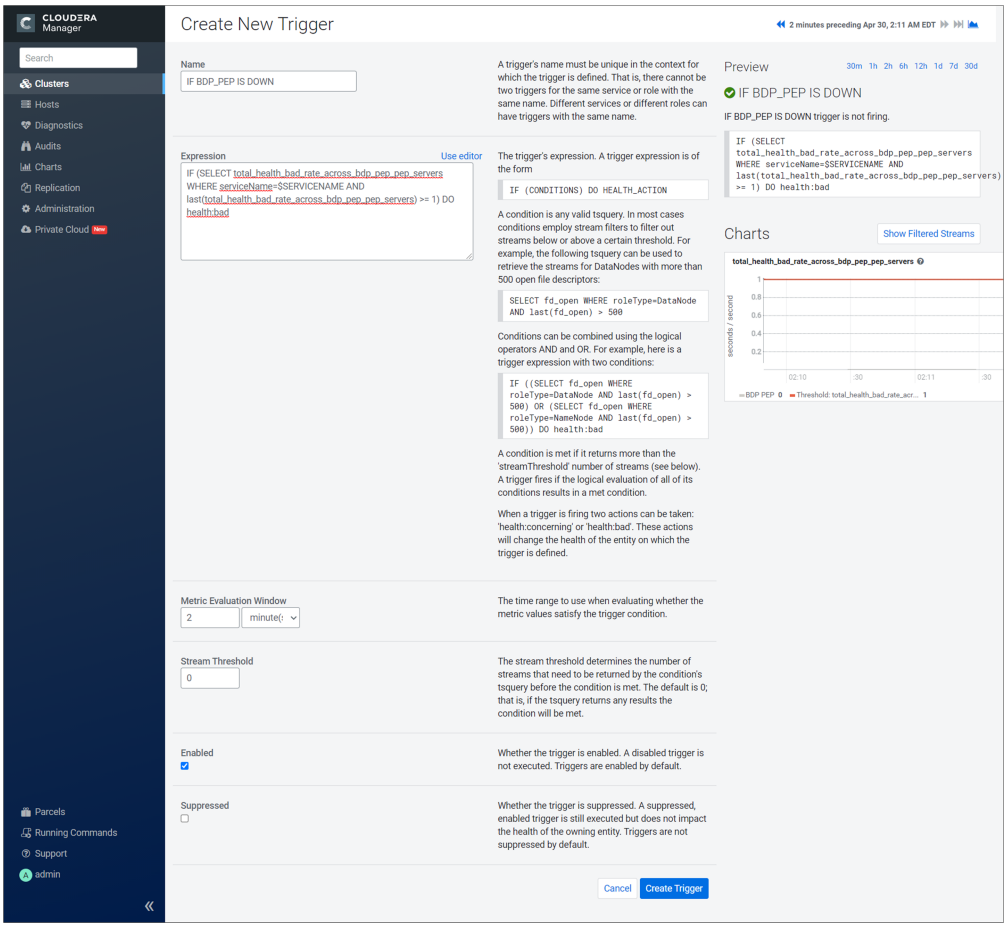


Figure 2-34: Create New Trigger Page

3. In the **Name** box, type a unique name for the trigger.
4. Set the trigger expression.

You can set the trigger expression using the **Edit Manually** option or using the **METRIC CONDITIONS**, **ATTRIBUTE CONDITIONS**, and **ACTION** options.

For example,

```
IF (SELECT total_health_bad_rate_across_bdp_pep_pep_servers WHERE
serviceName=$SERVICENAME AND last(total_health_bad_rate_across_bdp_pep_pep_servers) >= 1)
DO health:bad
```

In this example, the *total_health_bad_rate_across_bdp_pep_pep_servers* returns the total number of PEP servers in bad health.

5. Click **Create Trigger**.

2.11 Guidelines for Upgrading CDH and HDP Distributions

This section contains the guidelines for upgrading CDH and HDP distributions.

CDH distributions:

- If you are performing a rolling upgrade of the CDH distribution, then you need to uninstall Big Data Protector before starting the rolling upgrade. After the rolling upgrade of the CDH distribution is completed, you need to install the Big Data Protector version that is compatible with the updated version of the CDH distribution.
- If you are using CDH versions 5.12 and lower and need to upgrade the version of CDH, then refer to section *12.6.2.2.10 Performing an Upgrade of the CDH Distribution* in the *Installation Guide 9.1.0.0*.

HDP distributions:

- If you are using a Hortonworks distribution and need to upgrade the version of the Hortonworks distribution, then the Big Data Protector v8.1.0.0 does not participate in or hinder any rolling upgrades and remains unaffected.

2.12 Cloud Environment-specific Requirements

This section contains the cloud environment-specific features that are used for installing the Big Data Protector.

Table 2-2: Cloud Environment-specific Features used for Installation of Big Data Protector

Task	Amazon EMR	Google Dataproc	Microsoft Azure HDInsight
To save the Big Data Protector installation files	S3 Bucket	Storage Bucket	Blob Storage Location
To invoke the installation of Big Data Protector	Bootstrap Action	Initialization Action	Action Scripts
Clusters supported	Supports new and existing clusters	Supports a new cluster	Supports new and existing clusters

Chapter 3

Hadoop Application Protector

3.1 Using the Hadoop Application Protector

Various jobs written in the Hadoop cluster require data fields to be stored and retrieved. This data requires protection when it is at rest. The Hadoop Application Protector provides MapReduce, Hive and Pig the power to protect data while it is being processed and stored. Application programmers using these tools can include Protegrity software in their jobs to secure data.

For more information about using the protector APIs in various Hadoop applications and samples, refer to the following sections.

3.2 Prerequisites

Ensure that the following prerequisites are met before using Hadoop Application Protector:

- The Big Data Protector is installed and configured in the Hadoop cluster.
- The security officer has created the necessary security policy which creates data elements and user roles with appropriate permissions.

  For more information about creating security policies, data elements and user roles, refer to *Policy Management Guide 9.1.0.0*.

- The policy is deployed across the cluster.

For more information about the list of all APIs available to Hadoop applications, refer to sections *MapReduce APIs*, *Hive UDFs*, and *Pig UDFs*.

3.3 MapReduce APIs

For more information about the MapReduce APIs, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

> **Note:**

> The Protegrity MapReduce protector only supports *bytes* converted from the *string* data type.
>
> If any other data type is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

If you are using the Bulk APIs for the MapReduce protector, then the following two modes for error handling and return codes are available:

- Default mode: Starting with the Big Data Protector, version 6.6.4, the Bulk APIs in the MapReduce protector will return the detailed error and return codes instead of *0* for *failure* and *1* for *success*. In addition, the MapReduce jobs involving Bulk APIs will provide error codes instead of throwing exceptions.

  For more information about the return codes for *Big Data Protector, version 9.1.0.0*, refer to section *Appendix: Return Codes*.

- Backward compatibility mode: If you need to continue using the error handling capabilities provided with Big Data Protector, version 6.6.3 or lower, that is *0* for *failure* and *1* for *success*, then you can set this mode.

3.4 Sample Code Usage

The MapReduce sample program, described in this section, is an example on how to use the Protegrity MapReduce protector APIs. The sample program utilizes the following two Java classes:

- **ProtectData.java** – This main class calls the Mapper job.
- **ProtectDataMapper.java** – This Mapper class contains the logic to fetch the input data and store the protected content as output.

3.4.1 Main Job Class – ProtectData.java

ProtectData.java

```
package com.protegrity.samples.mapreduce;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class ProtectData extends Configured implements Tool {
       @Override
       public int run(String[] args) throws Exception
       {
    //Create the Job
       Job job = new Job(getConf(), "ProtectData");

    //Set the output key and value class
       job.setOutputKeyClass(NullWritable.class);
       job.setOutputValueClass(Text.class);

    //Set the output key and value class
       job.setMapOutputKeyClass(NullWritable.class);
       job.setMapOutputValueClass(Text.class);

    //Set the Mapper class which will perform the protect job
       job.setMapperClass(ProtectDataMapper.class);
```

```
     //Set number of reducer task
       job.setNumReduceTasks( 0 );

     //Set the input and output Format class
      job.setInputFormatClass(TextInputFormat.class);
       job.setOutputFormatClass(TextOutputFormat.class);

     //Set the jar class
       job.setJarByClass(ProtectData.class);

     //Store the input path and print the input path
       Path input = new Path(args[0]);
       System.out.println(input.getName());
     //Store the output path and print the output path
       Path output = new Path(args[1]);
       System.out.println(output.getName());

     //Add input and set output path
       FileInputFormat.addInputPath(job, new Path(args[0]));
       FileOutputFormat.setOutputPath(job, new Path(args[1]));

     //Call the job
       return job.waitForCompletion(true) ? 0 : 1;
       }

       public static void main(String args[]) throws Exception       {
              System.exit(ToolRunner.run(new Configuration(), new ProtectData(), args));
       }  }
```

3.4.2 Mapper Class – ProtectDataMapper.java

ProtectDataMapper.java

```
package com.protegrity.samples.mapreduce;

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
//Need to import the ptyMapReduceProtector class to use the Protegrity MapReduce protector
import com.protegrity.hadoop.mapreduce.ptyMapReduceProtector;

//Create the Mapper class i.e. ProtectDataMapper which will extends the Mapper Class
public class ProtectDataMapper extends Mapper<Object, Text, NullWritable, Text> {

    //Declare the member variable for the ptyMapReduceProtector class
    private ptyMapReduceProtector mapReduceProtector;
    //Declare the Array of Data Elements which will be required to do the protection/
unprotection
    private final String[] data_element_names = { "TOK_NAME", "TOK_PHONE", "TOK_CREDIT_CARD",
"TOK_AMOUNT" };

    //Initialize the mapreduce protector i.e ptyMapReduceProtector in the default constructor
    public ProtectDataMapper() throws Exception {
        // Create the new object for the class ptyMapReduceProtector
        mapReduceProtector = new ptyMapReduceProtector();
        // Open the session using the method " openSession("0") "
        int openSessionStatus = mapReduceProtector.openSession("0");
    }

    //Override the map method to parse the text and process it line by line
    //Split the inputs separated by delimiter "," in the line
    //Apply the protect/unprotect operation
    //Create the output text which will have protected/unprotected outputs separated by
delimiter ","
    //Write the output text to the context
    @Override
    public void map(Object key, Text value, Context context) throws IOException,
            InterruptedException
    {
        // Store the line in a variable strOneLine
```

```
        String strOneLine = value.toString();
        // Split the inputs separated by delimiter "," in the line
        StringTokenizer st = new StringTokenizer(strOneLine, ",");
        // Create the instance of StringBuilder to store the output
        StringBuilder sb = new StringBuilder();
        // Store the no of inputs in a line
        int noOfTokens = st.countTokens();
        if (mapReduceProtector != null) {
            //Iterate through the string token and apply the protect/unprotect operation
            for (int i = 0; st.hasMoreElements(); i++) {
                String data = (String)st.nextElement();
                if(i == 0) {
                    sb.append(new String(data));
                } else {
                    //To protect data, call the function protect method with parameters data
element and input data in bytes
                    //mapReduceProtector.protect( <Data Element> , <Data in bytes> )
                    //Output will be returned in bytes
                    //To unprotect data, call the function unprotect method with parameters
data element and input data in bytes
                    //mapReduceProtector.unprotect( <Data Element> , <Data in bytes> )
                    //Output will be returned in bytes
                    byte[] bResult =
                            mapReduceProtector.protect(data_element_names[i-1],
data.trim().getBytes());
                    if (bResult != null) {
                        // Store the result in string and append it to the output sb
                        sb.append(new String(bResult));
                    }
                    else {
                        // If output will be null, then store the result as "cryptoError" and
append it to the output sb
                        sb.append("cryptoError");
                    }
                }
                if(i < noOfTokens -1 ) {
                    // Append delimiter "," at the end of the processed result
                    sb.append(",");
                } } }
        // write the output text to context
        context.write(NullWritable.get(), new Text(sb.toString()));
    }

    // call flushAudits() in the cleanup method of Mapper so that all Protector
    // audit logs are flushed to Audit Store at the end of each Mapper task
    @Override
    public void cleanup(Mapper.Context context){
        mapReduceProtector.flushAudits();
    }

    //clean up the session and objects
    @Override
    protected void finalize() throws Throwable {
        //Close the session
        int closeSessionStatus = mapReduceProtector.closeSession();
        mapReduceProtector = null;
        super.finalize();
    }
}
```

3.5 Hive UDFs

For more information about the Hive User Defined Functions (UDFs), refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

> **Note:**

> If you are using Ranger or Sentry, then ensure that your policy provides *create* access permissions to the required UDFs.

3.6 Pig UDFs

For more information about the Pig UDFs, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

Chapter 4

HDFS File Protector (HDFSFP) (Deprecated from Big Data Protector 7.2.0)

Note:
Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.1.0.0.

4.1 Overview of HDFSFP

The files stored in HDFS are plain text files and can be accessed with a POSIX-based file system access control. These files may contain sensitive data which is vulnerable with exposure to unwanted users.

The HDFS File Protector (HDFSFP) helps to transparently protect these files as they are stored into HDFS and allow only authorized users to access the content in the files.

4.2 Features of HDFSFP

The following are the features of HDFSFP:

- Protects and stores files in HDFS and retrieves the protected files in the clear from HDFS, as per centrally defined security policy and access control.
- Stores and retrieves from HDFS transparently for the user, depending upon their access control rights.
- Preserves Hadoop distributed data processing ensuring that protected content is processed on data nodes independently.
- Blocks the addressing of files by the defined access control pass-through, transparently without any protection or unprotection.
- Protects temporary data, such as intermediate files generated by the MapReduce job.
- Provides recursive access control for HDFS directories and files. Protects directories, its subdirectories and files, as per defined security policy and access control.
- Protects files at rest so that unauthorized users can view only the protected content.
- Adds minimum overhead for data processing in HDFS.
- Can be accessed using the command shell and Java API.

4.3 Protector Usage

Files stored in HDFS are plain text files. Access controls for HDFS are implemented by using file-based permissions that follow the UNIX permissions model. These files may contain sensitive data, making it vulnerable when exposed to unwanted users. These files should be transparently protected as they are stored into HDFS and the content should be exposed only to authorized users.

The files are stored and retrieved from HDFS using Hadoop ecosystem products, such as file shell commands, MapReduce jobs, and so on.

Any user or application with write access to protected data at rest in HDFS can delete, update or move the protected data. Hence though the protected data can be lost, the data is not compromised as the user or application do not access the original data in the clear. Ensure that the Hadoop administrator assigns file permissions in HDFS cautiously.

4.4 File Recover Utility

The File Recover utility recovers the contents from a protected file.

For more information about the File Recover Utility, refer to section *Recover Utility (Deprecated from Big Data Protector 7.2.0)*.

4.5 HDFSFP Commands

Hadoop provides shell commands for modifying and administering HDFS. HDFSFP extends the modification commands to control access to files and directories in HDFS.

For more information about the commands supported in HDFSFP, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

4.6 Ingesting Files Securely

To ingest files into HDFS securely, use the *put* and *CopyFromLocal* commands.

For more information, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

If you need to ingest data to a protected ACL in HDFS using Sqoop, then use the *-D target.output.dir* parameter before any tool-specific arguments as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --driver com.mysql.jdbc.Driver --connect
"jdbc:mysql://master.localdomain/test" --username root --table test --target-dir /tmp/src -m 1
```

In addition, if you need to add additional data to any existing data, then use the *--append* parameter as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --driver com.mysql.jdbc.Driver --connect
"jdbc:mysql://master.localdomain/test" --username root --table test --target-dir /tmp/src -m
1 --append
```

4.7 Extracting Files Securely

To extract files from HDFS securely, use the *get* and *CopyToLocal* commands.

For more information, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

4.8 HDFSFP Java API

Protegrity provides a Java API for working with files and directories using HDFSFP. The Java API for HDFSFP provides an alternate means of working with HDFSFP besides the HDFSFP shell commands, **hadoop ptyfs**, and enables you to integrate HDFSFP with Java applications.

For more information about the commands supported in HDFSFP, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

4.9 Developing Applications using HDFSFP Java API

This section describes the guidelines to follow when developing applications using the HDFSFP Java API.

> **Note:**
> The guidelines described in this section are a sample and assumes that */opt/protegrity* is the base installation directory of Big Data Protector. These guidelines would need to be modified based on your requirements.

4.9.1 Setting up the Development Environment

Ensure that the following steps are completed before you begin to develop applications using the HDFSFP Java API:

- Add the required HDFSFP Java API jar, *hdfsfp-x.x.x.jar*, in the Classpath.
- Instantiate the HDFSFP Java API function using the following command:

  ```
  PtyHdfsProtector protector = new PtyHdfsProtector();
  ```

  After successful instantiation, you are ready to call the HDFSFP Java API functions.

4.9.2 Protecting Data using the Class file

To protect data using the Class file:

1. Compile the Java file to create a Class file with the following command.

```
javac -cp .:<PROTEGRITY_DIR>/hdfsfp/hdfsfp-x.x.x.jar ProtectData.java -d.
```

2. Protect data using the Class file with the following command.

```
hadoop ProtectData
```

4.9.3 Protecting Data using the JAR file

To protect data using the JAR file:

1. Compile the Java file to create a Class file with the following command.

```
javac -cp .:<PROTEGRITY_DIR>/hdfsfp/hdfsfp-x.x.x.jar ProtectData.java -d.
```

2. Create the JAR file from the Class file with the following command.

```
jar -cvf protectData.jar ProtectData
```

3. Protect data using the JAR file with the following command.

```
hadoop jar protectData.jar ProtectData
```

4.9.4 Sample Program for the HDFSFP Java API

HDFSFP Java API Sample Program

```
public class ProtectData {
public static PtyHdfsProtector protector = new PtyHdfsProtector();

public void copyFromLocalTest(String[] srcs, String dstf)
  {
    boolean result;

    try {
      result = protector.copyFromLocal(srcs, dstf);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void copyToLocalTest(String srcs, String dstf)
  {
    boolean result;

    try {
      result = protector.copyToLocal(srcs, dstf);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void copyTest(String srcs, String dstf)
  {
    boolean result;

    try {
      result = protector.copy(srcs, dstf);
```

```
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void mkdirTest(String dir)
  {
    boolean result;

    try {
      result = protector.mkdir(dir);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void moveTest(String srcs, String dstf)
  {
    boolean result;

    try {
      result = protector.move(srcs,dstf);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void deleteFileTest(String file,boolean skipTrash)
  {
    boolean result;

    try {
      result = protector.deleteFile(file, skipTrash);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public void deleteDirTest(String dir,boolean skipTrash)
  {
    boolean result;

    try {
      result = protector.deleteDir(dir, skipTrash);
    } catch (ProtectorException pe) {
      pe.printStackTrace();
    }
  }
  public static void main(String[] args) {
    ProtectData protect = new ProtectData();

    // Ingest Local Data into HDFS
    String srcsCFL[] = new String[2];
    srcsCFL[0] ="<Local source file location1>";
    srcsCFL[1] ="<Local Source file location2>";
    String dstfCFL ="<ACL activated HDFS destination directory location>";
    protect.copyFromLocalTest(srcsCFL, dstfCFL);

    // Extract HDFS file to Local
    String srcsCTL= "<ACL activated HDFS source file location>";
    String dstfCTL = "<Local destination directory location >";
    protect.copyToLocalTest(srcsCTL, dstfCTL);

    // Copy File from HDFS to HDFS
    String srcsCopy="<ACL activated HDFS source file location>";
    String dstfCopy ="<ACL activated HDFS destination directory location>";
    protect.copyTest(srcsCopy, dstfCopy);

    // Create HDFS Sub-Directory
    String dir = "<HDFS directory location>";
    protect.mkdirTest(dir);

    // Move from HDFS to HDFS
    String srcsMove = "<ACL activated HDFS source file location>";
    String dstfMove = "<ACL activated HDFS destination directory location>";
    protect.moveTest(srcsMove, dstfMove);
```

```
    // Delete File from HDFS
    String fileDelete = "<HDFS file location>";
    boolean skipTrashFile = false;
    protect.deleteFileTest(fileDelete,skipTrashFile);

    // Delete Sub-Directory and Children from HDFS
    String dirDelete = "<HDFS directory location>";
    boolean skipTrashDir = false;
    protect.deleteDirTest(dirDelete,skipTrashDir);
  }
}
```

4.10 Quick Reference Tasks

This section provides a quick reference for the tasks that can be performed by users.

4.10.1 Protecting Existing Data

The *dfsadmin* utility protects existing data after creating ACL for the HDFS path. It is a two-step process. In first step, the user creates new ACL entries. In the second step, the user will activate the newly created ACL entries. After activation, all ACL entries will be protected automatically.

The steps for activating ACL entries can be done for single or multiple entries. After activating ACL entries, the HDFSFP infrastructure protects the HDFS path in the ACL entry.

While installing HDFSFP, you need to configure the ingestion user in the *BDP.config* file. The HDFS administrator would have to ensure that the ingestion user has full access to the directories that would be protected with HDFSFP. This user would be the authorized user for protection. Permissions to protect or create are configured in the security policy. After the *dfsadmin* utility activates ACL entry using the preconfigured ingested user, the HDFS File Protector protects the ACL path.

For more information about adding and activating an ACL entry, refer to sections *Adding an ACL Entry for Protecting Directories in HDFS* and *Activating Inactive ACL Entries*.

4.10.2 Reprotecting Files

For information about reprotecting files, refer to section *Reprotecting Files or Directories*.

4.11 Appliance components of HDFSFP

This section describes the active components, shipped with the ESA and required to run HDFSFP.

4.11.1 Dfsdatastore Utility

This utility adds the Hadoop cluster under protection for HDFSFP.

4.11.2 Dfsadmin Utility

This utility handles management of access control entries for files and directories.

4.12 Access Control Rules for Files and Directories

The rules for files and directories that are stored or accessed in HDFS are managed by Access Control Lists (ACLs). The protection of HDFS files and directories is done after the ACL entry has been created. ACLs for multiple Hadoop clusters can be managed only from the ESA. Protegrity Cache is used to store or propagate secured ACLs across the clusters.

If you need to add, delete, search, update, or list a cluster, then use the DFS Cluster Management Utility (*dfsdatastore*).

If you need to protect, unprotect, reprotect, activate, search, or update ACLs, or get information about a job, then use the ACL Management Utility (*dfsadmin)*.

For more information about managing access control entries across clusters, refer to sections *Using DFS Cluster Management Utility (dfsdatastore)* and *Using the ACL Management Utility (dfsadmin)*.

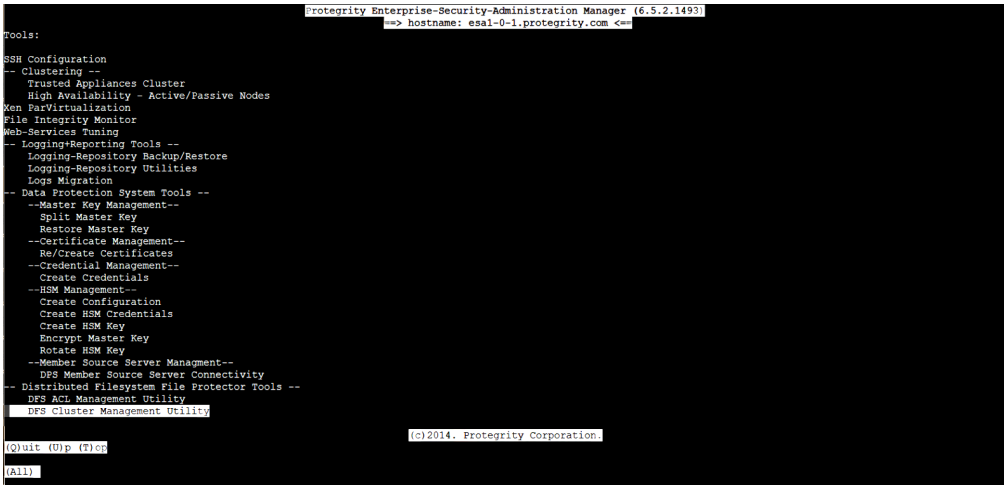
4.13 Using DFS Cluster Management Utility (dfsdatastore)

The *dfsdatastore* utility enables you to manage the configuration of clusters on the ESA. The details of all options supported by this utility are described in this section.

4.13.1 Adding a Cluster for Protection

To add a cluster for protection using the *dfsdatastore* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.

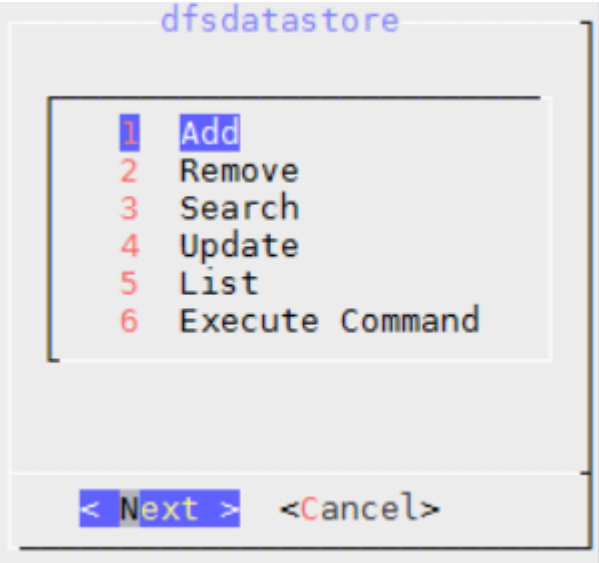


3. Press **ENTER**.

   The *root* password screen appears.

4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsdatastore* UI appears.



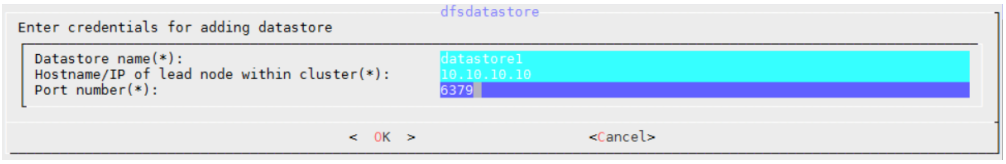
6. Select the option *Add*.
7. Select **Next**.
8. Press **ENTER**.

   The *dfsdatastore* credentials screen appears.

9. Enter the following parameters:
   - Datastore name – The name for the datastore or cluster.

     This name will be used for managing ACLs for the cluster.

   - Hostname/IP of the Lead node within the cluster – The hostname or IP address of the Lead node of the cluster.
   - Port number – The Protegrity Cache port which was specified in the *BDP.config* file during installation.



10. Select **OK**.
11. Press **ENTER**.

    The cluster with the specified parameters is added.

12. If the *DfsCacheRefresh* service is already running, then the datastore is added in an activated state.

    If the *DfsCacheRefresh* service is not running, then the datastore is added in an inactive state. The datastore can be activated by starting the *DfsCacheRefresh* service.

4.13.1.1 Start the DfsCacheRefresh Service

➤ To start the DfsCacheRefresh Service:

1. Login to the ESA Web UI.

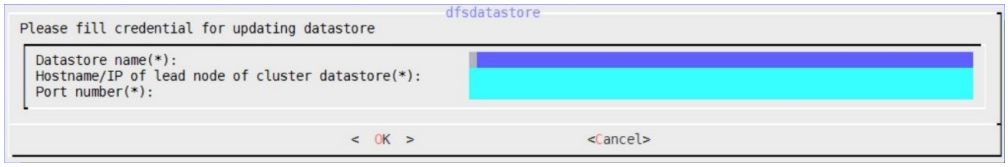
2. Navigate to **System** > **Services**.
3. Start the *DfsCacheRefresh* service.

4.13.2 Updating a Cluster

> **Note:** Ensure that you utilize the **Update** option in the *dfsdatastore* UI to modify the parameters of an existing datastore only.

➤ To update a cluster using the *dfsdatastore* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password
5. Press **ENTER**.
   The *dfsdatastore* UI appears.
6. Select the option *Update*.
7. Select **Next**.
8. Press **ENTER**.
   The *dfsdatastore* update screen appears.
9. Update the following parameters as required:
   - Hostname/IP of the Lead node within the cluster – The hostname or IP address of the Lead node of the cluster.
   - Port number – The Protegrity Cache port which was specified in the *BDP.config* file during installation.



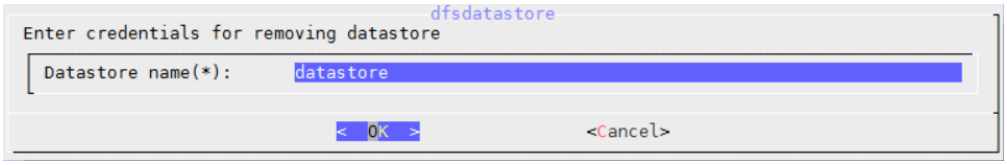
10. Select **OK**.
11. Press **ENTER**.
    The cluster is modified with the required updates.

4.13.3 Removing a Cluster

> **Note:** Ensure that the *Cache Refresh Service* is running in the ESA Web UI before removing a cluster.

➤ To remove a cluster using the *dfsdatastore* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsdatastore* UI appears.
6. Select the option *Remove*.
7. Select **Next**.
8. Press **ENTER**.
   The *dfsdatastore* remove screen appears.
9. Enter the following parameter:
   - Datastore name – The name for the datastore or cluster.
     This name will be used for managing ACLs for the cluster.



10. Select **OK**.
11. Press **ENTER**.
    The required cluster is removed.

4.13.4 Monitoring a Cluster

To monitor a cluster using the *dfsdatastore* UI from the ESA CLI Manager:

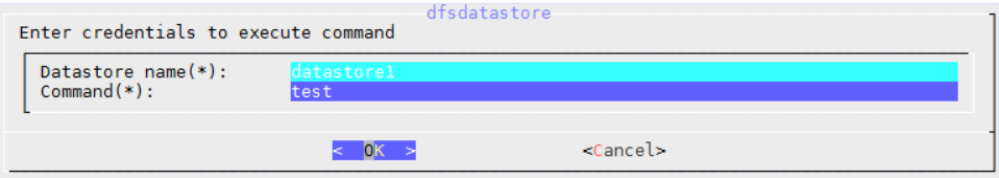
1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsdatastore* UI appears.
6. Select the option *Execute Command*.
7. Select **Next**.
8. Press **ENTER**.

The *dfsdatastore* execute command screen appears.

9. Enter the following parameters:
   - Datastore name – The name of the datastore or cluster.

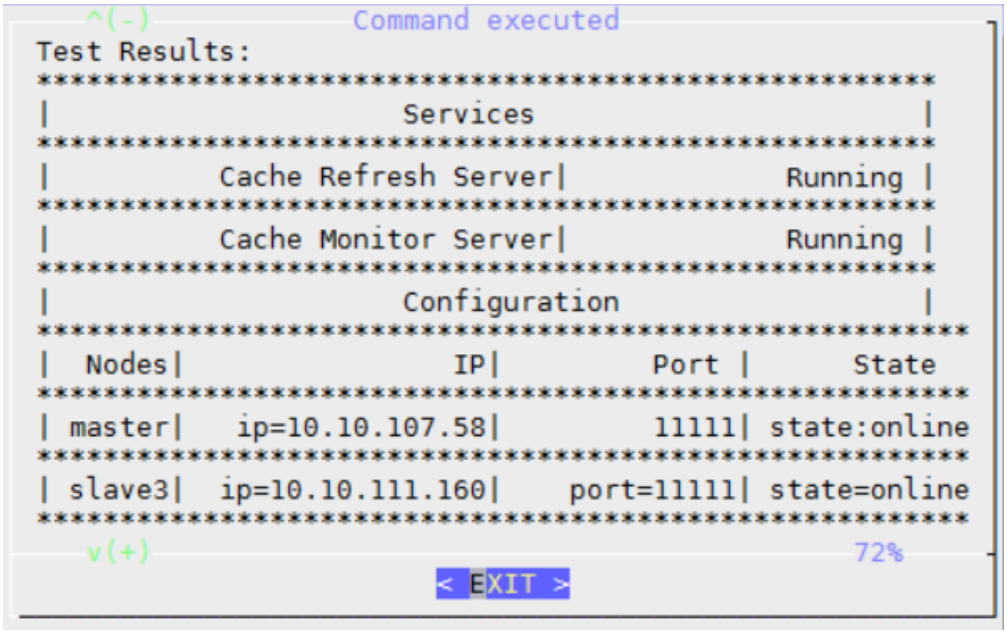
     This name will be used for managing ACLs for the cluster.

   - Command - The command to execute on the datastore. In this release, the only command supported is TEST. The command TEST is executed on the cluster, which is used to retrieve the statuses of the following servers:
     - Cache Refresh Server, running on the ESA
     - Cache Monitor Server, running on the Lead node of the cluster
     - Distributed Cache Server, running on the Lead and slave nodes of the cluster



10. Select **OK**.
11. Press **ENTER**.

    The *dfsdatastore* UI executes the TEST command on the cluster.



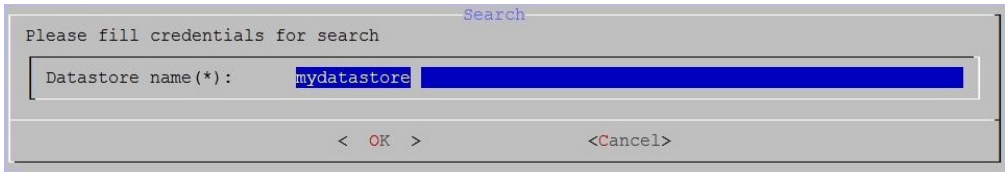
4.13.5 Searching a Cluster

To search for a cluster using the *dfsdatastore* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

The *root* password screen appears.

4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsdatastore* UI appears.
6. Select the option *Search*.
7. Select **Next**.
8. Press **ENTER**.
   The *dfsdatastore* search screen appears.
9. Enter the following parameter:
   - Datastore name – The name of the datastore or cluster.
     This name will be used for managing ACLs for the cluster.



10. Select **OK**.
11. Press **ENTER**.
    The *dfsdatastore* UI searches for the required cluster.

4.13.6 Listing all Clusters

To list all clusters using the *dfsdatastore* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsdatastore* UI appears.
6. Select the option *List*.
7. Select **Next**.
8. Press **ENTER**.
   The *dfsdatastore* search screen appears.

   A list of all the clusters appears. Each cluster description contains one of the following cluster statuses:
   - 1: Cluster is in active state

- 0: Cluster is in inactive state

4.14 Using the ACL Management Utility (dfsadmin)

The *dfsadmin* utility enables you to manage ACLs for cluster. Managing ACLs is a two-step process, which is creating or modifying ACL entries and then activating it. The protection of file or directory paths will not take effect until ACL entries are verified, confirmed and activated.

> **Note:**
> Ensure that an unstructured policy is created in the ESA, which is to be linked with the ACL.

The details of all options supported by this utility are described in this section.

4.14.1 Adding an ACL Entry for Protecting Directories in HDFS

> **Note:** It is recommended to not create ACLs for file paths.

> **Note:** If the ACL for the directory, which is containing a file for which an ACL already exists, is being unprotected, then a decryption failure might occur, if there is a mismatch between the data elements used for the protection of the directory and the file contained in the directory.

➤ To add an ACL entry for protecting files or directories using the *dfsadmin* UI from the ESA CLI Manager:

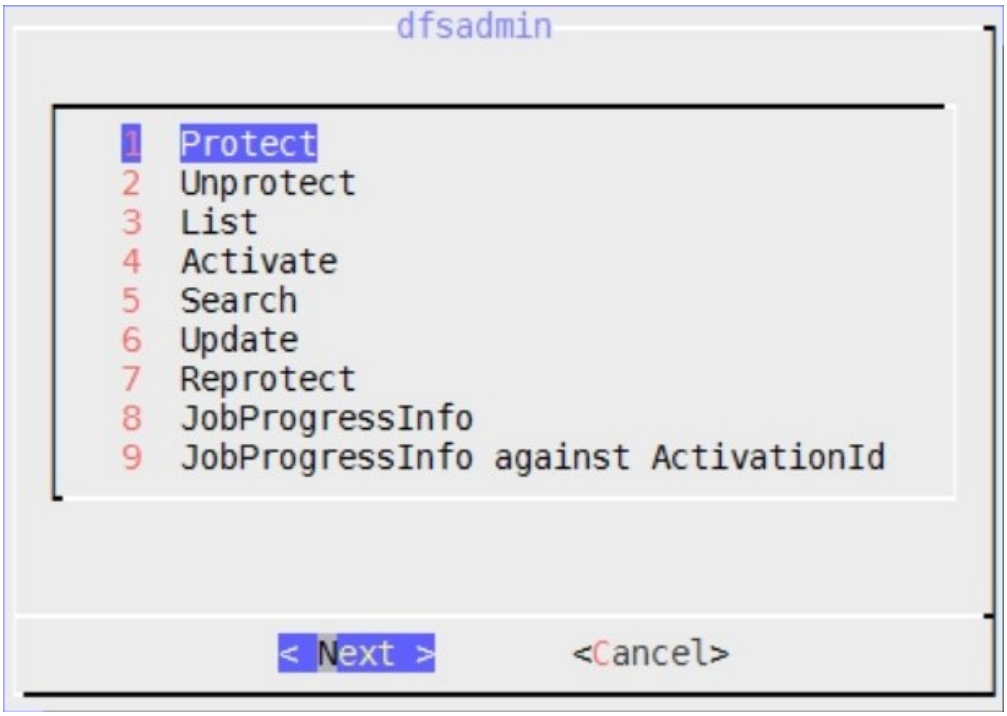
1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.

3. Press **ENTER**.

   The *root* password screen appears.

4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

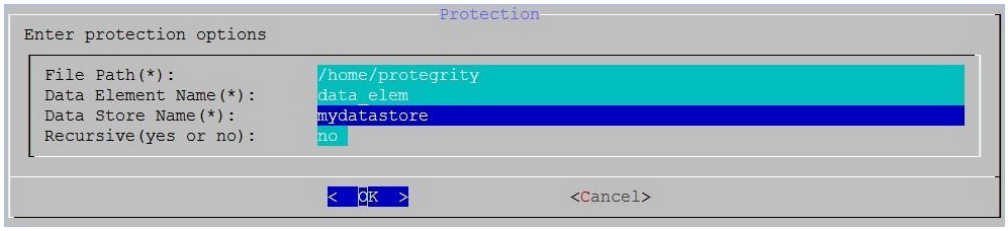


6. Select the option *Protect*.
7. Select **Next**.
8. Press **ENTER**.

   The *dfsadmin* protection screen appears.

9. Enter the following parameters:
   - File Path – The directory path to protect.

- Data Element Name – The unstructured data element name to protect the HDFS directory path with.
- Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.
- Recursive (yes or no) – Select one of the following options:
  - Yes – Protect all files, sub-directories and their files, in the directory path.
  - No – Protect the files in the directory path.



10. Select **OK**.
11. Press **ENTER**.

The ACL entry required for protecting the directory path is added to the *Inactive* list. The ACL entries can be activated by selecting the *Activate* option.

After the ACL entries are activated, the following actions occur, as required:

- If the recursive flag is not set, then all files inside the directory path are protected.
- If the recursive flag is set, then all the files, sub-directories and its files, in the directory path are protected.

If any MapReduce jobs or HDFS file shell commands are initiated on the ACL paths before the ACLs are activated, then the jobs or commands will fail. After the ACLs are activated, any new files that are ingested in the respecticve ACL directory path will get protected.

4.14.2 Updating an ACL Entry

➤ To update an ACL entry using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

   The *root* password screen appears.

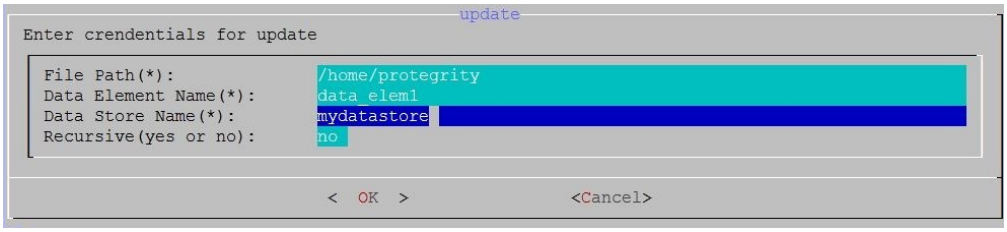
4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

6. Select the option *Update*.
7. Select **Next**.
8. Press **ENTER**.

The *dfsadmin* update screen appears.

9. Enter the following parameters as required:
   - File Path – The directory path to protect.
   - Data Element Name – The unstructured data element name to protect the HDFS directory path with.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.
   - Recursive (yes or no) – Select one of the following options:
     - Yes – Protect all files, sub-directories and their files, in the directory path.
     - No – Protect the files in the directory path.



10. Select **OK**.
11. Press **ENTER**.

    The ACL entry is updated as required.

4.14.3 Reprotecting Files or Directories

To reprotect files or directories using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

   The *root* password screen appears.

4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

6. Select the option *Reprotect*.
7. Select **Next**.
8. Press **ENTER**.

   The *dfsadmin* reprotection screen appears.

9. Enter the following parameters:
   - File Path – The directory path to protect.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.
   - Data Element Name – The unstructured data element name to protect the HDFS directory path with.

If the user has rotated the data element key and needs to reprotect the data, then this field is optional.

10. Select **OK**.
11. Press **ENTER**.

    The files inside the ACL entry are reprotected.

4.14.4 Deleting an ACL Entry to Unprotect Files or Directories

To delete an ACL entry to unprotect files or directories using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

   The *root* password screen appears.

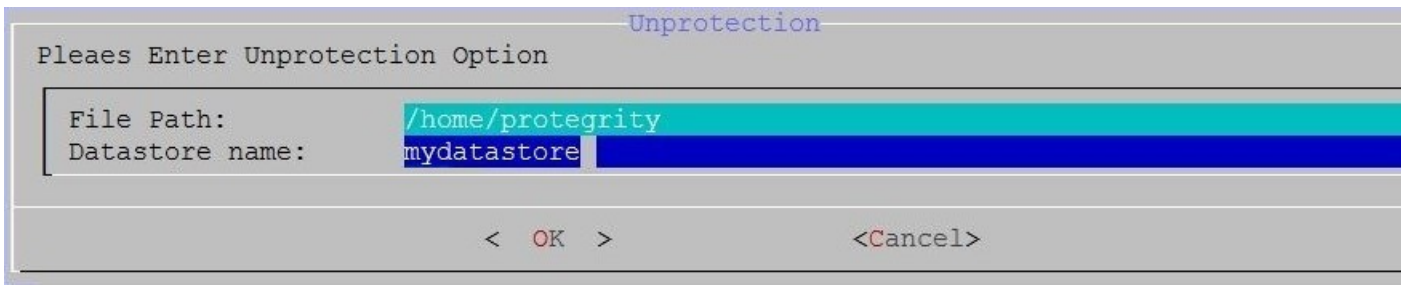
4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

6. Select the option *Unprotect*.
7. Select **Next**.
8. Press **ENTER**.

   The *dfsadmin* reprotection screen appears.

9. Enter the following parameters:
   - File Path – The directory path to protect.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press **ENTER**.

The files inside the ACL entry are unprotected.

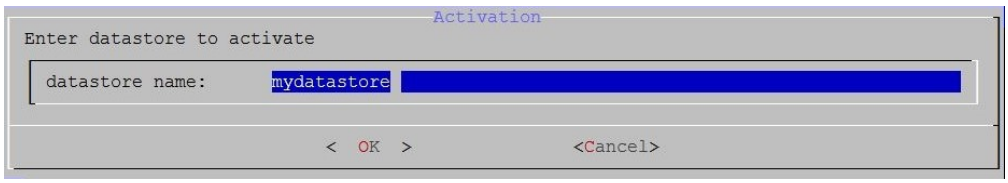
4.14.5 Activating Inactive ACL Entries

> **Note:** If you are using a Kerberos-enabled Hadoop cluster, then ensure that the user *ptyitusr* has a valid Kerberos ticket and write access permissions on the HDFS path for which the ACL is being created.

4.14.5.1 Activating Inactive ACL entries using the *dfsadmin* UI

➤ To activate inactive ACL entries using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsadmin* UI appears.
6. Select the option *Activate*.
7. Select **Next**.
8. Press **ENTER**.
   The *dfsadmin* activation screen appears.
9. Enter the following parameters as required:
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press **ENTER**.
    The inactive ACL entries in the Datastore are activated.

> **Note:**
> If an ACL entry for a directory containing files is activated (for Protect/Unprotect/Reprotect), then the ownerships and permissions for only the files contained in the directory are changed.

> **Note:**

To avoid this issue, ensure that the user configured in the *PROTEGRITY_IT_USR* property in the *BDP.config* file during the Big Data Protector installation is added to the HDFS superuser group by running the following command on the Lead node:

```
usermod -a -G hdfs <User_configured_in_the_“PROTEGRITY_IT_USR”  property>
```

Note:

If the protect or unprotect operation fails on the files or directories, which are a part of the ACL entry being activated and the message *ACL is locked* appears, then ensure that you monitor the *beuler.log* file for any exceptions and take the required corrective action.

4.14.5.2 Monitoring the *beuler.log* file

➤ To monitor the beuler.log file:

1. Login to the Lead node with *root* permissions.
2. Switch the user to PROTEGRITY_IT_USR, as configured in the *BDP.config* file.
3. Navigate to the *<PROTEGRITY_DIR>/hdfsfp/ptyitusr* directory.
4. Monitor the *beuler.log* file for any exceptions.
5. If any exceptions appear in the *beuler.log* file, then resolve the exceptions as required.
6. Login to the Lead node and run the *beuler.sh* script.

   The following is a sample *beuler.sh* script command.

   ```
   sh beuler.sh -path <ACL_directory_path> -datastore <datastore_name> -activationid
   <activation_ID> -beulerjobid <beuler_job_ID>
   ```

   Alternatively, you can restart the *DfsCacheRefresh* service.

4.14.5.3 Restarting the DfsCacheRefresh Service

➤ To restart the DfsCacheRefresh Service:

1. Login to the ESA Web UI.
2. Navigate to **System** > **Services**.
3. Restart the *DfsCacheRefresh* service.

4.14.6 Viewing the ACL Activation Job Progress Information in the Interactive Mode

➤ To view the ACL Activation Job Progress Information in the Interactive mode using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

   The *root* password screen appears.

4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

6. Select the option *JobProgressInfo*.
7. Select **Next**.
8. Press **ENTER**.

   The *Activation ID* screen appears.

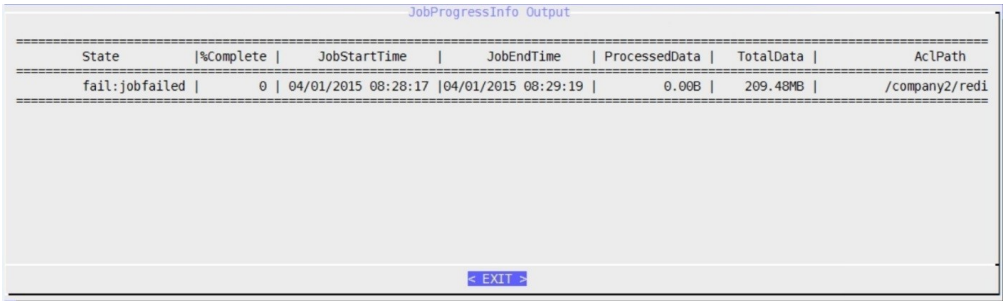
9. Enter the Activation ID.
10. Press **ENTER**.

    The filter search criteria screen appears.

11. If you need to specify the filtering criteria, then perform the following steps.
    a. Type *Y* or *y*.
    b. Select one of the following filtering criteria:
       - Start Time
       - Status
       - ACL Path
12. Select **Next**.
13. Press **ENTER**.
14. If you do not need to specify the search criteria, then type *N* or *n*.

    The *dfsadmin* job progress information screen appears listing all the jobs against the required Activation ID with the following information:

    - State: One of the following states of the job:
      - Started
      - Failed
      - In progress
      - Completed
      - Yet to start
      - Failed as Path Does not Exist
    - Percentage Complete: The percentage completion for the directory encryption
    - Job Start Time: The time when the directory encryption started
    - Job End Time: The time when the directory encryption ended
    - Processed Data: The amount of data that is encrypted
    - Total Data: The total directory size being encrypted
    - ACL Path: The directory path being encrypted



4.14.7 Viewing the ACL Activation Job Progress Information in the Non Interactive Mode

To view the ACL Activation Job Progress Information in the Non Interactive mode using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.
   The *dfsadmin* UI appears.
6. Select the option *JobProgressInfo* against *ActivationId*.
   The *JobProgressInfo* Activation ID screen appears.
7. Enter the Activation ID.
8. Select **OK**.
9. Press **ENTER**.
   The *dfsadmin* job progress information screen for the required Activation ID appears.

4.14.8 Searching ACL Entries

To search for ACL entries using the *dfsadmin* UI from the ESA CLI Manager:

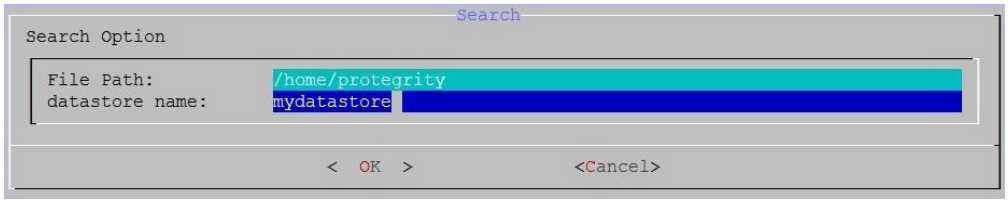
1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.
   The *root* password screen appears.
4. Enter the *root* password.
5. Press **ENTER**.

The *dfsadmin* UI appears.

6. Select the option *Search*.
7. Select **Next**.
8. Press **ENTER**.

   The *dfsadmin* search screen appears.

9. Enter the following parameters as required:
   - File Path – The directory path to protect.
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press **Enter**.

    The *dfsadmin* UI searches for the required ACL entry.

4.14.9 Listing all ACL Entries

➤ To list ACL entries using the *dfsadmin* UI from the ESA CLI Manager:

1. Login to the ESA CLI Manager.
2. Navigate to **Tools** > **DFS Cluster Management Utility**.
3. Press **ENTER**.

   The *root* password screen appears.

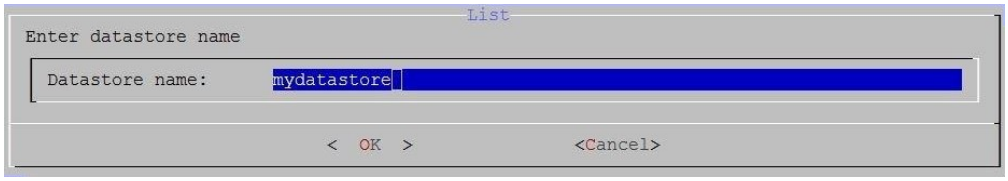
4. Enter the *root* password.
5. Press **ENTER**.

   The *dfsadmin* UI appears.

6. Select the option *List*.
7. Select **Next**.
8. Press **ENTER**.

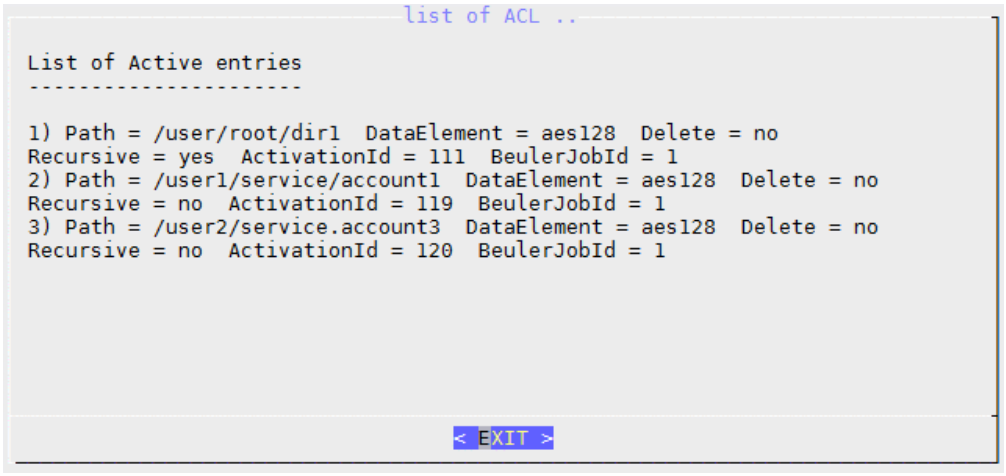
   The *dfsadmin* search screen appears.

9. Enter the following parameters as required:
   - Datastore name – The datastore or cluster name specified while adding it on the ESA using the DFS Cluster Management Utility.



10. Select **OK**.
11. Press **Enter**.

    A list of all the ACL entries appears



.

4.15 HDFS Codec for Encryption and Decryption

A codec is an algorithm which provides compression and decompression. Hadoop provides a codec framework to compress blocks of data before storage. The codec compresses data while writing the blocks and decompresses data while reading the blocks.

A split-able codec is an algorithm which is applied after splitting a file, making it possible to recover original data from any part of the split.

The Protegrity HDFS codec is a split-able cryptographic codec. It uses encryption, such as AES 128, AES 256, DES and so on. It utilizes the infrastructure of the Protegrity Application Protector for applying cryptographic support. The protection is governed by the Policy deployed by the ESA, as defined by the Security Officer.

Chapter 5

HBase

HBase is a database, which provides random read and write access to tables, consisting of rows and columns, in real-time. HBase is designed to run on commodity servers, to automatically scale as more servers are added, and is fault tolerant as data is divided across servers in the cluster. HBase tables are partitioned into multiple regions. Each region stores a range of rows in the table. Regions contain a datastore in memory and a persistent datastore (HFile). The Name node assigns multiple regions to a region server. The Name node manages the cluster and the region servers store portions of the HBase tables and perform the work on the data.

5.1 Overview of the HBase Protector

The Protegrity HBase protector extends the functionality of the data storage framework and provides transparent data protection and unprotection using coprocessors, which provide the functionality to run code directly on region servers. The Protegrity coprocessor for HBase runs on the region servers and protects the data stored in the servers. All clients which work with HBase are supported.

The data is transparently protected or unprotected, as required, utilizing the coprocessor framework.

5.2 HBase Protector Usage

The Protegrity HBase protector utilizes the *get*, *put*, and *scan* commands and calls the Protegrity coprocessor for the HBase protector. The Protegrity coprocessor for the HBase protector locates the metadata associated with the requested column qualifier and the current logged in user. If the data element is associated with the column qualifier and the current logged in user, then the HBase protector processes the data in a row based on the data elements defined by the security policy deployed in the Big Data Protector.

Warning:
The Protegrity HBase coprocessor only supports *bytes* converted from the *string* data type.

If any other data type is directly converted to *bytes* and inserted in an HBase table, which is configured with the Protegrity HBase coprocessor, then data corruption might occur.

5.3 Adding Data Elements and Column Qualifier Mappings to a New Table

In an HBase table, every column family of a table stores metadata for that family, which contain the column qualifier and data element mappings.

Users need to add metadata to the column families for defining mappings between the data element and column qualifier, when a new HBase table is created.

The following command creates a new HBase table with one column family.

```
create 'table', { NAME => 'column_family_1', METADATA => {
'DATA_ELEMENT:credit_card'=>'CC_NUMBER','DATA_ELEMENT:name'=>'TOK_CUSTOMER_NAME' } }
```

Parameters

table: Name of the table.

column_family_1: Name of the column family.

METADATA: Data associated with the column family.

DATA_ELEMENT: Contains the column qualifier name. In the example, the column qualifier names *credit_card* and *name*, correspond to data elements CC_NUMBER and TOK_CUSTOMER_NAME respectively.

5.4 Adding Data Elements and Column Qualifier Mappings to an Existing Table

Users can add data elements and column qualifiers to an existing HBase table. Users need to alter the table to add metadata to the column families for defining mappings between the data element and column qualifier.

The following command adds data elements and column qualifier mappings to a column in an existing HBase table.

```
alter 'table', { NAME => 'column_family_1', METADATA =>
{ 'DATA_ELEMENT:credit_card'=>'CC_NUMBER', 'DATA_ELEMENT:name'=>'TOK_CUSTOMER_NAME' } }
```

Parameters

table: Name of the table.

column_family_1: Name of the column family.

METADATA: Data associated with the column family.

DATA_ELEMENT: Contains the column qualifier name. In the example, the column qualifier names *credit_card* and *name*, correspond to data elements CC_NUMBER and TOK_CUSTOMER_NAME respectively.

5.5 Inserting Protected Data into a Protected Table

Users can ingest protected data into a protected table in HBase using the BYPASS_COPROCESSOR flag. If the BYPASS_COPROCESSOR flag is set while inserting data in the HBase table, then the Protegrity coprocessor for HBase is bypassed.

The following command bypasses the Protegrity coprocessor for HBase and ingests protected data into an HBase table.

```
put 'table', 'row_2', 'column_family:credit_card', '3603144224586181', {
ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

Parameters

table: Name of the table.

column_family: Name of the column family.

METADATA: Data associated with the column family.

ATTRIBUTES: Additional parameters to consider when ingesting the protected data. In the example, the flag to bypass the Protegrity coprocessor for HBase is set.

5.6 Retrieving Protected Data from a Table

If users need to retrieve protected data from an HBase table, then they need to set the BYPASS_COPROCESSOR flag to retrieve the data. This is necessary to retain the protected data as is since HBase performs protects and unprotects the data transparently.

The following command bypasses the Protegrity coprocessor for HBase and retrieves protected data from an HBase table.

```
scan 'table', { ATTRIBUTES => {'BYPASS_COPROCESSOR'=>'1'}}
```

Parameters

table: Name of the table.

ATTRIBUTES: Additional parameters to consider when ingesting the protected data. In the example, the flag to bypass the Protegrity coprocessor for HBase is set.

5.7 HBase Commands

Hadoop provides shell commands to ingest, extract, and display the data in an HBase table.

For more information about the commands supported by HBase, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

5.8 Ingesting Files Securely

To ingest data into HBase securely, use the *put* command.

For more information, refer to section *3.6.1 put* in *Protegrity APIs, UDFs, and Commands Reference Guide Release 8.1.0.0*.

5.9 Extracting Files Securely

To extract data from HBase securely, use the *get* command.

For more information, refer to section *3.6.2 get* in *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

Chapter 6

Impala

Impala is an MPP SQL query engine for querying the data stored in a cluster. It provides the flexibility of the SQL format and is capable of running the queries on HDFS in HBase.

> **Note:**
> This section is applicable for the CDH native installer only.

This section provides information about the Impala protector, the UDFs provided, and the commands for protecting and unprotecting data in an Impala table.

6.1 Overview of the Impala Protector

Impala is an MPP SQL query engine for querying the data stored in a cluster. The Protegrity Impala protector extends the functionality of the Impala query engine and provides UDFs which protect or unprotect the data as it is stored or retrieved.

6.2 Impala Protector Usage

The Protegrity Impala protector provides UDFs for protecting data using encryption or tokenization, and unprotecting data by using decryption or detokenization.

> **Note:** Ensure that the */user/impala* path exists in HDFS with the Impala supergroup permissions.
> You can verify this by the following command:
>
> ```
> # hadoop fs –ls /user
> ```

6.2.1 Creating the */user/impala* path in Impala with Supergroup permissions

To create the */user/impala* path in Impala with Supergroup permissions:

If the */user/impala* path does not exist or does not have supergroup permissions, then perform the following steps.

1. Create the */user/impala* directory in HDFS using the following command.

```
# sudo –u hdfs hadoop –mkdir /user/impala
```

2. Assign Impala supergroup permissions to the */user/impala* path using the following command.

```
# sudo –u hdfs hadoop –chown –R impala:supergroup /user/impala
```

6.3 Impala UDFs

For more information about the Impala UDFs, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

6.4 Inserting Data from a File into a Table

To insert data from a file into an Impala table, ensure that the required user permissions for the directory path in HDFS are assigned for the Impala table.

6.4.1 Preparing the environment for the *basic_sample.csv* file

To prepare the environment for the *basic_sample.csv* file:

1. Assign permissions to the path where data from the *basic_sample.csv* file needs to be copied using the following command:

```
sudo -u hdfs hadoop fs -chown root:root /tmp/basic_sample/sample/
```

2. Copy the data from the *basic_sample.csv* file into HDFS using the following command:

```
hdfs dfs -put basic_sample.csv /tmp/basic_sample/sample/
```

3. Verify the presence of the *basic_sample.csv* file in the HDFS path using the following command:

```
hdfs dfs -ls /tmp/basic_sample/sample/
```

4. Assign permissions for Impala to the path where the *basic_sample.csv* file is located using the following command:

```
sudo -u hdfs hadoop fs -chown impala:supergroup /path/
```

6.4.2 Populating the table *sample_table* from the *basic_sample_data.csv* file

To populate the table *sample_table* from the *basic_sample_data.csv* file:

The following commands populate the table *basic_sample* with the data from the *basic_sample_data.csv* file.

```
create table sample_table(colname1 colname1_format, colname2 colname2_format, colname3
colname3_format)
row format delimited fields terminated by ',';
LOAD DATA INPATH '/tmp/basic_sample/sample/' INTO TABLE sample_table;
```

Parameters

sample_table: Name of the Impala table created to load the data from the input CSV file from the required path.

colname1, colname2, colname3: Name of the columns.

colname1_format, colname2_format, colname3_format: The data types contained in the respective columns. The data types can only be of types STRING, INT, DOUBLE, or FLOAT.

ATTRIBUTES: Additional parameters to consider when ingesting the data.

In the example, the row format is delimited using the ‘,’ character as the row format in the input file is comma separated. If the input file is tab separated, then the the row format is delimited using '\t'.

6.5 Protecting Existing Data

To protect existing data, users should define the mappings between the columns and their respective data elements in the data security policy.

The following commands ingest cleartext data from the *basic_sample* table to the *basic_sample_protected* table in protected form using Impala UDFs.

```
create table basic_sample_protected (colname1 colname1_format, colname2 colname2_format,
colname3 colname3_format)
insert into basic_sample_protected(colname1, colname2, colname3)
select ID,pty_stringins(colname1, dataElement1),pty_stringins(colname2,
dataElement2),pty_stringins(colname3, dataElement3)
from basic_sample;
```

Parameters

basic_sample_protected: Table to store protected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

basic_sample: Table containing the original data in cleartext form.

6.6 Unprotecting Protected Data

To unprotect protected data, you need to specify the name of the table which contains the protected data, the table which would store the unprotected data, and the columns and their respective data elements.

Ensure that the user performing the task has permissions to unprotect the data as required in the data security policy.

The following commands unprotect the protected data in a table and stores the data in cleartext form in to a different table, if the user has the required permissions.

```
create table table_unprotected (colname1 colname1_format, colname2 colname2_format, colname3
colname3_format)
insert into table_unprotected (colname1, colname2, colname3) select
ID,pty_stringsel(colname1, dataElement1),
pty_stringsel(colname2, dataElement2),pty_stringsel(colname3, dataElement3) from
table_protected;
```

Parameters

table_unprotected: Table to store unprotected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

table_protected: Table containing protected data.

6.7 Retrieving Data from a Table

To retrieve data from a table, the user needs to have access to the table.

The following command displays the data contained in the table.

```
select * from table;
```

Parameters

table: Name of the table.

Chapter 7

Spark

Spark is an execution engine that carries out batch processing of jobs in-memory and handles a wider range of computational workloads. In addition to processing a batch of stored data, Spark is capable of manipulating data in real time.

Spark leverages the physical memory of the Hadoop system and utilizes Resilient Distributed Datasets (RDDs) to store the data in-memory and lowers latency, if the data fits in the memory size. The data is saved on the hard drive only if required. As RDDs are the basic units of abstraction and computation in Spark, you can use the protection and unprotection APIs, provided by the Spark protector, when performing the transformation operations on an RDD.

If you need to use the Spark Protector API in a Spark Java job, then the users will have to implement the function interface as per the Spark Java programming specifications and subsequently use it in the required transformation of an RDD to tokenize the data.

This section provides information about the Spark protector, the APIs provided, and the commands for protecting and unprotecting data in a file by using the respective Spark APIs for protection or unprotection. In addition, it provides information about Spark SQL, which is a module that adds relational data processing capabilities to the Spark APIs, and a sample program for Spark Scala.

> **Note:**
> This section considers Spark, version 1.5.x, or higher as reference.

7.1 Overview of the Spark Protector

The Protegrity Spark protector extends the functionality of the Spark engine and provides APIs that protect or unprotect the data as it is stored or retrieved.

7.2 Spark Protector Usage

The Protegrity Spark protector provides APIs for protecting and reprotecting the data using encryption or tokenization, and unprotecting data by using decryption or detokenization.

> **Note:** Ensure that configure the Spark protector after installing the Big Data Protector.

7.3 Spark Java

This section describes the Spark APIs (Java) available for protection and unprotection in the Big Data Protector to build secure Big Data applications.

7.3.1 Spark Java APIs

For more information about the Spark APIs (Java, refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

Warning:

The Protegrity Spark protector only supports *bytes* converted from the *string* data type.

If any other data type is directly converted to *bytes* and passed as input to the API that supports *byte* as input and provides *byte* as output, then data corruption might occur.

7.3.2 Spark APIs and Supported Protection Methods

The following table lists the Spark APIs, the input and output data types, and the supported Protection Methods.

Note:

- Starting from the Version 7.1, Maintenance Release 1 (MR1), the DTP2 protection method is deprecated.
- For assistance in switching to a different protection method, contact Protegrity.

Table 7-1: Spark APIs and Supported Protection Methods

Operation	Input	Output	Protection Method Supported
Protect	Byte	Byte	Tokenization, Encryption, No Encyption, DTP2, CUSP
Protect	Short	Short	Tokenization, No Encyption
Protect	Short	Byte	Encryption, CUSP
Protect	Int	Int	Tokenization, No Encyption
Protect	Int	Byte	Encryption, CUSP
Protect	Long	Long	Tokenization, No Encyption
Protect	Long	Byte	Encryption, CUSP
Protect	Float	Float	Tokenization, No Encyption
Protect	Float	Byte	Encryption, CUSP
Protect	Double	Double	Tokenization, No Encyption
Protect	Double	Byte	Encryption, CUSP
Protect	String	String	Tokenization, No Encyption, DTP2
Protect	String	Byte	Encryption, CUSP
Unprotect	Byte	Byte	Tokenization, Encryption, No Encyption, DTP2, CUSP
Unprotect	Short	Short	Tokenization, NoEncyption
Unprotect	Byte	Short	Encryption, CUSP

Operation	Input	Output	Protection Method Supported
Unprotect	Int	Int	Tokenization, No Encyption
Unprotect	Byte	Int	Encryption, CUSP
Unprotect	Long	Long	Tokenization, No Encyption
Unprotect	Byte	Long	Encryption, CUSP
Unprotect	Float	Float	Tokenization, No Encyption
Unprotect	Byte	Float	Encryption, CUSP
Unprotect	Double	Double	Tokenization, No Encyption
Unprotect	Byte	Double	Encryption, CUSP
Unprotect	String	String	Tokenization, No Encyption, DTP2
Unprotect	Byte	String	Encryption, CUSP
Reprotect	Byte	Byte	Tokenization, Encryption, DTP2, CUSP **Note:** If a protected value is generated using *Byte* as both *Input* and *Output*, then only Encryption/CUSP is supported.
Reprotect	Short	Short	Tokenization
Reprotect	Int	Int	Tokenization
Reprotect	Long	Long	Tokenization
Reprotect	Float	Float	Tokenization
Reprotect	Double	Double	Tokenization
Reprotect	String	String	Tokenization, DTP2

7.3.3 Loading the Cleartext Data from a File to HDFS

You must first create a sample csv file that contains the cleartext data in comma separated value format. For example, create the *basic_sample_data.csv* file.

For more information on the sample data to be used for creating this csv file, refer to section *Sample Data*.

To load the cleartext data from the *basic_sample_data.csv* file:

To load the cleartext data from the *basic_sample_data.csv* file to HDFS, run the following command.

```
hadoop fs -put <Local_Filesystem_Path>/basic_sample_data.csv <
Path_of_Cleartext_data_file>
```

where,

Parameters	Description
basic_sample_data.csv	Specifies the name of the file containing cleartext data

Parameters	Description
<Local_Filesystem_Path>	Specifies the directory path on the local machine where the *basic_sample_data.csv* file is saved.
<Path_of_Cleartext_data_file>	Specifies the HDFS directory path for the file with the cleartext data. **Note:** Ensure that the user who is running the command has read and write access to this location.

7.3.4 Protecting the Existing Data

To protect cleartext data, you must specify the name of the file, which contains the cleartext data and the name of the location that contains the file which would store the protected data.

The following command reads the cleartext data from the *basic_sample_data.csv* file and stores it in the *basic_sample_protected* directory in protected form using the Spark APIs.

```
./spark-submit --master yarn --class com.protegrity.spark.ProtectData <PROTEGRITY_DIR>/
samples/spark/lib/spark_protector_demo.jar
<Path_of_Cleartext_data_file>/basic_sample_data.csv
<Path_of_Protected_data_file>/basic_sample_protected
```

Note: Ensure that the user performing the task has the permissions to protect the data, as required, in the data security policy.

Parameters	Description
com.protegrity.spark.ProtectData	Specifies the Spark protector class for protecting the data.
spark_protector_demo.jar	Specifies the sample *.jar* file utilizing the Spark protector API for protecting data in the *.csv* file. You must create this sample *.jar* file by compiling the scala class files listed in section 9.5.1 Sample Code Usage for Spark (Scala)
<Path_of_Cleartext_data_file>	Specifies the HDFS directory path for the file with cleartext data.
<Path_of_Protected_data_file>	Specifies the HDFS directory path for the file with protected data.
basic_sample_data	Specifies the name of the file to read cleartext data.

7.3.5 Unprotecting the Protected Data

To unprotect the protected data, you must specify the name of the location that contains the file, which stores the protected data and the name of the location that contains the file to store the unprotected data.

To retrieve the protected data from the *basic_sample_protected* directory and save it in the *basic_sample_unprotected* directory in unprotected form, use the following command.

```
./spark-submit --master yarn --class com.protegrity.spark.UnProtectData <PROTEGRITY_DIR>/
samples/spark/lib/spark_protector_demo.jar
<Path_of_Protected_data_file>/basic_sample_protected_data
<Path_of_Unprotected_data_file>/basic_sample_unprotected_data
```

Note: Ensure that the user performing the task has the permissions to unprotect the data, as required, in the data security policy.

where,

Parameter	Description
com.protegrity.spark.UnProtectData	Specifies the Spark protector class for unprotecting the protected data.
spark_protector_demo.jar	Specifies the sample *.jar* file utilizing the Spark protector API for unprotecting the protected data in the *.csv* file. You must create this sample *.jar* file by compiling the scala class files listed in section 9.5.1 Sample Code Usage for Spark (Scala).
<Path_of_Protected_data_file>/basic_sample_protected_data	Specifies the HDFS directory path for the file with protected data.
<Path_of_Unprotected_data_file>/basic_sample_unprotected_data	Specifies the HDFS directory path for the file to store the unprotected data.

7.3.6 Retrieving the Unprotected Data from a File

To retrieve data from a file containing protected data, the user needs to have access to the file.

To view the unprotected data contained in the file, use the following command.

```
hadoop fs -cat <Path_of_Unprotected_data_file> /basic_sample_unprotected_data/part*
```

where,

Parameter	Description
<Path_of_Unprotected_data_file>/basic_sample_unprotected_data	Specifies the HDFS directory path for the file that contains the unprotected data.

7.4 Spark SQL

The Spark SQL module provides relational data processing capabilities to Spark. The module allows you to run SQL queries with Spark programs. It contains DataFrames, which is an RDD with an associated schema, that provide support for processing structured data in Hive tables.

Spark SQL enables structured data processing and programming of RDDs providing relational and procedural processing through a DataFrame API that integrates with Spark.

7.4.1 DataFrames

A DataFrame is a distributed collection of data, such as RDDs, with a corresponding schema. DataFrames can be created from a wide array of sources, such as Hive tables, external databases, structured data files, or existing RDDs.

It can act as a distributed SQL query engine and is equivalent to a table in a relational database that can be manipulated, similar to RDDs. To optimize execution, DataFrames support relational operations and track their schema.

7.4.2 SQLContext

A SQLContext is a class that is used to initialize Spark SQL. It enables applications to run SQL queries, while running SQL functions, and provides the result as a DataFrame.

HiveContext extends the functionality of SQLContext and provides capabilities to use Hive UDFs, create Hive queries, and access and modify the data in Hive tables.

The Spark SQL CLI is used to run the Hive metastore service in local mode and execute queries. When we run Spark SQL (*spark-sql*), which is client for running queries in Spark, it creates a *SparkContext* defined as *sc* and *HiveContext* defined as *sqlContext*.

7.4.3 Spark SQL UDFs

For more information about the Spark SQL User Defined Functions (UDFs), refer to *Protegrity APIs, UDFs, and Commands Reference Guide Release 9.1.0.0*.

> **Note:**
> The example code snippets provided in this section utilize SQL queries to invoke the UDFs, after they are registered, using the *sql.Context.sql()* method.

7.4.4 Inserting Data from a File into a Table

The following commands create a class named *Person* with columns to store data.

```
scala> import sqlContext.implicits._

scala> case class Person(colname1: colname1_format, colname2: colname2_format, colname3:
colname3_format)
```

The following command reads the local sample file *basic_sample_data.csv*.

```
scala> val input = sc.textFile("file:///opt/protegrity/samples/data/basic_sample_data.csv")
```

The following command creates a DataFrame by mapping the RDD to the RDD [Person] object.

```
scala> val df = input.map(x => x.split(",")).map(p => Person(p(0).toInt, p(1), p(2),
p(3))).toDF()
```

The following command registers the temporary table *sample_table*.

```
scala> df.registerTempTable("sample_table")
```

The following commands save the table *sample_table* to a Parquet file.

```
scala> import org.apache.spark.sql.SaveMode
scala> df.write.mode(SaveMode.Ignore).save("sample_table.parquet")
```

Parameters

sample_table: Name of the table created to load the data from the input CSV file from the required path.

colname1, colname2, colname3: Name of the columns.

colname1_format, colname2_format, colname3_format: The data types contained in the respective columns.

7.4.5 Protecting Existing Data

This following command creates a Spark SQL table with the protected data.

```
"SELECT ID, " +
       "ptyProtectStr(colname1, 'dataElement1') as colname1," +
       "ptyProtectStr(colname1, 'dataElement2') as colname2," +
       "ptyProtectStr(colname3, 'dataElement3') as colname3," +
       "FROM basic_sample"
        ).registerTempTable("basic_sample_protected")
```

> **Note:**
> Ensure that the user performing the task has the permissions to protect the data, as required, in the data security policy.

Parameters

basic_sample_protected: Table to store protected data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

basic_sample: Table containing the original data in cleartext form.

basic_sample_protected: Table to store protected data.

7.4.6 Unprotecting and Viewing the Protected Data

To unprotect and view the protected data, you need to specify the name of the table which contains the protected data, and the columns and their respective data elements.

Ensure that the user performing the task has permissions to unprotect the data as required in the data security policy.

The following commands unprotect the protected data from the table *table_protected*.

```
scala> drop table if exists table_unprotected;
scala> create table table_unprotected (colname1 colname1_format, colname2 colname2_format,
colname3 colname3_format) distributed randomly;

scala> sqlContext.sql(
      "SELECT ID," +
        "ptyUnprotectStr(colname1, 'dataElement1') as colname1," +
        "ptyUnprotectStr(colname2, 'dataElement2') as colname2," +
        "ptyUnprotectStr(colname3, 'dataElement3') as colname3," +
        "FROM table_protected"
        ).show(false)
```

Parameters

ptyUnprotectStr: The Protegrity Spark SQL UDF to unprotect *String* data.

colname1, colname2, colname3: Name of the columns.

dataElement1, dataElement2, dataElement3: The data elements corresponding to the columns.

table_protected: Table containing protected data.

7.4.7 Retrieving Data from a Table

To retrieve data from a table, the user needs to have access to the table.

The following command displays the data contained in the table.

```
scala> sqlContext.sql("SELECT * table").show()
```

Parameters

table: Name of the table.

7.4.8 Calling Spark SQL UDFs from Domain Specific Language (DSL)

You can utilize the functions of the Domain-Specific Langugage (DSL) and call Spark SQL UDFs to protect or unprotect data from the Dataframe APIs. The following sample snippet describes how to call the Spark SQL UDFs from a DSL.

Calling Spark SQL UDFs from Domain Specific Language (DSL)

```
package com.protegrity.spark.dsl

import com.protegrity.spark.PtySparkProtectorException
import org.apache.spark.sql.{Column, DataFrame, UserDefinedFunction}

/**
  * DSL API for applying protection on DataFrames implicitly.
  *
  * e.g
  * import sqlContext.implicits._
  * import com.protegrity.spark.dsl.PtySparkDSL._
  * val df = sc.parallelize(List("hello", "world")).toDF()
  * df.protect("_1", "AlphaNum")
  *    .withColumnRenamed("_1", "protected")
  *    .show()
  */
object PtySparkDSL {

  implicit class PtySparkDSL(dataFrame: DataFrame) {

    import org.apache.spark.sql.functions._

    private def applyUDFOnColumns(colname: String,
                                  dataElement: String,
                                  func: UserDefinedFunction): Seq[Column] = {
      dataFrame.schema.map { field =>
        val name = field.name
        if (name.equals(colname)) {
          func(col(colname), lit(dataElement)).as(colname)
        } else {
          column(name)
        }
      }
    }

    private def applyUDFOnColumns(colname: String, oldDataElement: String, newDataElement:
String, func: UserDefinedFunction): Seq[Column] = {
      dataFrame.schema.map { field =>
        val name = field.name
        if (name.equals(colname)) {
          func(col(colname), lit(oldDataElement), lit(newDataElement)).as(colname)
        } else {
          column(name)
        }
      }
    }

    /**
```

```
      * Returns data type of input field from DataFrame
      * @param colname
      * @return data type of the column
      */
    private def getFieldType(colname: String): String = {
      try {
        dataFrame.schema(colname).dataType.typeName
      } catch {
        case e: IllegalArgumentException =>
          throw new PtySparkProtectorException(e.getMessage)
      }
    }

    def protect(colname: String, dataElement: String): DataFrame = {
      val dataType = getFieldType(colname)
      val function = dataType match {
        case "short" => udf(com.protegrity.spark.udf.ptyProtectShort _)
        case "integer" => udf(com.protegrity.spark.udf.ptyProtectInt _)
        case "long" => udf(com.protegrity.spark.udf.ptyProtectLong _)
        case "float" => udf(com.protegrity.spark.udf.ptyProtectFloat _)
        case "double" => udf(com.protegrity.spark.udf.ptyProtectDouble _)
        case "decimal(38,18)" =>
          udf(com.protegrity.spark.udf.ptyProtectDecimal _)
        case "string" => udf(com.protegrity.spark.udf.ptyProtectStr _)
        case "date" => udf(com.protegrity.spark.udf.ptyProtectDate _)
        case "timestamp" => udf(com.protegrity.spark.udf.ptyProtectDateTime _)
        case _ =>
          throw new PtySparkProtectorException(
            "Error!! DSL API invoked on unsupported column type - " + dataType)
      }
      val columns = applyUDFOnColumns(colname, dataElement, function)
      dataFrame.select(columns: _*)
    }

    def protectUnicode(colname: String, dataElement: String): DataFrame = {
      val function = udf(com.protegrity.spark.udf.ptyProtectUnicode _)
      val columns = applyUDFOnColumns(colname, dataElement, function)
      dataFrame.select(columns: _*)
    }

    def unprotect(colname: String, dataElement: String): DataFrame = {
      val dataType = getFieldType(colname)
      val function = dataType match {
        case "short" => udf(com.protegrity.spark.udf.ptyUnprotectShort _)
        case "integer" => udf(com.protegrity.spark.udf.ptyUnprotectInt _)
        case "long" => udf(com.protegrity.spark.udf.ptyUnprotectLong _)
        case "float" => udf(com.protegrity.spark.udf.ptyUnprotectFloat _)
        case "double" => udf(com.protegrity.spark.udf.ptyUnprotectDouble _)
        case "decimal(38,18)" =>
          udf(com.protegrity.spark.udf.ptyUnprotectDecimal _)
        case "string" => udf(com.protegrity.spark.udf.ptyUnprotectStr _)
        case "date" => udf(com.protegrity.spark.udf.ptyUnprotectDate _)
        case "timestamp" =>
          udf(com.protegrity.spark.udf.ptyUnprotectDateTime _)
        case _ =>
          throw new PtySparkProtectorException(
            "Error!! DSL API invoked on unsupported column type - " + dataType)
      }
      val columns = applyUDFOnColumns(colname, dataElement, function)
      dataFrame.select(columns: _*)
    }

    def unprotectUnicode(colname: String, dataElement: String): DataFrame = {
      val function = udf(com.protegrity.spark.udf.ptyUnprotectUnicode _)
      val columns = applyUDFOnColumns(colname, dataElement, function)
      dataFrame.select(columns: _*)
    }

    def reprotect(colname: String, oldDataElement: String, newDataElement: String): DataFrame
= {
      val dataType = getFieldType(colname)
      val function = dataType match {
        case "short" => udf(com.protegrity.spark.udf.ptyReprotectShort _)
```

```
        case "integer" => udf(com.protegrity.spark.udf.ptyReprotectInt _)
        case "long" => udf(com.protegrity.spark.udf.ptyReprotectLong _)
        case "float" => udf(com.protegrity.spark.udf.ptyReprotectFloat _)
        case "double" => udf(com.protegrity.spark.udf.ptyReprotectDouble _)
        case "decimal(38,18)" =>
          udf(com.protegrity.spark.udf.ptyReprotectDecimal _)
        case "string" => udf(com.protegrity.spark.udf.ptyReprotectStr _)
        case "date" =>
          udf(com.protegrity.spark.udf.ptyReprotectDate _)
        case "timestamp" =>
          udf(com.protegrity.spark.udf.ptyReprotectDateTime _)
        case _ =>
          throw new PtySparkProtectorException(
            "Error!! DSL API invoked on unsupported column type - " + dataType)
      }
      val columns = applyUDFOnColumns(colname, oldDataElement, newDataElement, function)
      dataFrame.select(columns: _*)
    }

def reprotectUnicode(colname: String, oldDataElement: String, newDataElement: String):
DataFrame = {
  val function = udf(com.protegrity.spark.udf.ptyReprotectUnicode _)
  val columns = applyUDFOnColumns(colname, oldDataElement, newDataElement, function)
  dataFrame.select(columns: _*)
  }
  }
}
```

7.5 Spark Scala

The Protegrity Spark protector (Java) can be used with Scala to protect and reprotect the data by using encryption or tokenization, and unprotect the data by using decryption or detokenization.

> **Note:** In this Big Data Protector release, a sample code snippet for Spark Scala is provided.

7.5.1 Sample Code Usage for Spark (Scala)

The Spark protector sample program, described in this section, is an example on how to use the Protegrity Spark protector APIs with Scala.

The sample program utilizes the following three Scala classes for protecting and unprotecting data:

- **ProtectData.scala** – This main class creates the Spark context object and calls the DataLoader class for reading cleartext data.
- **UnProtectData.scala** - This main class creates the Spark Context object and calls the DataLoader class for reading protected data.
- **DataLoader.scala** - This loader class fetches the input from the input path, calls the *ProtectFunction* to protect the data, and stores the protected data as output in the output path. In addition, it fetches the input from the protected path, calls the *UnProtectFunction* to unprotect the data, and stores the cleartext content as output.

The following functions perform protection for every new line in the input or unprotection for every new line in the output.

- **ProtectFunction** - This class calls the Spark protector for every new line specified in the input to protect data.
- **UnProtectFunction** - This class calls the Spark protector for every new line specified in the input to unprotect data.

7.5.1.1 Main Job Class for Protect Operation – ProtectData.scala

ProtectData.scala

```
package com.protegrity.samples.spark.scala
```

```
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object ProtectData {
  def main(args: Array[String]) {
    // create a SparkContext object, which tells Spark how to access a cluster.
    val sparkContext = new SparkContext(new SparkConf())
    // create the new object for class DataLoader
    val protector = new DataLoader(sparkContext)
    // Call writeProtectedData method which read clear data from input Path i.e (args[0]) and
write data in output path after protect operation
    protector.writeProtectedData(args(0), args(1), ",")
  }
}
```

7.5.1.2 Main Job Class for Unprotect Operation – UnProtectData.scala

UnProtectData.scala

```
package com.protegrity.samples.spark.scala

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext

object UnProtectData {
  def main(args: Array[String]) {
    val sparkContext = new SparkContext(new SparkConf())
    val protector = new DataLoader(sparkContext)
    protector.unprotectData(args(0), args(1), ",")
  }
}
```

7.5.1.3 Utility to call Protect or Unprotect Function – DataLoader.scala

DataLoader.scala

```
package com.protegrity.samples.spark.scala

import org.apache.log4j.Logger
import org.apache.spark.SparkContext

object DataLoader {
  private val logger = Logger.getLogger(classOf[DataLoader])
}
/**
 * A Data loader utility for reading & writing protected and un-protected data
 */
class DataLoader(private var sparkContext: SparkContext) {

  private var data_element_names: Array[String] = Array("TOK_NAME", "TOK_PHONE",
"TOK_CREDIT_CARD", "TOK_AMOUNT")

  private var appid: String = sparkContext.getConf.getAppId
  /**
   * Writes protected data to the output path delimited by the input delimiter
   *
   * @param inputPath - path of the input employee info file
   * @param outputPath - path where the output should be saved
   * @param delim - denotes the delimiter between the fields in the file
   */

  def writeProtectedData(inputPath: String, outputPath: String, delim: String) {
    // read lines from the input path & create RDD
    val rdd = sparkContext.textFile(inputPath)
    //import ProtectFunction
    import com.protegrity.samples.spark.scala.ProtectFunction._
    //call ProtectFunction on rdd
    rdd.ProtectFunction(delim, appid, data_element_names, outputPath)
  }

  /**
```

```
   * Reads protected data from the input path delimited by the input delimiter
   *
   * @param protectedInputPath - path of the protected employee data
   * @param unprotectedOutputPath - output path where unprotected data should be stored.
   * @param delim
   */

  def unprotectData(protectedInputPath: String, unprotectedOutputPath: String, delim: String)
{
    // read lines from the protectedInputPath & create RDD
    val protectedRdd = sparkContext.textFile(protectedInputPath)
    //import UnProtectFunction
    import com.protegrity.samples.spark.scala.UnProtectFunction._
    //call UnprotectFunction on rdd
    protectedRdd.UnprotectFunction(delim, appid, data_element_names, unprotectedOutputPath)
  }

}
```

7.5.1.4 ProtectFunction.scala

ProtectFunction.scala

```
package com.protegrity.samples.spark.scala

import java.util.ArrayList
import org.apache.spark.rdd.RDD
import com.protegrity.spark.Protector
import com.protegrity.spark.PtySparkProtector


object ProtectFunction {
  /*Defining this class as implicit,so that we can add new functionality to an RDD on the fly.
  implicits are lexically bounded i.e If we import this class, then only we can use it's
functions otherwise not*/
  implicit class Protect(rdd: RDD[String]) {
    def ProtectFunction(delim: String, appid: String, dataElement: Array[String],
protectoutputpath: String) =
      {
        val protectedRDD = rdd.map { line =>
          // splits the input seperated by delimiter in the line
          val splits = line.split(delim)
          // store first split in protectedString as we are not going to protect first split.
          var protectedString = splits(0)
          // Initialize input size
          val input = Array.ofDim[String](splits.length)
          // Initialize output size
          val output = Array.ofDim[String](splits.length)
          // Initialize errorList
          val errorList = new ArrayList[Integer]()
          // create the new object for class ptySparkProtector
          var protector: Protector = new PtySparkProtector(appid)
          // Iterate through the splits and call protect operation
          for (i <- 1 until splits.length) {
            input(i) = splits(i)
            // To protect data, call protect method with parameter dataElement, errorList,
input array and output array.output will be stored in output[]
            protector.protect(dataElement(i - 1), errorList, input, output)
            //Apppend output with protectedString
            protectedString += delim + output(i)
          }
          protectedString
        }

        // Save protectedRDD into output path
        protectedRDD.saveAsTextFile(protectoutputpath)

      }
  }
}
```

7.5.1.5 UnprotectFunction.scala

UnprotectFunction.scala

```
package com.protegrity.samples.spark.scala

import java.util.ArrayList
import org.apache.spark.rdd.RDD
import com.protegrity.spark.Protector
import com.protegrity.spark.PtySparkProtector


object UnProtectFunction {
  /*Defining this class as implicit,so that we can add new functionality to an RDD on the fly.
  implicits are lexically bounded i.e If we import this class, then only we can use it's
functions otherwise not*/
  implicit class Unprotect(protectedRDD: RDD[String]) {
    def UnprotectFunction(delim: String, appid: String, dataElement: Array[String],
unprotectoutputpath: String) =
      {
        val unprotectedRDD = protectedRDD.map { line =>
          // splits the input seperated by delimiter in the line
          val splits = line.split(delim)
          // store first split in unprotectedString
          var unprotectedString = splits(0)
          // Initialize input size
          val input = Array.ofDim[String](splits.length)
          // Initialize output size
          val output = Array.ofDim[String](splits.length)
          // Initialize errorList
          val errorList = new ArrayList[Integer]()
          // create the object for class ptySparkProtector
          var protector: Protector = new PtySparkProtector(appid)
          // Iterate through the splits and call unprotect operation
          for (i <- 1 until splits.length) {
            input(i) = splits(i)
            // To unprotect data, call unprotect method with parameter dataElement,
errorList, input array and output array.output will be stored in output[]
            protector.unprotect(dataElement(i - 1), errorList, input, output)
            //Apppend output with protectedString
            unprotectedString += delim + output(i)
          }
          unprotectedString
        }

        // Save unprotectedRDD into output path
        unprotectedRDD.saveAsTextFile(unprotectoutputpath)
      }
  }
}
```

Chapter 8

Appendix: Return Codes

If you are using the HDFSFP protector and any failures occur, then the protector throws an exception. The exception consists of an error code and error message. The following table lists all possible error codes and error descriptions.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

Table 8-1: Error Codes for HDFSFP Protector

Code	Error	Error scription
0	XC_FAILED	The requested operation or service failed.
3	XC_LOG_CHECK_ACCESS	The access was denied as the user does not have the required privileges to perform the requested operation.
4	XC_LOG_TIME_ACCESS	The access was denied as the user does not have the required privileges to perform the requested operation at this point in time.
6	XC_LOG_ENCRYPT_SUCCESS	The data was successfully encrypted.
7	XC_LOG_ENCRYPT_FAILED	The encryption of data failed.
8	XC_LOG_DECRYPT_SUCCESS	The data was successfully decrypted.
9	XC_LOG_DECRYPT_FAILED	The decryption of data failed.
100	XC_INVALID_PARAMETER	The parameter specified in the function call is invalid.
101	XC_TIMEOUT	The operation timed out before a result was returned.
102	XC_ACCESS_DENIED	Permission to access an object or file on the filesystem is denied.
103	XC_NOT_SUPPORTED	The requested operation is not supported.
104	XC_SESSION_REFUSED	The remote peer client did not accept the session request.
105	XC_DISCONNECTED	The session was terminated.
106	XC_UNREACHABLE	The host could not be reached.
107	XC_SESSION_IN_USE	The session is already in use.
108	XC_EOF	The end of file is reached.
109	XC_NOT_FOUND	Not found.
110	XC_BUFFER_TOO_SMALL	Supplied input or output buffer is too small.

If you are using MapReduce, Hive, Pig, HBase, or Spark, and any failures occur, then the protector throws an exception. The exception consists of an error code and error message. The following table lists all possible error codes and error descriptions.

The following table lists all possible return codes provided to the PEP log files.

Table 8-2: PEP Log Return Codes

Code	Error	Error Description
0	NONE	
1	USER_NOT_FOUND	The user name could not be found in the policy residing in the shared memory.
2	DATA_ELEMENT_NOT_FOUND	The data element could not be found in the policy residing in the shared memory.
3	PERMISSION_DENIED	The user does not have the required permissions to perform the requested operation.
4	TIME_PERMISSION_DENIED	The user does not have the appropriate permissions to perform the requested operation at this point in time.
5	INTEGRITY_CHECK_FAILED	Integrity check failed.
6	PROTECT_SUCCESS	The operation to protect the data was successful.
7	PROTECT_FAILED	The operation to protect the data failed.
8	UNPROTECT_SUCCESS	The operation to unprotect the data was successful.
9	UNPROTECT_FAILED	The operation to unprotect the data failed.
10	OK_ACCESS	The user has the required permissions to perform the requested operation. This return code ensures a verification and no data is protected or unprotected.
11	INACTIVE_KEYID_USED	The operation to unprotect the data was successful using an inactive Key ID.
12	INVALID_PARAM	The input is null or not within allowed limits.
13	INTERNAL_ERROR	An internal error occurred in a function call after the PEP provider is started.
14	LOAD_KEY_FAILED	Failed to load the data encryption key.
17	INIT_FAILED	The PEP server failed to initialize, which is a fatal error.
20	OUT_OF_MEMORY	Failed to allocate memory.
21	BUFFER_TOO_SMALL	The input or output buffer is very small.
22	INPUT_TOO_SHORT	The data is too short to be protected or unprotected.
23	INPUT_TOO_LONG	The data is too long to be protected or unprotected.
25	USERNAME_TOO_LONG	The user name is longer than the maximum supported length of the user name that can be used for protect or unprotect operations.
26	UNSUPPORTED	The algorithm or action for the specific data element is unsupported.
27	APPLICATION_AUTHORIZED	The application is authorized.
28	APPLICATION_NOT_AUTHORIZED	The application is not authorized.
29	JSON_NOTSERIALIZABLE	The JSON type is not serializable.
30	JSON_MALLOC_FAILED	The memory allocation for the JSON type failed.
31	EMPTY_POLICY	The policy residing in the shared memory is empty.

Code	Error	Error Description
32	DELETE_SUCCESS	The operation to delete the data was successful.
33	DELETE_FAILED	The operation to delete the data failed.
34	CREATE_SUCCESS	The operation to create or add the data was successful.
35	CREATE_FAILED	The operation to create or add the data failed.
36	MNGPROT_SUCCESS	The management of the protection operation was successful.
37	MNGPROT_FAILED	The management of the protection operation failed.
39	POLICY_LOCKED	The policy residing in the shared memory is locked. This error can be caused by a *Disk Full* alert.
40	LICENSE_EXPIRED	The license is not valid or the current date is beyond the license expiration date.
41	METHOD_RESTRICTED	The use of the Protection method is restricted by license.
42	LICENSE_INVALID	The license is invalid or the time is prior to the start of the license tenure.
44	INVALID_FORMAT	The content of the input data is invalid.
45	PROCESSING_SUCCESS	It is used for audit entries used for collecting Access Counter records.
46	INVALID_POLICY	It is used for a z/OS Query regarding the default data element when the policy name is not found.
50	REPROTECT_SUCCESS	The data reprotection was successful.

The following table lists all possible result codes provided as a result of operations performed on the PEP.

Table 8-3: PEP Result Codes

Code	Error	Error Description
1	SUCCESS	The operation was successful.
0	FAILED	The operation failed.
-1	INVALID_PARAMETER	The parameter is invalid.
-2	EOF	The end of file was reached.
-3	BUSY	The operation is already in progress or the PEP server is busy with some other operation.
-4	TIMEOUT	The time-out threshold was reached as the PEP server was waiting for a response.
-5	ALREADY_EXISTS	The object, such as file, already exists.
-6	ACCESS_DENIED	The permission to access the object was denied.
-7	PARSE_ERROR	The error occurred when the contents were parsed.
-8	NOT_FOUND	The search operation was not successful.
-9	NOT_SUPPORTED	The operation is not supported.
-10	CONNECTION_REFUSED	The connection was refused.
-11	DISCONNECTED	The connection was terminated.
-12	UNREACHABLE	The Internet link is down or the host is not reachable.
-13	ADDRESS_IN_USE	The IP Adddress or port is already utilized.
-14	OUT_OF_MEMORY	The operation to allocate memory failed.

Code	Error	Error Description
-15	CRC_ERROR	The CRC check failed.
-16	BUFFER_TOO_SMALL	The buffer size is very small.
-17	BAD_REQUEST	The message received was not in a standard format.
-18	INVALID_STRING_LENGTH	The input string is very long.
-19	INVALID_TYPE	The incorrect type of <NEED INPUTS> was used.
-20	READONLY_OBJECT	The object is set with read-only access.
-21	SERVICE_FAILED	The service failed.
-22	ALREADY_CONNECTED	The Administrator is already connected to the server.
-23	INVALID_KEY	The key is invalid.
-24	INTEGRITY_ERROR	The integrity check failed.
-25	LOGIN_FAILED	The attempt to login failed.
-26	NOT_AVAILABLE	The object is not available.
-27	NOT_EXIST	The object does not exist.
-28	SET_FAILED	The Set operation failed.
-29	GET_FAILED	The Get operation failed.
-30	READ_FAILED	The Read operation failed.
-31	WRITE_FAILED	The Write operation failed.
-33	REWRITE_FAILED	The Rewrite operation failed.
-34	DELETE_FAILED	The Delete operation failed.
-35	UPDATE_FAILED	The Update operation failed.
-36	SIGN_FAILED	The Sign operation failed.
-37	VERIFY_FAILED	The Verification failed.
-38	ENCRYPT_FAILED	The Encrypt operation failed.
-39	DECRYPT_FAILED	The Decrypt operation failed.
-40	REENCRYPT_FAILED	The Reencrypt operation failed.
-41	EXPIRED	The object has expired.
-42	REVOKED	The object has been revoked.
-43	INVALID_FORMAT	The format is invalid.
-44	HASH_FAILED	The Hash operation failed.
-45	NOT_DEFINED	The property or setting is not defined.
-46	NOT_INITIALIZED	The service requested or function is performed on an object that is not initialized.
-47	POLICY_LOCKED	The Policy is locked.
-48	THROW_EXCEPTION	The error message is used to convey that an exception should be thrown during decryption.
-49	USER_AUTHENTICATION_FAILED	The Authentication operation failed.
-54	INVALID_CARD_TYPE	The credit card number provided does not confirm to the required credit card format.
-55	LICENSE_AUDITONLY	The License provided is for the audit functionality and only *No Encryption* data elements are allowed.
-56	NO_VALID_CIPHERS	No valid ciphers were found.
-57	NO_VALID_PROTOCOLS	No valid protocols were found.

Code	Error	Error Description
-201	CRYPT_KEY_DATA_ILLEGAL	The key data specified is invalid.
-202	CRYPT_INTEGRITY_ERROR	The integrity check for the data failed.
-203	CRYPT_DATA_LEN_ILLEGAL	The data length specified is invalid.
-204	CRYPT_LOGIN_FAILURE	The Crypto login failed.
-205	CRYPT_CONTEXT_IN_USE	An attempt to close a key being used is made.
-206	CRYPT_NO_TOKEN	The hardware token is available.
-207	CRYPT_OBJECT_EXISTS	The object to be created already exists.
-208	CRYPT_OBJECT_MISSING	A request for a non-existing object is made.
-221	X509_SET_DATA	The operation to set data in the object failed.
-222	X509_GET_DATA	The operation to get data from the object failed.
-223	X509_SIGN_OBJECT	The operation to sign the object failed.
-224	X509_VERIFY_OBJECT	The verification operation for the object failed.
-231	SSL_CERT_EXPIRED	The certificate has expired.
-232	SSL_CERT_REVOKED	The certificate has been revoked.
-233	SSL_CERT_UNKNOWN	The Trusted certificate was not found.
-234	SSL_CERT_VERIFY_FAILED	The certificate cound not be verified.
-235	SSL_FAILED	A general SSL error occurs.
-241	KEY_ID_FORMAT_ERROR	The format on the Key ID is invalid.
-242	KEY_CLASS_FORMAT_ERROR	The format on the KeyClass is invalid.
-243	KEY_EXPIRED	The key expired.
-250	FIPS_MODE_FAILED	The FIPS mode failed.

Chapter 9

Appendix: Using Hive with HDFSFP (Deprecated from Big Data Protector 7.2.0)

This section describes how to use Hive with HDFSFP.

Note:
Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

Warning:
The Hive native *Load* command for loading data from any file in the local file system or HDFS to a protected Hive table is not supported.

Note:
This release supports *TEXTFILE*, *RCFile*, and *SEQUENCEFILE* formats only.

9.1 Data Used by the Samples

The *employee.csv* file used in the examples demonstrating usage of Hive with HDFSFP contains the following sample data.

```
928724,Hultgren Caylor
928725,Bourne Jose
928726,Sorce Hatti
928727,Lorie Garvey
928728,Belva Beeson
928729,Hultgren Caylor
928730,Bourne Jose
928731,Lorie Garvey
928732,Bourne Jose
928733,Hultgren Caylor
928734,Lorie Garvey
```

9.2 Ingesting Data to Hive Table

If you need to ingest data from a relational database to a Hive protected table in HDFS, then ensure that you load the data through Sqoop and use the *-D target.output.dir* parameter, as described in the following command.

```
sqoop import -D target.output.dir="/tmp/src" --fields-terminated-by ',' --driver
com.mysql.jdbc.Driver --connect jdbc:mysql://master.localdomain:3306/db --username
user1 --password protegrity --table emp --hive-import --hive-table=emp_sqp -m 1;
```

9.2.1 Ingesting Data from HDFSFP Protected External Hive Table to HDFSFP Protected Internal Hive Table

To ingest data from an HDFSFP protected external Hive table to an HDFSFP protected internal Hive table:

1. Create a Protect ACL for the following path.

   ```
   /user/ptyitusr/ext1
   ```

2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:

   ```
   hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
   hadoop fs -ls /user/ptyitusr/ext1
   hadoop fs -cat /user/ptyitusr/ext1/employee.csv
   ```

3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the following commands.

   ```
   CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
   TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
   ext1';
   select * from ext1;
   ```

4. Create a Protect ACL for the following path.

   ```
   <hive.metastore.warehouse.dir>/int1
   ```

5. Create an internal Hive table (*int1*) using the following command.

   ```
   CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY
   ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
   ```

6. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in protected form using the following commands.

   ```
   SET hive.exec.compress.output=true;
   insert overwrite table int1 select empid, name from ext1;
   ```

   The data is stored in the internal Hive table *int1* in protected form.

7. Revert the value of the *hive.exec.compress.output* parameter using the following command.

   ```
   SET hive.exec.compress.output=false;
   ```

8. To view the data in the table *int1*, execute the following command.

   ```
   select * from int1;
   ```

9.2.2 Ingesting Protected Data from HDFSFP Protected Hive Table to another HDFSFP Protected Hive Table

To ingest protected data from an HDFSFP protected external Hive table to another HDFSFP protected Hive table:

1. Create Protect ACLs for the following paths.

```
/user/ptyitusr/ext1
/user/ptyitusr/ext2
```

2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:

```
hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
hadoop fs -ls /user/ptyitusr/ext1
hadoop fs -cat /user/ptyitusr/ext1/employee.csv
```

3. Create an external table (*ext1*) with the data contained in the protected staging directory using the following commands:

```
CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext1';
select * from ext1;
```

4. Create an external Hive table (*ext2*) using the following command:

```
CREATE EXTERNAL TABLE ext2 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext2';
```

5. Insert the cleartext data from the external Hive table *ext1* to the Hive table *ext2* in protected form using the following commands:

```
SET hive.exec.compress.output=true;
insert overwrite table ext2 select empid, name from ext1;
```

The data is stored in the Hive table *ext2* in protected form.

6. Revert the value of the *hive.exec.compress.output* parameter using the following command.

```
SET hive.exec.compress.output=false;
```

7. To view the data in the table *ext2*, execute the following command.

```
select * from ext2;
```

9.3 Tokenization and Detokenization with HDFSFP

9.3.1 Verifying Prerequisites for Using Hadoop Application Protector

Ensure that the following data elements are present in data security policy and the policy is deployed on all the Big Data Protector nodes:

- ***TOK_NUM_1_3_LP***– Length preserving Numeric tokenization data element with SLT_1_3
- ***TOK_ALPHA_2_3_LP***– Length preserving Alpha tokenization data element with SLT_2_3

9.3.2 Ingesting Data from HDFSFP Protected External Hive Table to HDFSFP Protected Internal Hive Table in Tokenized Form

To ingest data from an HDFSFP protected external Hive table to an HDFSFP protected internal Hive table in tokenized form:

1. Create a Protect ACLs for the following paths.

```
/user/ptyitusr/ext1
<hive.metastore.warehouse.dir>/int1
```

2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:

```
hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
hadoop fs -ls /user/ptyitusr/ext1
hadoop fs -cat /user/ptyitusr/ext1/employee.csv
```

3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the following commands.

```
CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext1';
select * from ext1;
```

4. Create an internal Hive table (*int1*) using the following command.

```
CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY
',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
```

5. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in tokenized form using the following commands.

```
SET hive.exec.compress.output=true;
create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';
insert overwrite table int1 select ptyProtectStr(empid, 'TOK_NUM_1_3_LP'),
ptyProtectStr(name, 'TOK_ALPHA_2_3_LP') from ext1;
```

   The data is stored in the internal Hive table *int1* in tokenized form.

6. Revert the value of the *hive.exec.compress.output* parameter using the following command.

```
SET hive.exec.compress.output=false;
```

7. To view the data in the table *int1*, execute the following command.

```
select * from int1;
```

9.3.3 Ingesting Detokenized Data from HDFSFP Protected Internal Hive Table to HDFSFP Protected External Hive Table

To ingest detokenized data from an HDFSFP protected internal Hive table with tokenized data to an HDFSFP protected external Hive table:

1. Create Protect ACLs for the following paths.

```
/user/ptyitusr/ext1
<hive.metastore.warehouse.dir>/int1
```

2. Create an external Hive table *(ext1)* with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.

```
CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext1';
```

3. Insert the tokenized data from the internal Hive table (*int1*) to the external Hive table (*ext1*) in detokenized form using the following commands.

```
SET hive.exec.compress.output=true;
create temporary function ptyUnprotectStr AS 'com.protegrity.hive.udf.ptyUnprotectStr';
insert overwrite table ext2 select ptyUnprotectStr(empid, 'TOK_NUM_1_3_LP'),
ptyUnprotectStr(name, 'TOK_ALPHA_2_3_LP') from int1;
```

   The tokenized data is stored in the external Hive table *ext1* in detokenized form.

4. Revert the value of the *hive.exec.compress.output* parameter using the following command.

```
SET hive.exec.compress.output=false;
```

5. To view the data in the table *ext1*, execute the following command.

```
select * from ext1;
```

9.3.4 Ingesting Data from HDFSFP Protected External Hive Table to Internal Hive Table not protected by HDFSFP in Tokenized Form

➤ To ingest data from an HDFSFP protected external Hive table to an internal Hive table not protected by HDFSFP in tokenized form:

1. Create a Protect ACL for the following path.

```
/user/ptyitusr/ext1
```

2. Copy the cleartext file (*employee.csv*) into the protected staging directory (*/user/ptyitusr/ext1*) using the following commands:

```
hadoop ptyfs -put employee.csv /user/ptyitusr/ext1
hadoop fs -ls /user/ptyitusr/ext1
hadoop fs -cat /user/ptyitusr/ext1/employee.csv
```

3. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the following commands.

```
CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext1';
select * from ext1;
```

4. Create an internal Hive table (*int1*) using the following command.

```
CREATE TABLE int1 (empid string, name string) ROW FORMAT DELIMITED FIELDS TERMINATED BY
',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/ext1';
```

5. Insert the cleartext data from the external Hive table (*ext1*) to the internal Hive table (*int1*) in tokenized form using the following commands.

```
SET hive.exec.compress.output=false; // Set this property to false to prevent insertion
of encrypted data into the unprotected table
create temporary function ptyProtectStr AS 'com.protegrity.hive.udf.ptyProtectStr';
insert overwrite table int1 select ptyProtectStr(empid, 'TOK_NUM_1_3_LP'),
ptyProtectStr(name, 'TOK_ALPHA_2_3_LP') from ext1;
```

   The data is stored in the internal Hive table *int1* in tokenized form.

6. To view the tokenized data in the table *int1*, execute the following command.

```
select * from int1;
```

9.3.5 Ingesting Detokenized Data from Internal Hive Table not protected by HDFSFP to HDFSFP Protected External Hive Table

➤ To ingest detokenized data from an internal Hive table with tokenized data not protected by HDFSFP to an HDFSFP protected external Hive table:

1. Create a Protect ACL for the following path.

```
/user/ptyitusr/ext1
```

2. Create an external Hive table (*ext1*) with the data contained in the protected staging directory (*/user/ptyitusr/ext1*) using the followind commands.

```
CREATE EXTERNAL TABLE ext1 (empid string, name string) ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' LINES TERMINATED BY '\n' STORED AS TEXTFILE LOCATION '/user/ptyitusr/
ext1';
```

3. Insert the tokenized data from the internal Hive table (*int1*) to the external Hive table (*ext1*) in detokenized form using the following commands.

```
SET hive.exec.compress.output=true;
create temporary function ptyUnprotectStr AS 'com.protegrity.hive.udf.ptyUnprotectStr';
insert overwrite table ext1 select ptyUnprotectStr(empid, 'TOK_NUM_1_3_LP'),
ptyUnprotectStr(name, 'TOK_ALPHA_2_3_LP') from int1;
```

   The tokenized data is stored in the external Hive table *ext1* in detokenized form.

4. Revert the value of the *hive.exec.compress.output* parameter using the following command.

```
SET hive.exec.compress.output=false;
```

5. To view the detokenized data in the table *ext1*, execute the following command.

```
select * from ext1;
```

Chapter 10

Appendix: Configuring Talend with HDFSFP (Deprecated from Big Data Protector 7.2.0)

This section describes the procedures for configuring Talend with HDFSFP.

> **Note:**
> Starting from the Big Data Protector 7.2.0 release, the HDFS File Protector (HDFSFP) is deprecated. The HDFSFP-related sections are retained to ensure coverage for using an older version of Big Data Protector with the ESA 9.0.0.0.

> **Note:** This section considers Talend, version 5.6 for reference.

10.1 Verifying Prerequisites before Configuring Talend with HDFSFP

Ensure that the following prerequisites are met before configuring Talend with HDFSFP:

- Install the Big Data Protector with the **HDFSFP** option in the *BDP.config* file as *Yes*.
- Deploy the unstructured policy on the cluster.

10.2 Verifying the Talend Packages

Verify if the *<PROTEGRITY_DIR>/etl/talend* directory is populated with the following directories:

- *tptyHDFSInput* – This directory contains the custom Protegrity HDFS input component.
- *tptyHDFSOutput* – This directory contains the custom Protegrity HDFS output component.
- *jars* – This directory contains the third-party JARs.
- *docs* – This directory contains the user guide for Talend.

10.3 Configuring Talend with HDFSFP

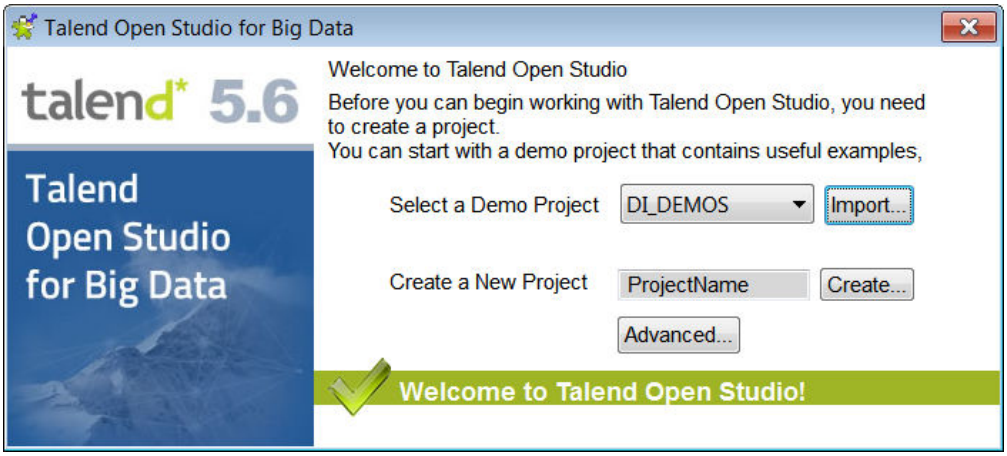
To configure Talend with HDFSFP:

1. Login to the ESA.
2. Create the datastore.
3. Create the ACL entry for the datastore.
4. Activate the ACL entry for the datastore.
5. Create a directory for Talend using the following command.
   ```
   mkdir <PROTEGRITY_DIR>/talend/
   ```
6. Create the *hdfsfp* directory for Talend using the following command.
   ```
   mkdir <TALEND_Installation_DIR>/hdfsfp/
   ```
7. Copy the *tptyHDFSInput* and *tptyHDFSOutput* directories from either the *etl/talend-ge-6.1.0* or the *etl/talend-le-6.0.1* directory, depending on the Talend version used, to the *<TALEND_Installation_DIR>/talend/* directory.
8. If you are using the CDH native installer, then perform the following steps.
   a. Copy the following files files from the */opt/cloudera/parcels/PTY_HDFSFP/hdfsfp/* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
      - *hdfsfp.jar*
      - *jedis-2.1.0.jar*
      - *beuler.properties*
      - *hdfsfp-log4j.properties*
   b. Copy the following files from the */opt/cloudera/parcels/PTY_HDFSFP/defiance_xc/java/lib/* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
      - *xcpep2jni.jar*
      - *xcpep2jni.plm*
      - *xcpep2jni.properties*
9. If you are using the Ambari native installer, then perform the following steps.
   a. Copy the following files files from the */var/lib/ambari-agent/cache/common-services/BDPHDFSFP/<build_version>/package/setup/hdfsfp/* directory to the *<TALEND_Installation_DIR>/hdfsfp* directory.
      - *hdfsfp.jar*
      - *jedis-2.1.0.jar*
      - *beuler.properties*
      - *hdfsfp-log4j.properties*
   b. Copy the following files from the */var/lib/ambari-agent/cache/common-services/BDPHDFSFP/<build_version>/package/setup/defiance_xc/java/lib/* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
      - *xcpep2jni.jar*
      - *xcpep2jni.plm*
      - *xcpep2jni.properties*
10. Copy the following third-party JAR files from the *<PROTEGRITY_DIR>/etl/talend/jars* directory to the *<TALEND_Installation_DIR>/hdfsfp/* directory.
    - *commons-codec-1.4.jar*
    - *commons-collections-3.2.1.jar*

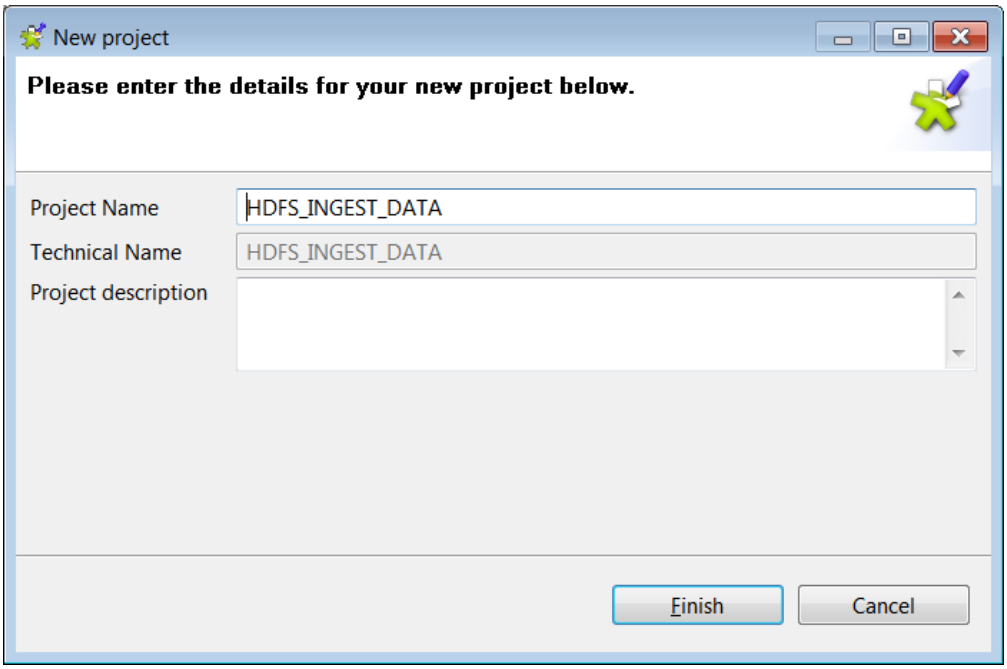
10.4 Starting a Project in Talend

To start a project in Talend:

1. Login to the Edgenode machine with Talend installed.
2. Execute the *TOS_BD-linux-gtk-x86.sh* script to start **Talend Open Studio for Big Data**.
   The **Talend Open Studio for Big Data** window appears.

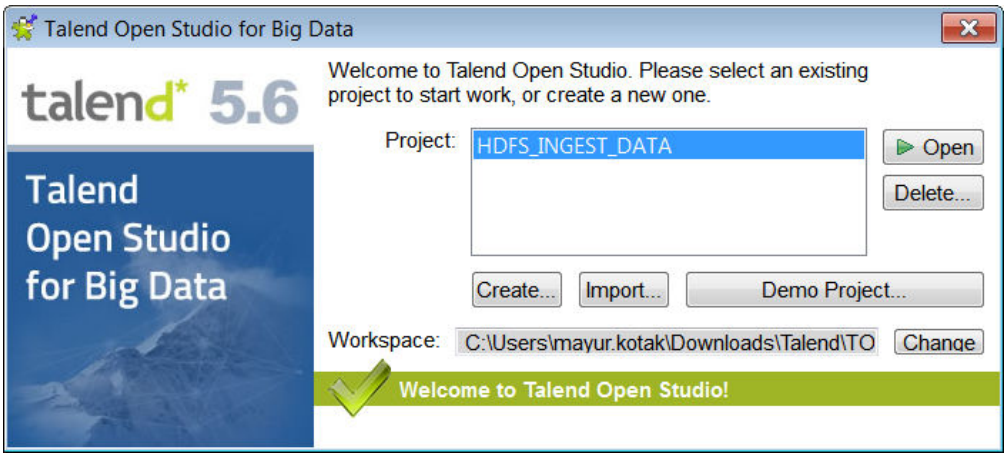


3. In the Create a New Project box, type the following:
   HDFS_INGEST_DATA
4. Click **Create** to create a new project.
   The **New Project** window appears.



5. Click **Finish** to create the new project.

The **Talend Open Studio for Big Data** window appears listing the new project.



6. Click **Open**.

   The **Talend Open Studio for Big Data** workspace appears.

10.5 Configuring the Preferences for Talend

➤ To configure the preferences for Talend:

1. On the **Talend Open Studio for Big Data** workspace, click **Window**.

The **Window** menu appears.

2. Click **Preferences**.

   The **Preferences** window appears.

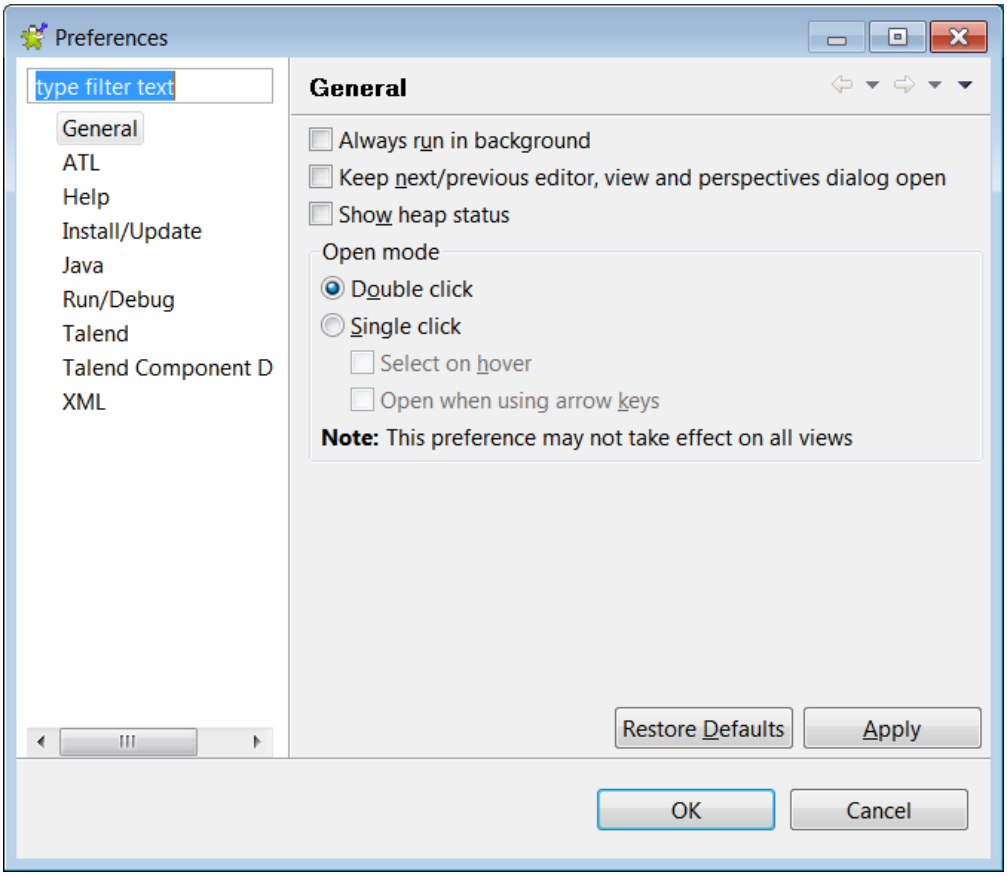


Figure 10-1: Preferences window

3. Click **Talend** in the Preferences pane.

   The general preferences for Talend appears.

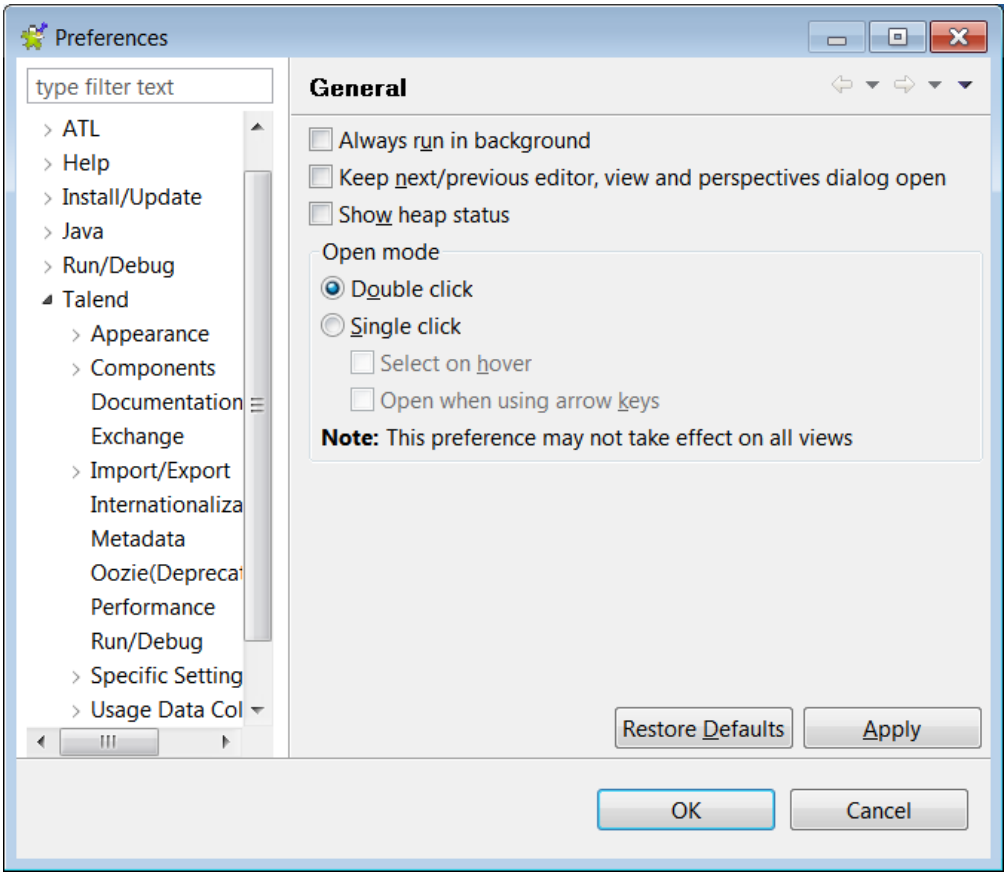


Figure 10-2: Talend preferences

4. Click **Components** in the Preferences pane.

   The components preferences for Talend appears.

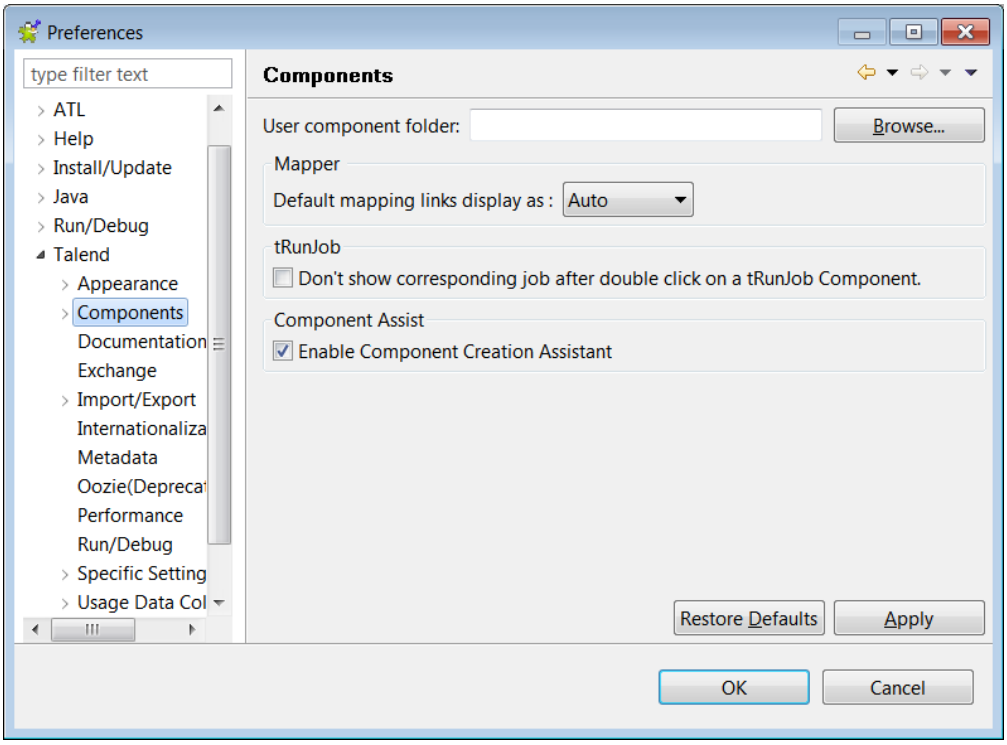


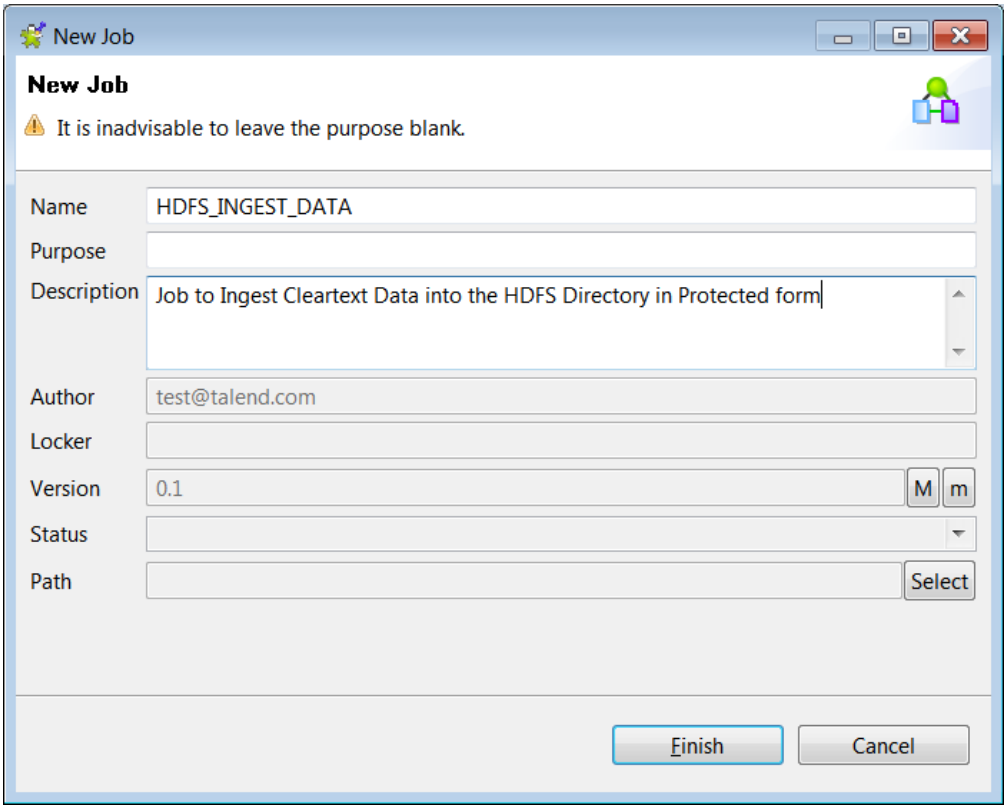
Figure 10-3: Components preferences

5. In the User component directory box, type the following path:
   <PROTEGRITY_DIR>/talend/
6. Click **Apply**.
7. Click **OK**.
   The preferences for Talend are updated.

10.6 Ingesting Data in the Target HDFS Directory in Protected Form

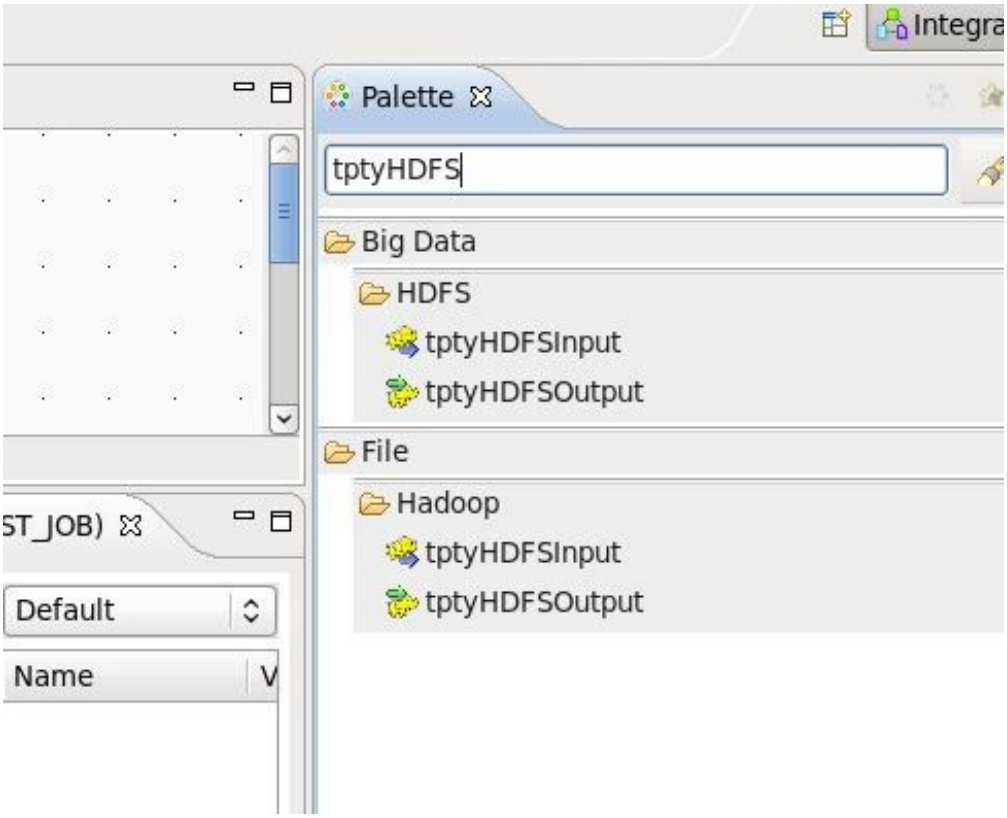
➤ To ingest Cleartext Data into the Target HDFS Directory:

1. Click the **Job** link under the **Create a new** section.
   The **New Job** window appears.



2. Type the following in the **Name** box.
   HDFS_INGEST_DATA
3. Enter a description in the **Description** box.
4. Click **Finish**.
   The new job in Talend is created.

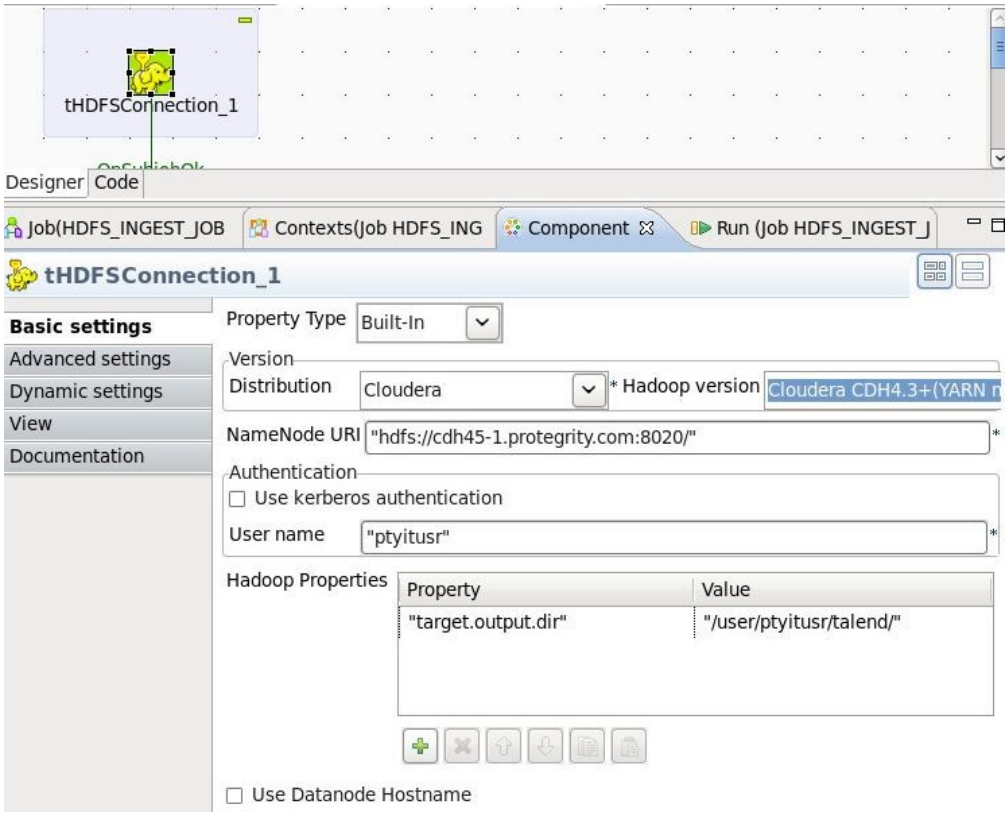
5. Verify if the custom components, *tptyHDFSInput* and *tptyHDFSOutput*, are loaded into the palette by searching the name *tptyHDFS*.
   The two components, *tptyHDFSInput* and *tptyHDFSOutput* appear.



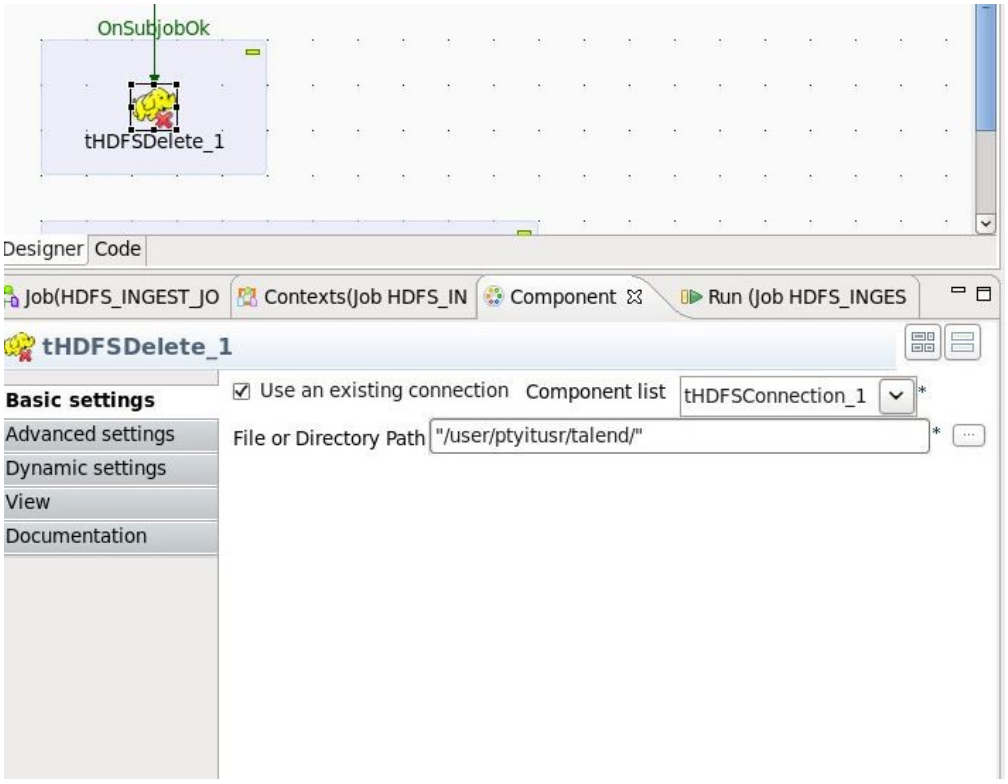
6. Double-click the *tHDFSConnection* component to create the HDFS connection.

   Enter the following properties for the connection, as required:

   - Distribution – The distribution name
   - Hadoop version – The version required for the Hadoop cluster
   - NameNode URI – The domain name and port of the Name node to connect to HDFS
   - User name – The user name to perform the HDFSFP operations. For instance, we have the user as *ptyitusr*.
   - target.output.dir – The targeted HDFS directory to store the protected data. For instance, we have the HDFS directory as */user/ptyitusr/talend*.



7. Create the *tHDFSDelete* component.
   a. If the directory exists, then delete the contents of the directory.
   b. Select the *Use an existing connection box* to reuse the existing connection.



8. Create the *tFileInputDelimited* component.

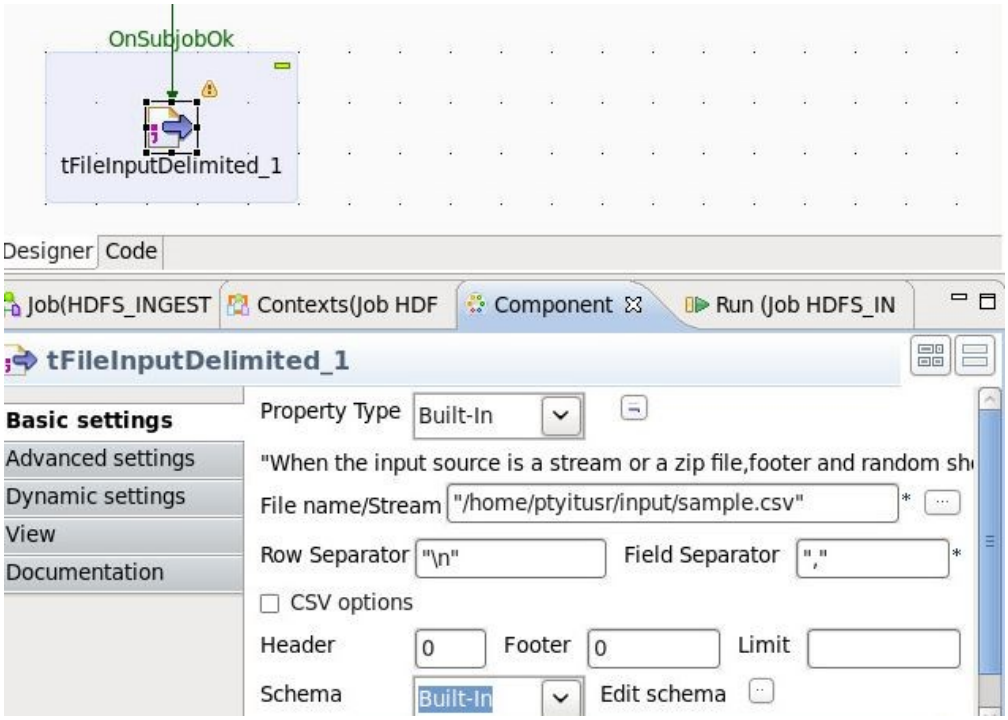
a. Create a *sample.csv* file in the local file system directory */home/ptyitusr/input/* with the following entries.

```
101,Adam,Wiley,Sales
102,Brian,Chester,Service
103,Julian,Cross,Sales
104,Dylan,Moore,Marketing
105,Chris,Murphy,Service
106,Brian,Collingwood,Service
107,Michael,Muster,Marketing
108,Miley,Rhodes,Sales
109,Chris,Coughlan,Sales
110,Aaron,King,Marketng
111,Adam,Young,Service
112,Tyler,White,Sales
113,Martin,Reeves,Service
114,Michael,Morton,Sales
```

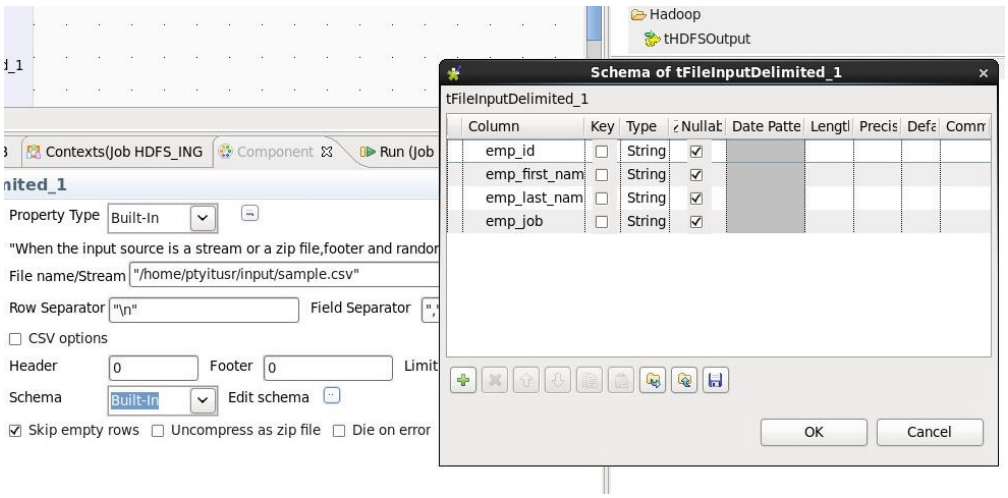
b. Enter the input directory location, as required.

c. Enter the *Row Separator* option, as required.

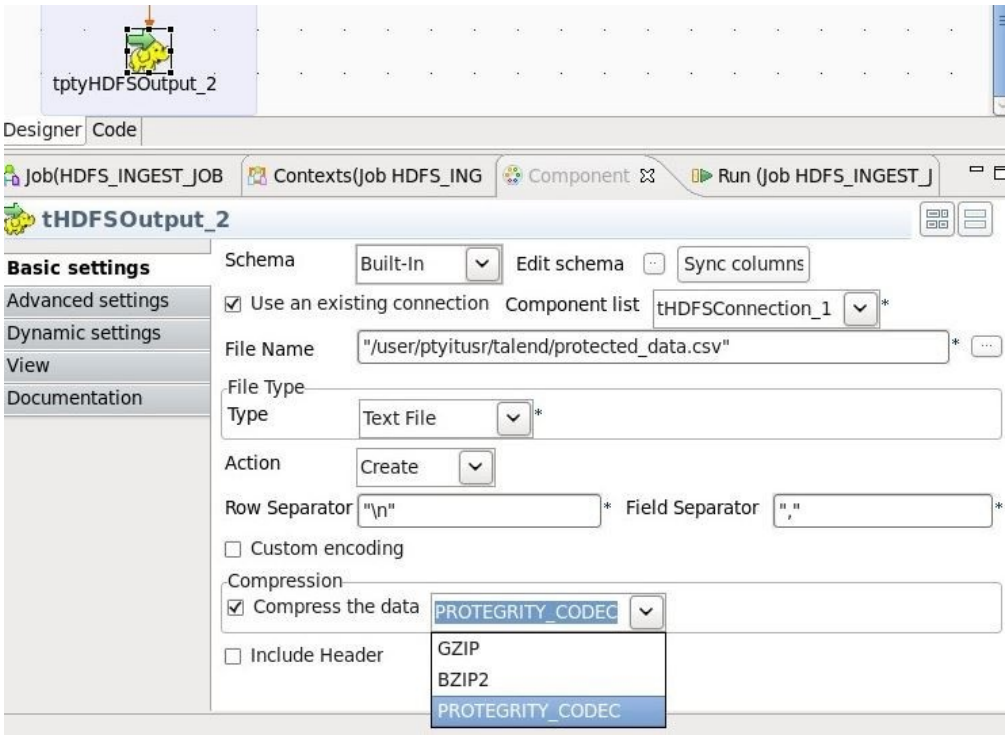
d. Enter the *Field Separator* option, as required.



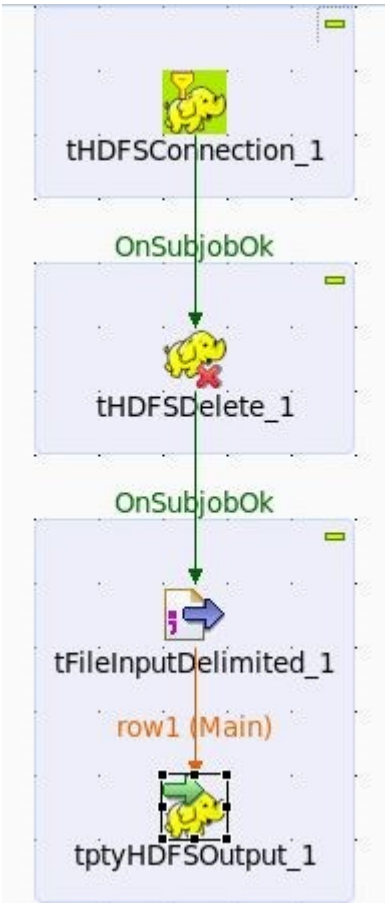
e. Edit the schema for the input supplied, which would be required while writing data to the HDFS path.



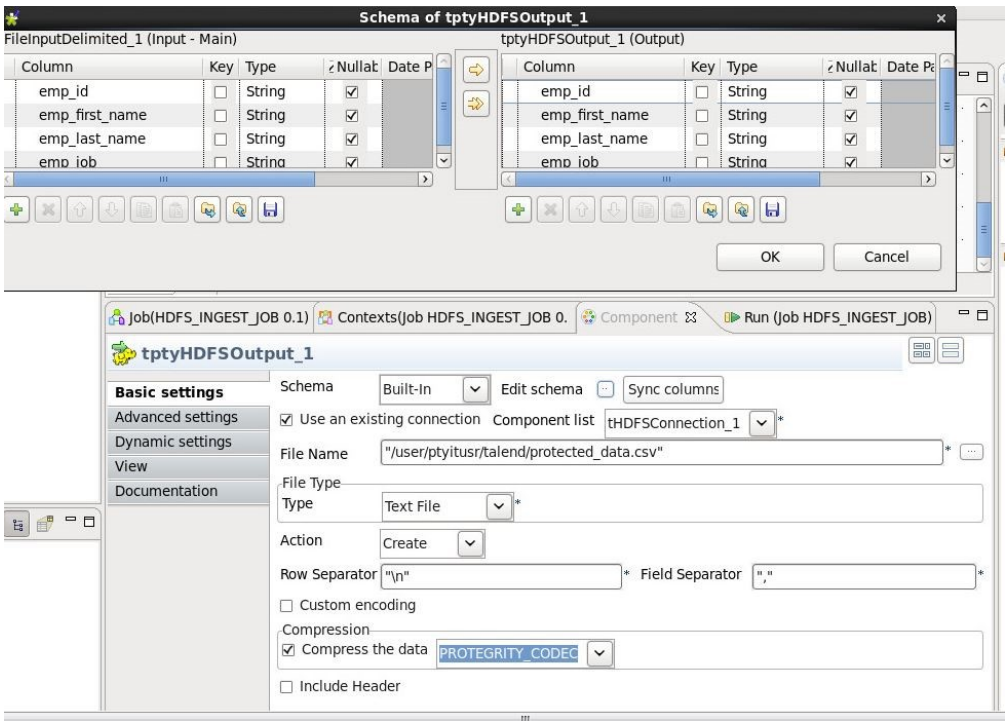
9. Create the *tptyHDFSOutput* component.
   a. Click the *Use an existing connection* box to utilize the existing connection.
   b. Enter the file location in HDFS, where the data needs to be protected and stored.
   c. Enter the *Row Separator* option, as required.
   d. Enter the *Field Separator* option, as required.
   e. Click *Compress the data* box.
   f. Select **PROTEGRITY_CODEC** from the drop down.



   g. Connect the *tFileInputDelimited_1* component to the *tptyHDFSOutput_1* component as illustrated in the following figure.

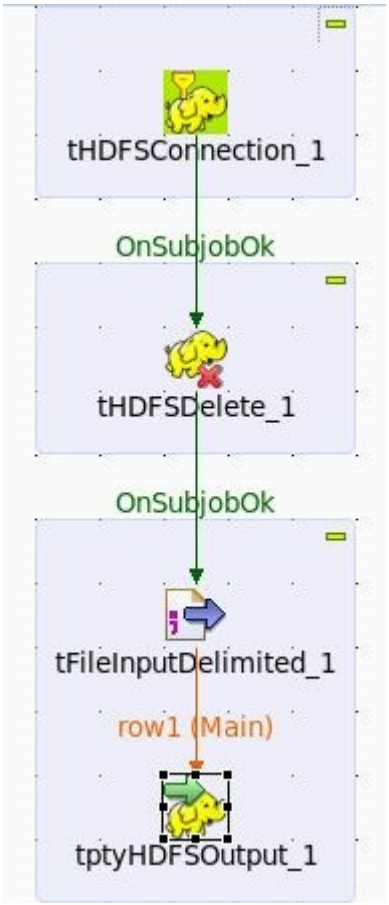


h. Click **Edit schema** to map the input supplied and output file, which will be stored in the HDFS path.



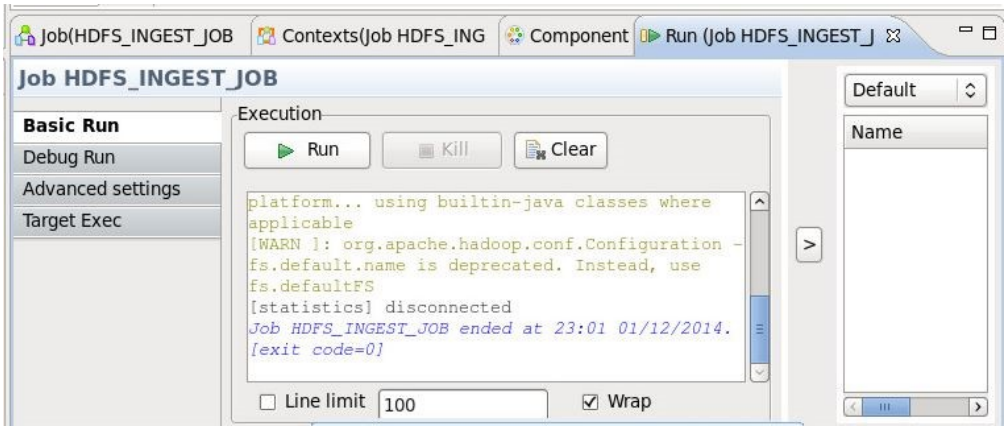
i. Click **OK**.

10. Connect all the components in the job diagram, as illustrated by the following figure.



11. Click on the *Run* tab.
    The Run tab appears.

12. Click **Run** to run the job.
    Check the job execution status in the console.



13. Login to the system with the user *ptyitusr* using the following command.
    ```
    >> su - ptyitusr
    ```
14. List the file present in the HDFS directory */user/ptyitusr/talend/* using the following command.
    ```
    >> hadoop fs -ls /user/ptyitusr/talend/
    ```
15. Read the *protected_data.csv* file from the HDFS directory */user/ptyitusr/talend/* using the following command.
    ```
    >> hadoop fs -cat /user/ptyitusr/talend/protected_data.csv
    ```

The protected data appears.

```
[root@cdh45-1 ~]# su - ptyitusr
-sh-4.1$ hadoop fs -ls /user/ptyitusr/talend/
Found 1 items
-rw-r--r--   3 ptyitusr ptyitusrgroup        451 2014-12-01 23:56 /user/ptyitusr/talend/protected_data.csv
-sh-4.1$ hadoop fs -cat /user/ptyitusr/talend/protected_data.csv
}Ô,v@+ý)¡°0ýFÔuÍ[0dÏÃHI·Tx¦·òÉØ|hÜü×"þëtÍ]×d¿Ub°Ð#±FìÍHÞæ´0K%,ñ{\LG-.àí þ
                                                                                +}ípÒÓ:ÂïÃÑdbõq0dËÉÛ=·iïäA°-6¤ ¾½P0pÎoe{·
;ûN®óý)±È6áxÏ,,/Ô
                ,Í!ÓdYán9(!¹°
                              @ÅÇîjÉºàCí6+Ï¼{Ó"p/çýðÔò[e2ÃtãWòú«`U&©] ý¶©ÞÚÉ=yvÕÑs{°E(P(È¨;±§Ûk#Ò~¦d²â-sh-4.1$
-sh-4.1$
```

10.7 Accessing Data from the Protected Directory in HDFS

➤ To ingest Cleartext Data into the Target HDFS Directory:

1. Click the **Job** link under the **Create a new** section.

   The **New Job** window appears.

2. Type the following in the **Name** box.

   HDFS_READ_DATA

3. Enter a description in the **Description** box.
4. Click **Finish**.

   The new job in Talend is created.

5. Double-click the *tHDFSConnection* component to create the HDFS connection.

   Enter the following properties for the connection, as required:

   - Distribution – The distribution name
   - Hadoop version – The version required for the Hadoop cluster
   - NameNode URI – The domain name and port of the Name node to connect to HDFS
   - User name – The user name to perform the HDFSFP operations. For instance, we have the user as *ptyitusr*.
   - target.input.dir – The targeted HDFS directory to read the protected data. For instance, we have the HDFS directory as */user/ptyitusr/talend*.

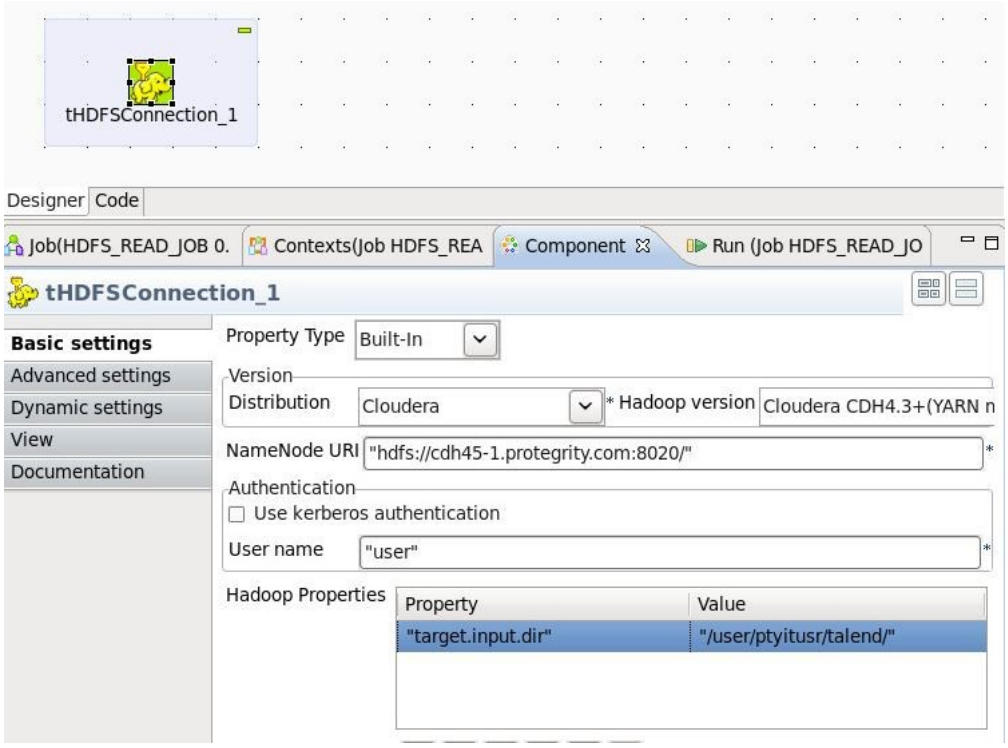
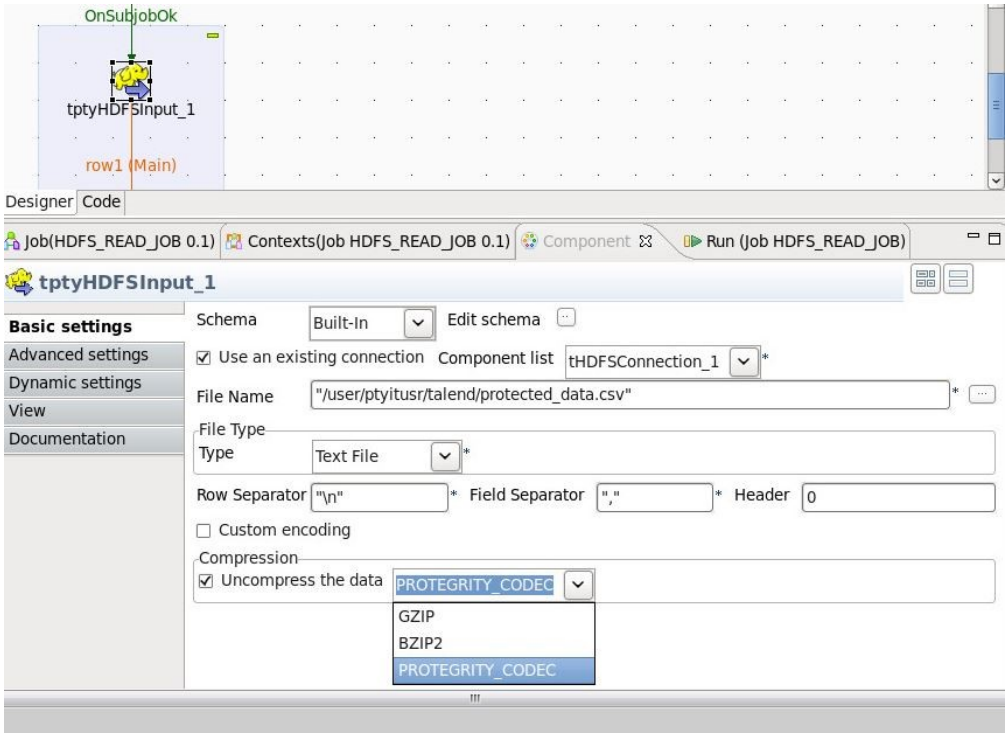


Figure 10-4:

6. Create the *tptyHDFSInput* component.
   a. Select the *Use an existing connection box* to reuse the existing connection.
   b. Enter the HDFS location where the protected data is to be read, as required.
   c. Enter the *Row Separator* option, as required.
   d. Enter the *Field Separator* option, as required.
   e. Click *Uncompress* the data box.
   f. Select **PROTEGRITY_CODEC** from the drop down.

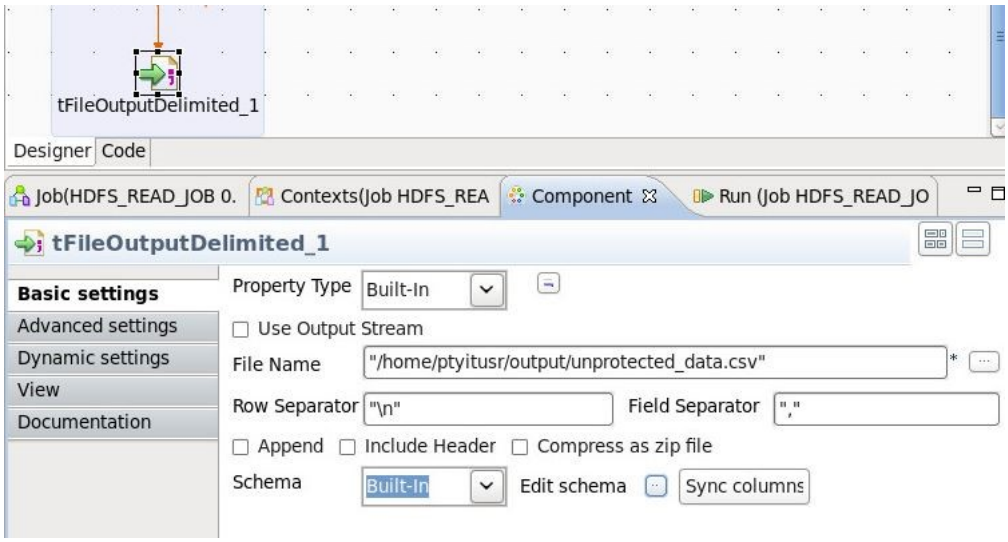


g. Create the *tFileOutputDelimited* component.

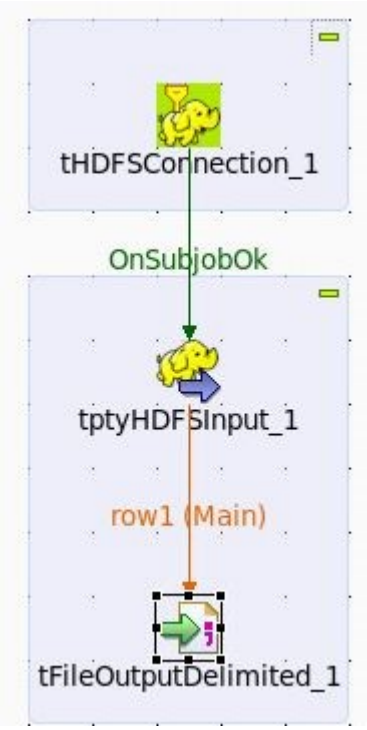
h. Enter the local file system location, where the cleartext data needs to be stored.

i. Enter the *Row Separator* option, as required.

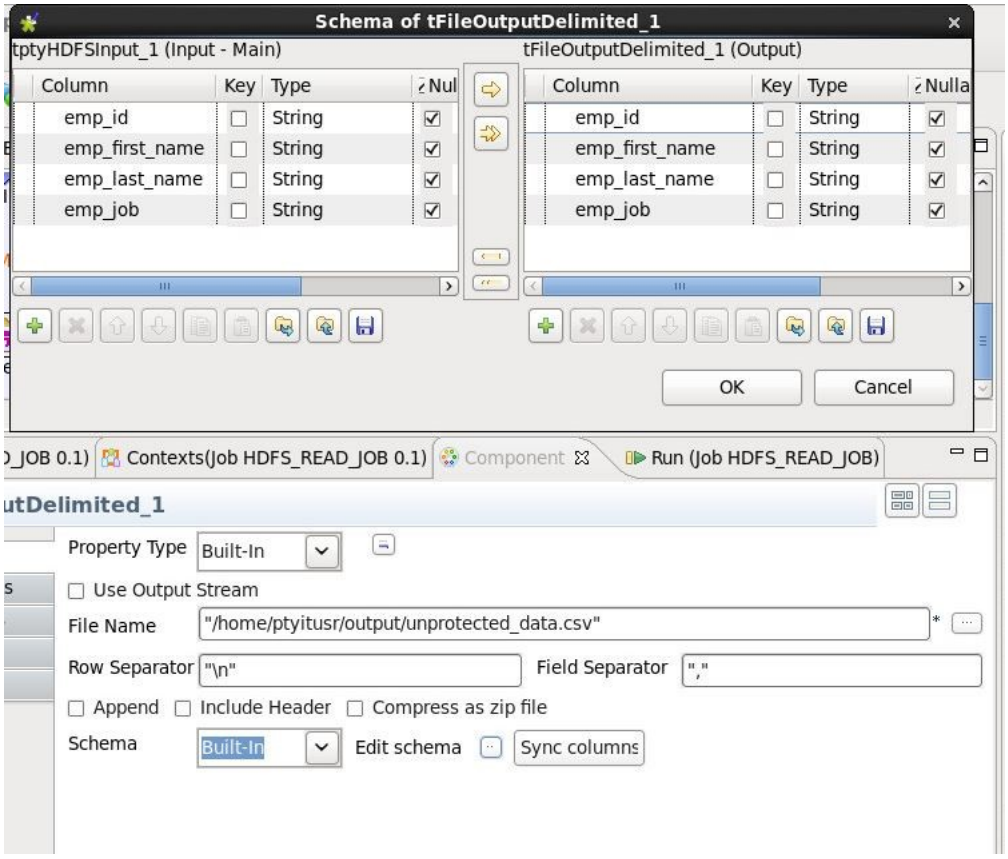
j. Enter the *Field Separator* option, as required.



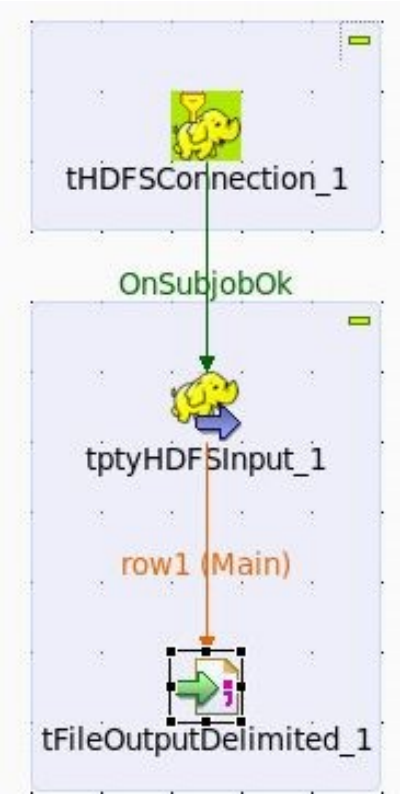
k. Connect the *tptyHDFSInput_1* component to the *tFileOutputDelimited_1* component as illustrated in the following figure.



l. Click **Edit schema** to map the fields in the protected data in the HDFS directory to the local file location, which will store the cleartext data.

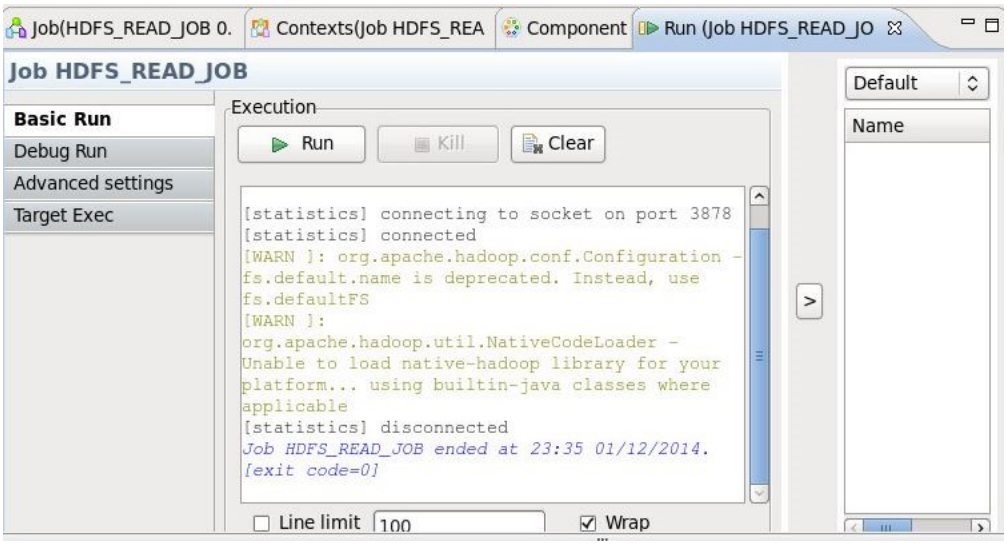


7. Connect all the components in the job diagram, as illustrated by the following figure.



8. Click on the *Run* tab.
   The Run tab appears.

9. Click **Run** to run the job.
   Check the job execution status in the console.



10. Login to the system with the user *ptyitusr* using the following command.
    ```
    >> su - ptyitusr
    ```

11. Navigate to the */home/ptyitusr/output* directory using the following command.
    ```
    >> cd /home/ptyitusr/output
    ```

12. Read the *unprotected_data.csv* file from the HDFS directory */home/ptyitusr/output/* using the following command.
    ```
    cat unprotected_data.csv
    ```

The cleartext data appears.

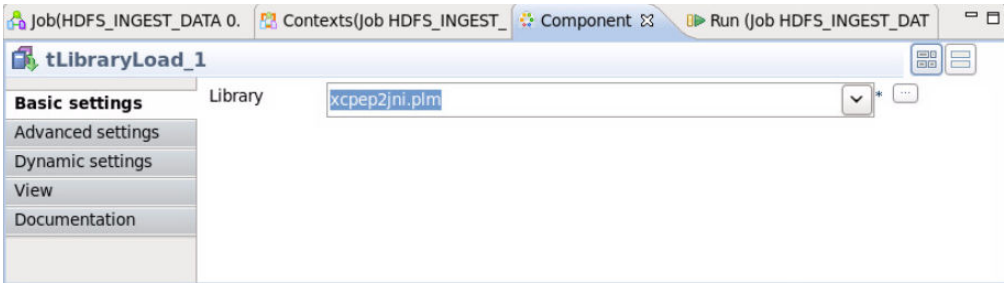
```
[root@cdh45-1 ~]# su - ptyitusr
-sh-4.1$ cd output/
-sh-4.1$ pwd
/home/ptyitusr/output
-sh-4.1$ ll
total 4
-rw-r--r-- 1 root root 373 Dec  1 23:35 unprotected_data.csv
-sh-4.1$ cat unprotected_data.csv
101,Adam,Wiley,Sales
102,Brian,Chester,Service
103,Julian,Cross,Sales
104,Dylan,Moore,Marketing
105,Chris,Murphy,Service
106,Brian,Collingwood,Service
107,Michael,Muster,Marketing
108,Miley,Rhodes,Sales
109,Chris,Coughlan,Sales
110,Aaron,King,Marketng
111,Adam,Young,Service
112,Tyler,White,Sales
113,Martin,Reeves,Service
114,Michael,Morton,Sales
115,Tim,Rhodes,Marketing
```

10.8 Configuring Talend Jobs to run with HDFSFP with Target Exec as Remote

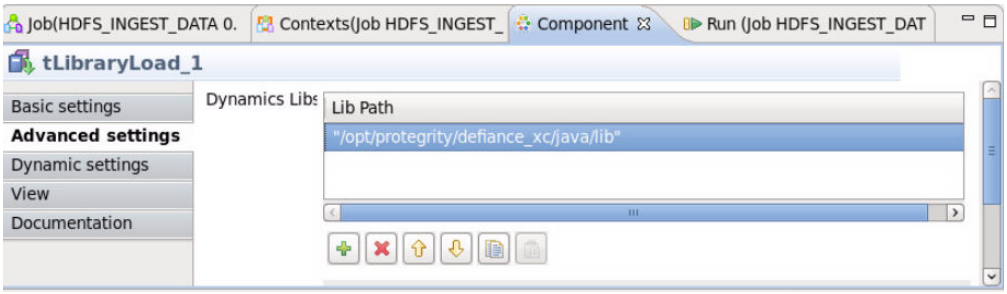
To configure Talend to run jobs with HDFSFP remotely:

1. Connect the *tLibraryLoad* component in the job diagram, as illustrated by the following figure.e

2. In the **Basic settings** tab for the *tLibraryLoad* component, load the *xcpep2jni.plm* file, as illustrated by the following figure.

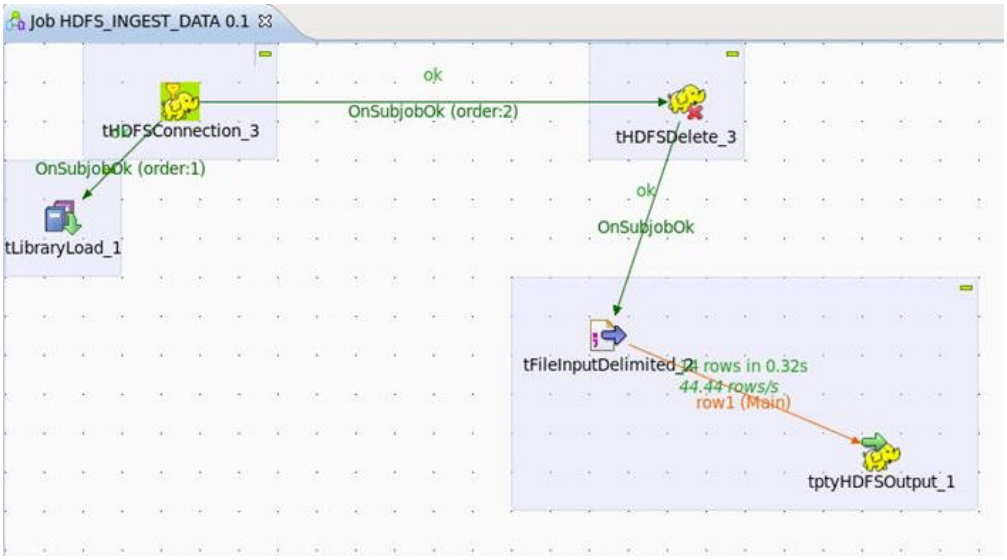


3. In the **Advanced settings** tab for the tLibraryLoad component, set the dynamic library path to */opt/protegrity/defiance_xc/java/lib*, as illustrated by the following figure.

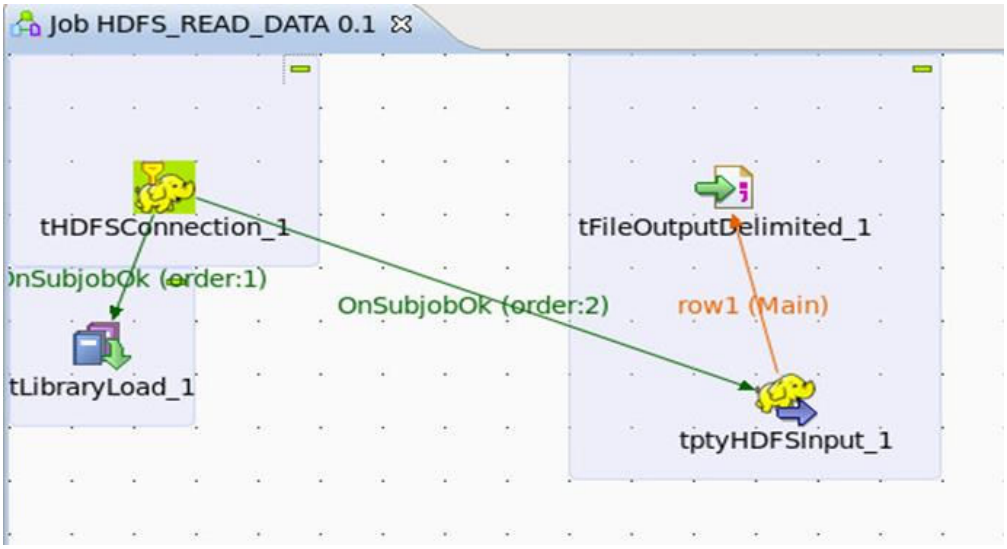


4. Copy the *beuler.properties* file from the *<TALEND_Installation_DIR>/hdfsfp* directory to the home directory of the user, which is being used to run the Talend job server.
5. Ensure that the setting for Target Exec is set to *Remote server*, as illustrated by the following figure.

6. If you are performing a *protect* operation, then connect all the components in the job diagram, as illustrated by the following figure.



7. If you are performing an *unprotect* operation, then connect all the components in the job diagram, as illustrated by the following figure.



10.9 Using Talend with HDFSFP and MapReduce

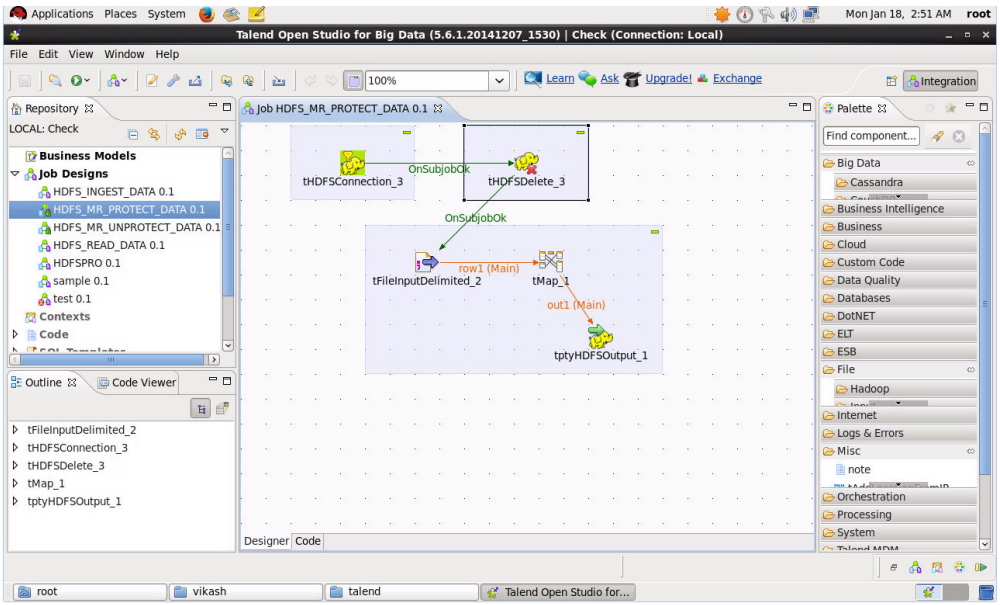
If you need to use Talked with HDFSFP and MapReduce, then perform the following action.

- Create a Routine in Talend Studio, which contains the Java code for invoking the MapReduce Protector APIs for Protect or Unprotect.
- Utilize the *tMap* component from the palette to perform the operations for processing the data.

10.9.1 Protecting Data Using Talend with HDFSFP and MapReduce

➤ To protect data using Talend with HDFSFP and MapReduce:

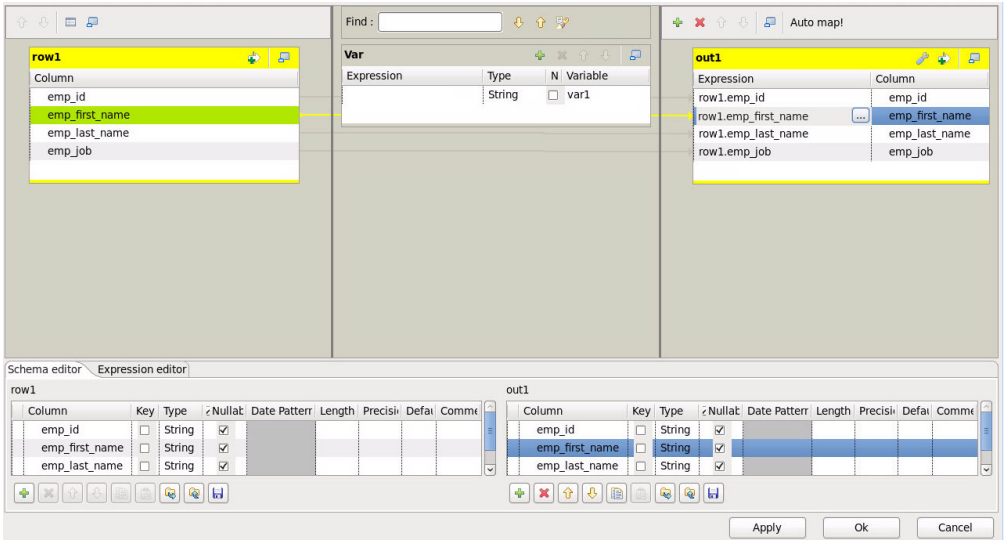
1. Access the *.csv* file from the local directory which contains cleartext data.
2. Protect the fields in the *.csv* file using the required token elements.
3. To ingest the protected data into the protected HDFS directory, where HDFSFP encryption takes place, connect all components in the job diagram, as illustrated by the following figure.



The routine and the required properties to protect the data reside in the *tMap* component, before it is loaded into an HDFSFP protected table.

4. To view the contents of the *tMap* component, double-click on *tMap*.

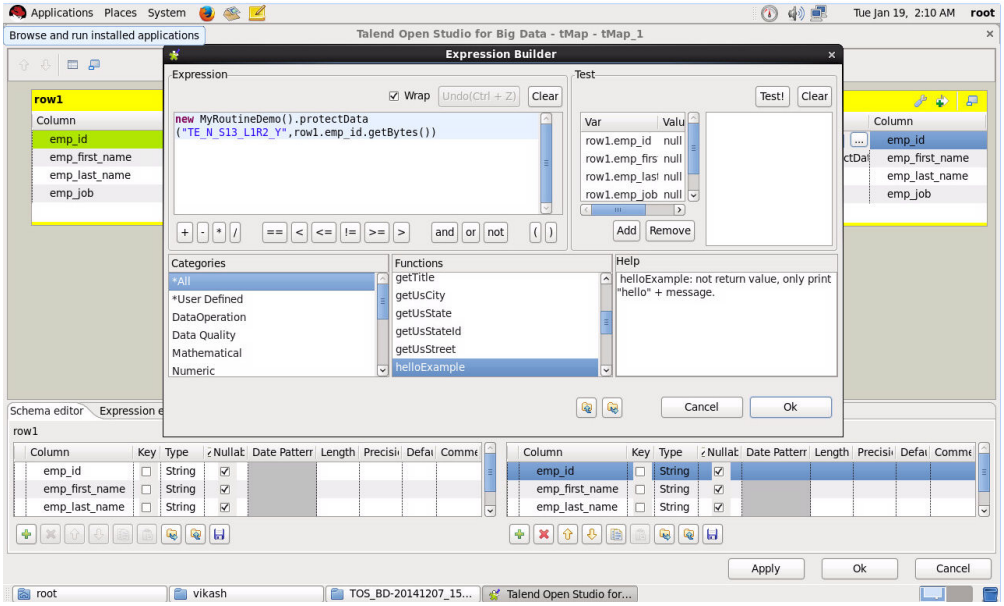
   The following screen listing the contents of the *tMap* component appears.



The left pane lists the table with four columns and the right pane lists the routines called for protecting the data.

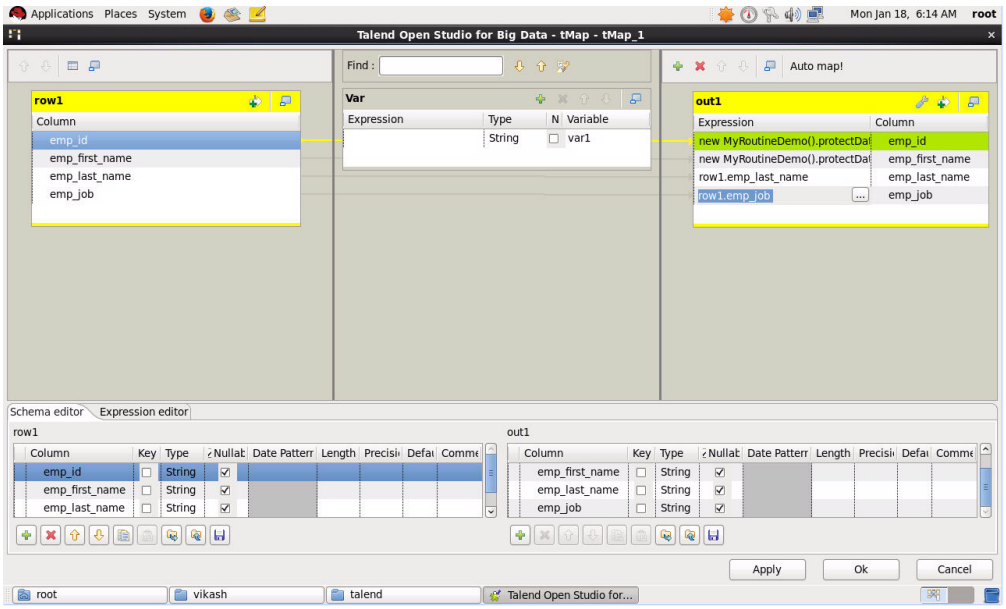
5. To protect the data in the first two rows of the table, click the button beside the rows in the right pane, and as illustrated in the following figure, call the class and functions from the routine using the following syntax,

   new class_name().function name("<Data Element>",<row_no.>.<row.name>)



In the example, the first row and second rows are protected using the function *protectData()* and the token elements *TE_N_S13_L1R2_Y and TE_LASCII_L2R1_Y* respectively.

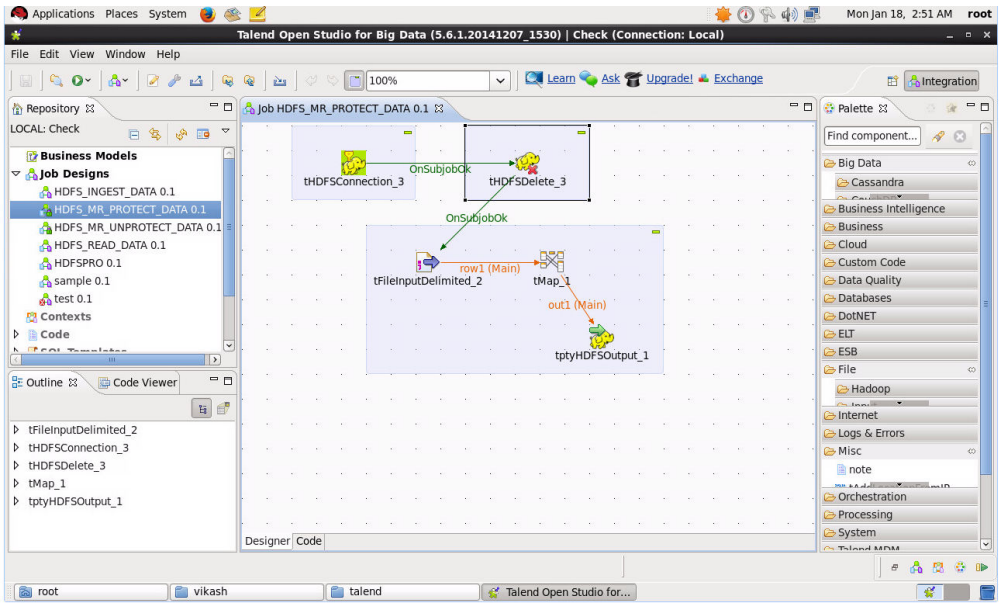
6. After the routine is called for the rows which need to be protected, the right pane illustrates how the *protect* function is called from the Routine.



10.9.2 Unprotecting Data Using Talend with HDFSFP and MapReduce

➤ To unprotect data using Talend with HDFSFP and MapReduce:

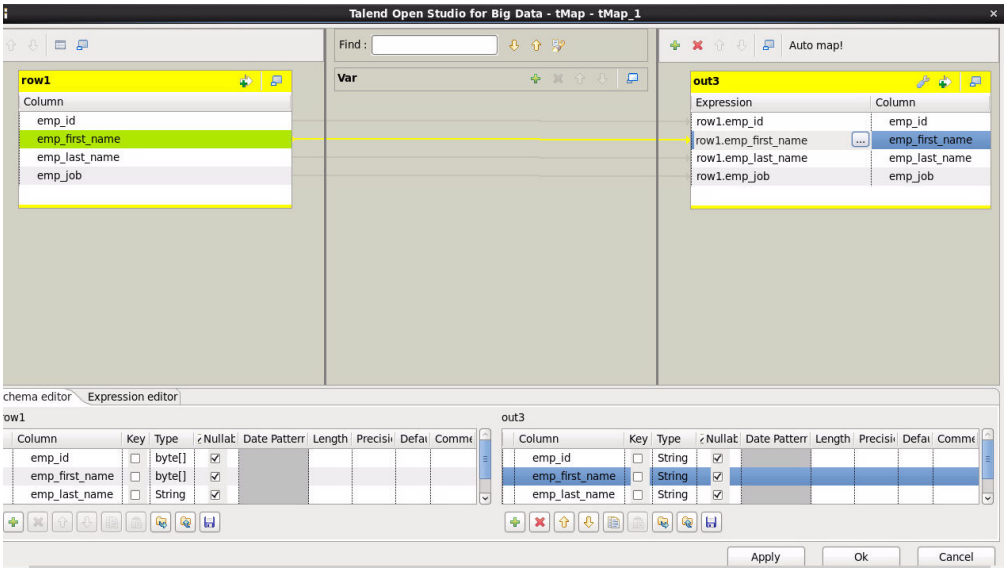
1. Access the protected data from the protected HDFS directory, where HDFSFP decryption takes place.
2. Unprotect the fields in the *.csv* file using the required token elements.
3. To ingest the unprotected data into an output *.csv* file in the local directory, connect all components in the job diagram, as illustrated by the following figure.



The routine and the required properties to protect the data reside in the *tMap* component, before it is loaded into an HDFSFP protected table.

4. To view the contents of the *tMap* component, double-click on *tMap*.

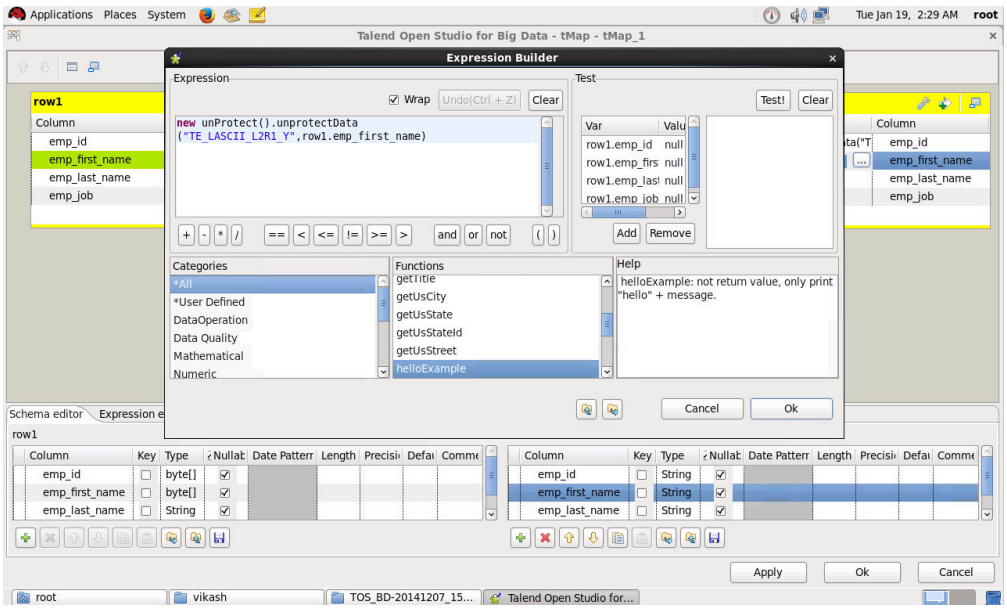
   The following screen listing the contents of the *tMap* component appears.



The left pane lists the table with four columns and the right pane lists the routines called for unprotecting the data.

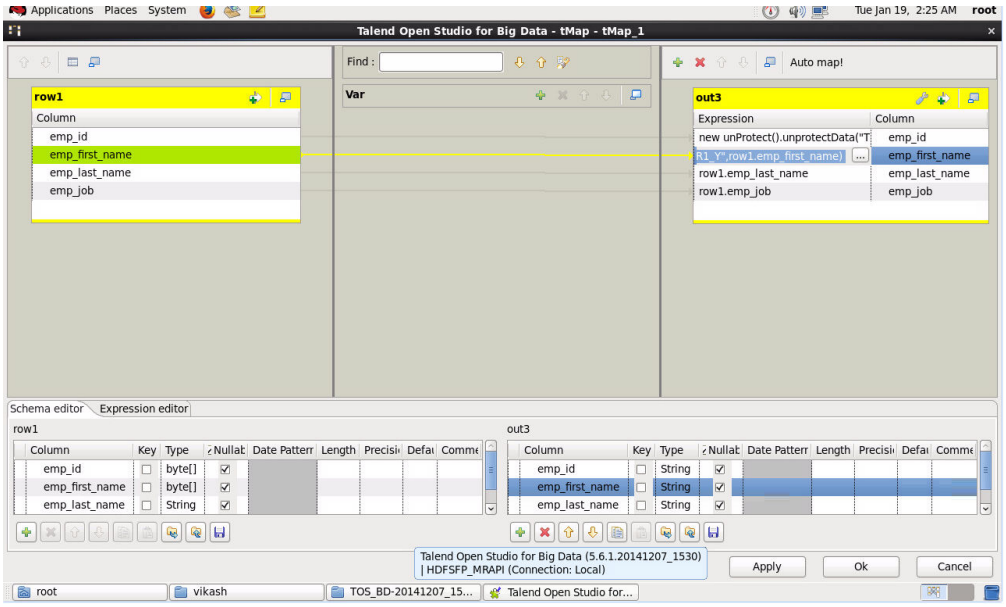
5. To unprotect the data in the first two rows of the table, click the button beside the rows in the right pane, and as illustrated in the following figure, call the class and functions from the routine using the following syntax,

   new class_name().function name("<Data Element>",<row_no.>.<row.name>)



In the example, the first row and second rows are unprotected using the function *unprotectData()* and the token elements *TE_N_S13_L1R2_Y* and *TE_LASCII_L2R1_Y* respectively.

6. After the routine is called for the rows which need to be unprotected, the right pane illustrates how the *unprotect* function is called from the Routine.



10.9.3 Sample Code Usage

The MapReduce sample routine, described in this section, is an example on how to use the Protegrity MapReduce protector APIs. The sample program utilizes the following two routine files:

- **Routine Item** – The main routine calls the Mapper job.
- **Routine Properties** – The properties related to the main routine.

10.9.3.1 Routine Item

Routine Item

```
package routines;

import com.protegrity.hadoop.fileprotector.fs.ProtectorException;
import com.protegrity.hadoop.fileprotector.fs.PtyHdfsProtector;
import com.protegrity.hadoop.mapreduce.ptyMapReduceProtector;

public class MyRoutineDemo {

    public static void helloExample(String message) {
        if (message == null) {
            message = "World"; //$NON-NLS-1$
        }
        System.out.println("Hello " + message + " !"); //$NON-NLS-1$ //$NON-NLS-2$
    }

       public static PtyHdfsProtector protector = new PtyHdfsProtector();

       public void copyFromLocalTest(String[] srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copyFromLocal(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void copyToLocalTest(String srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copyToLocal(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void copyTest(String srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.copy(srcs, dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void mkdirTest(String dir)
       {
       boolean result;
       try {
       result = protector.mkdir(dir);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }
       public void moveTest(String srcs, String dstf)
       {
       boolean result;
       try {
       result = protector.move(srcs,dstf);
       } catch (ProtectorException pe) {
       pe.printStackTrace();
       }
       }

       public void deleteFileTest(String file,boolean skipTrash)
       {
       boolean result;
       try {
       result = protector.deleteFile(file, skipTrash);
       } catch (ProtectorException pe) {
```

```
	pe.printStackTrace();
	}
	}
	public void deleteDirTest(String dir,boolean skipTrash)
	{
	boolean result;
	try {
	result = protector.deleteDir(dir, skipTrash);
	} catch (ProtectorException pe) {
	pe.printStackTrace();
	}
	}

	public String  protectData(String dataElement,byte[] data){
		ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
		mapReduceProtector.openSession("0");
		System.out.println(dataElement);
		System.out.println(new String(data));
		byte[] output = mapReduceProtector.protect(dataElement, data);
		return new String(output);
	}

	public String  unprotectData(String dataElement,byte[] data){
		ptyMapReduceProtector mapReduceProtector = new ptyMapReduceProtector();
		mapReduceProtector.openSession("0");
		System.out.println(dataElement);
		System.out.println(new String(data));
		byte[] output = mapReduceProtector.unprotect(dataElement, data);
		return new String(output);
	}

	public static void main(String[] args) {
	MyRoutineDemo protect = new MyRoutineDemo();
	// Ingest Local Data into HDFS
	String srcsCFL[] = new String[2];
	srcsCFL[0] ="/home/ptyitusr/input/sample.csv";
	srcsCFL[1] ="/home/ptyitusr/input/test.csv";
	String dstfCFL ="/user/ptyitusr/talend";
	protect.copyFromLocalTest(srcsCFL, dstfCFL);

	// Extract HDFS file to Local
	String srcsCTL= "/user/ptyitusr/talend/prot.csv";
	String dstfCTL = "/home/ptyitusr/input";
	protect.copyToLocalTest(srcsCTL, dstfCTL);
	// Copy File from HDFS to HDFS
	String srcsCopy="/user/ptyitusr/talend/prot.csv";
	String dstfCopy ="/user/ptyitusr/talend";
	protect.copyTest(srcsCopy, dstfCopy);
	// Create HDFS Sub-Directory
	String dir = "/user/ptyitusr/talend/sub";
	protect.mkdirTest(dir);
	// Move from HDFS to HDFS
	String srcsMove = "/user/ptyitusr/talend/prot.csv";
	String dstfMove = "/user/ptyitusr/talend";
	protect.moveTest(srcsMove, dstfMove);
	// Delete File from HDFS
	String fileDelete = "/user/ptyitusr/talend/prot.csv";
	boolean skipTrashFile = false;
	protect.deleteFileTest(fileDelete,skipTrashFile);
	// Delete Sub-Directory and Children from HDFS
	String dirDelete = "/user/ptyitusr/talend";
	boolean skipTrashDir = false;
	protect.deleteDirTest(dirDelete,skipTrashDir);
	}
	}
```

10.9.3.2 Routine Properties

Routine Properties

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:TalendProperties="http://
```

```
www.talend.org/properties">
  <TalendProperties:Property xmi:id="_DTUMYDIkEeW6nfU7n79eDQ" id="_DTS-
QDIkEeW6nfU7n79eDQ" label="MyRoutineDemo" creationDate="2015-07-24T11:50:12.646-0500"
modificationDate="2016-01-19T02:06:24.583-0500" version="0.1" statusCode=""
item="_DTUMYjIkEeW6nfU7n79eDQ" maxInformationLevel="WARN" displayName="MyRoutineDemo">
    <author href="../../talend.project#_5-CdsDHkEeWZEZzbK6p_uA"/>
    <informations xmi:id="_-Y8DoDJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y8DoTJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y8DojJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y8DozJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y9RwDJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y9RwTJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
    <informations xmi:id="_-Y9RwjJQEeW6nfU7n79eDQ" level="WARN" text="The local variable
result is never read"/>
  </TalendProperties:Property>
  <TalendProperties:ItemState xmi:id="_DTUMYTIkEeW6nfU7n79eDQ" path=""/>
  <TalendProperties:RoutineItem xmi:id="_DTUMYjIkEeW6nfU7n79eDQ"
property="_DTUMYDIkEeW6nfU7n79eDQ" state="_DTUMYTIkEeW6nfU7n79eDQ">
    <content href="MyRoutineDemo_0.1.item#/"/>
    <imports xmi:id="_uWRN4DInEeW6nfU7n79eDQ" mESSAGE="" mODULE="hdfsfp.jar"
nAME="MyRoutineDemo" rEQUIRED="true" urlPath="/TalendInstall/Talend-5.6.1/hdfsfp/hdfsfp.jar"/>
    <imports xmi:id="_NMUX8L30EeWz-4GwVQX7Ig" mESSAGE="" mODULE="pepmapreduce-2.7.1.jar"
nAME="MyRoutineDemo" rEQUIRED="true" urlPath="/opt/protegrity/hadoop_protector/lib/
pepmapreduce-2.7.1.jar"/>
  </TalendProperties:RoutineItem>
</xmi:XMI>
```

Appendix A

Appendix: Migrating Tokenized Unicode Data from and to a Teradata Database

This section describes the procedures for migrating tokenized Unicode data from and to a Teradata database.

Note: This section is only applicable for Legacy Unicode and Base64 Unicode data element.

Note: This section considers the Teradata database for reference.

Note: In addition to the Teradata database, the Big Data Protector works with other databases, such as Netezza, Greenplum, and so on.

11.1 Migrating Tokenized Unicode Data from a Teradata Database

This section describes the task to unprotect the tokenized Unicode data in Hive, Impala, or Spark, which was tokenized in the Teradata database using the Protegrity Database Protector and then migrated to Hive, Impala, MapReduce, or Spark.

Note:
Ensure that the data elements used in the data security policy, deployed on the Teradata Database Protector and Big Data Protector machines are uniform.

11.1.1 Migrating Tokenized Unicode data from Teradata database to Hive or Impala and unprotecting it using Hive or Impala protector

➤ To migrate Tokenized Unicode data from Teradata database to Hive or Impala and unprotect it using Hive or Impala protector:

1. Tokenize the Unicode data in the Teradata database using Protegrity Database Protector.
2. Migrate the tokenized Unicode data from the Teradata database to Hive or Impala.
3. To unprotect the tokenized Unicode data on Hive or Impala, ensure that the following UDFs are used, as required:
   - Hive: *ptyUnprotectUnicode()*
   - Impala: *pty_UnicodeStringSel()*

11.1.2 Migrating Tokenized Unicode data from a Teradata database to Hadoop and Unprotecting it using MapReduce or Spark protector

➤ To migrate Tokenized Unicode data from a Teradata database to Hadoop and unprotect it using MapReduce or Spark protector:

1. Migrate the tokenized Unicode data to the Hadoop ecosystem using any data migration utilities.
2. To unprotect the tokenized Unicode data using MapReduce or Spark, ensure that the following APIs are used, as required:
   - MapReduce: *public byte[] unprotect(String dataElement, byte[] data)*
   - Spark: *void unprotect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)*
3. Convert the protected tokens to bytes using UTF-8 encoding.
4. Send the data as input to the Unprotect API in the MapReduce or Spark protector, as required.
5. Convert the unprotected output in bytes to *String* using UTF-16LE encoding.

   The *string* data will display the data in cleartext format.

   The following sample code snippet describes how to unprotect the Tokenized Unicode data, that is migrated from a Teradata database to Hadoop, using the MapReduce or Spark protector.

```
private Protector protector = null;
String[] unprotectinput= new String[SIZE] ;
byte[][] inputValueByte = new byte [unprotectinput.length][];
StringBuilder unprotectedString = new StringBuilder();
int x=0;
for (x=0; x< unprotectinput.length; x++)
inputValueByte[x]= unprotectinput[x].getBytes(StandardCharsets.UTF_8);  // Point a
implementation
protector.unprotect(DATAELEMENT_NAME, errorIndexList, inputValueByte,
outputValueByte); // Point b implementation
unprotectedString.apprend(new String(outputValueByte[j],StandardCharsets.UTF_16LE))//
Point c implementation
```

11.2 Migrating Tokenized Unicode Data to a Teradata Database

This section describes the task to protect Unicode data in Hive, Impala, MapReduce, or Spark, migrate it to a Teradata database, and then unprotect the tokenized Unicode data using the Protegrity Database Protector.

> **Note:**
> Ensure that the data elements used in the data security policy, deployed on the Teradata Database Protector and Big Data Protector machines are uniform.

11.2.1 Migrating Tokenized Unicode data using Hive or Impala protector to Teradata database

To migrate Tokenized Unicode data using Hive or Impala protector to Teradata database:

1. To protect the Unicode data on Hive or Impala, ensure that the following UDFs are used, as required:
   - Hive: *ptyProtectUnicode()*
   - Impala: *pty_UnicodeStringIns()*
2. Migrate the tokenized Unicode data from Hive or Impala to the Teradata database.
3. To unprotect the tokenized Unicode data in the Teradata database, use the Protegrity Database Protector.

11.2.2 Protecting Unicode data using MapReduce or Spark protector and Migrating it to a Teradata database

To protect Unicode data using MapReduce or Spark protector and migrate it to a Teradata database:

1. Convert the cleartext format Unicode data to bytes using UTF-16LE encoding.
2. To migrate the tokenized Unicode data using MapReduce or Spark to the Teradata database, ensure that the following APIs are used, as required:
   - MapReduce: *public byte[] protect(String dataElement, byte[] data)*
   - Spark: *void protect(String dataElement, List<Integer> errorIndex, byte[][] input, byte[][] output)*
3. Send the data as input to the Protect API in the MapReduce or Spark protector, as required.
4. Convert the protected output in bytes to *String* using UTF-8 encoding.

   The output is protected tokenized data.

5. Migrate the protected data to the Teradata database using any data migration utilities.

   The following sample code snippet describes how to protect Unicode data using the MapReduce or Spark protector, and migrating it to a Teradata database.

```
private Protector protector = null;
String[] clear_data = new String[SIZE] ;
byte[][] inputValueByte = new byte [clear_data.length][];
StringBuilder protectedString = new StringBuilder();
inputValueByte= data.getBytes(StandardCharsets.UTF_16LE); //Point a implementation
protector.protect(DATAELEMENT_NAME, errorIndexList, inputValueByte, outputValueByte); //
Point b implementation
int x=0;
for (x=0; x<outputValueByte.length; x++)
protectedString.append(new String(outputValueByte[x],StandardCharsets.UTF_8)); //Point c
implementation
```

Appendix B

Appendix: Using Sample Data

This section provides examples of sample data that you must create for testing the Big Data protector.

12.1 Sample Roles

The sample roles that you must create in the ESA are described in the following table.

Table B-1: List of Sample Roles

Roles	User	Role Description
SAMPLE_ADMIN	root	This user is able to protect and unprotect the data, and access the data in the cleartext form.
SAMPLE_INGESTION_USER	John	This user is allowed to ingest the data and protect the sensitive data.
SAMPLE_ANALYST	Fred	This user is able to unprotect the Name and Amount fields and is able to access the other fields in protected form.

12.2 Sample Data Elements in the Security Policy

The sample data elements that you must create in the ESA are described in the following table.

Table B-2: List of Sample Data Elements

Data Element	Data Securing Method	Data Element Type	Input Accepted	Description
TOK_NAME	Tokenization	Alpha-Numeric	Alphabetic symbols including lowercase (a-z) and uppercase letters (A-Z)	Data element for the *Customer Name*.

Data Element	Data Securing Method	Data Element Type	Input Accepted	Description
			and digits from 0 through 9. Min length: 1 Max length: 4080	
TOK_CREDIT_CARD	Tokenization	Creditcard	Digits 0 through 9 with no separators. Min length: 6 Max length: 256	Data element for the *Credit Card* number.
TOK_AMOUNT	Tokenization	Decimal	Digits 0 through 9. The sign (+ or -) and decimal point (. or ,) can be used as separators. Min length: 1 Max length: 36	Data element for the *Amount* spent by the customer using the credit card.
TOK_PHONE	Tokenization	Numeric	Digits 0 through 9 with no separators. Min length: 1 Max length: 3933	Data element for the *Phone* number.

12.3 Role-based Sample Permissions for Data Elements

The role-based sample permissions for the data elements that you must assign in the ESA are described in the following table.

Table B-3: Role-based Sample Permissions for Data Elements

Roles	TOK_AMOUNT	TOK_NAME	TOK_CREDIT_CARD	TOK_PHONE
SAMPLE_ADMIN	Protect Unprotect	Protect Unprotect	Protect Unprotect	Protect Unprotect
SAMPLE_ANALYST_1	Protect	Protect	Protect	Protect
SAMPLE_ANALYST_2	Unprotect	Unprotect	-	-

12.4 Sample Data

The sample data that you must create to test the Big Data Protector is described in the following table.

Table B-4: Sample Fields and Values

ID	Name	Phone	Credit Card	Amount
928724	Hultgren Caylor	9823750987	376235139103947	6959123
928725	Bourne Jose	9823350487	6226600538383292	42964354
928726	Sorce Hatti	9824757883	6226540862865375	7257656
928727	Lorie Garvey	9913730982	5464987835837424	85447788

ID	Name	Phone	Credit Card	Amount
928728	Belva Beeson	9948752198	5539455602750205	59040774
928729	Hultgren Caylor	9823750987	376235139103947	3245234
928730	Bourne Jose	9823350487	6226600538383292	2300567
928731	Lorie Garvey	9913730982	5464987835837424	85447788
928732	Bourne Jose	9823350487	6226600538383292	3096233
928733	Hultgren Caylor	9823750987	376235139103947	5167763
928734	Lorie Garvey	9913730982	5464987835837424	85447788

Appendix C

Appendix: Sample Files for Ambari Alerts

13 Sample *Alerts.json* File

```
{
    "AlertDefinition" :
        {
          "service_name": "BDPPEP",
          "scope": "HOST",
          "enabled": true,
          "description": "check the status of pepserver",
          "name": "check_pep",
          "component_name": "PEP",
          "interval": 1,
          "label": "Pepserver status",
          "source": {
            "path": "checkpep.py",
            "type": "SCRIPT"
          }
        }
  }
```

13 Sample *checkpep.py* File

```
#!/usr/bin/env python

from ambari_commons.os_family_impl import OsFamilyFuncImpl, OsFamilyImpl
from ambari_commons import OSConst
from subprocess import check_output

@OsFamilyFuncImpl(os_family=OsFamilyImpl.DEFAULT)
def execute(configurations={}, parameters={}, host_name=None):
    try:
        pid = check_output(["pidof","pepserver"])
        if pid > 0:
            return ("OK", ["Pepserver is running"])
    except:
        return ("CRITICAL", ["Pepserver is not running"])
```