Table of Contents

Session 2 – Architecture 6.3++ 2
Summary 2
Splunk's architecture 3
The need for parallelization 5
Index parallelization 6
NOTE 8
Search parallelization 9
Pipeline parallelization 9
NOTE 10
The search scheduler 10
Summary parallelization 11
Data integrity control 13
NOTE 13
Intelligent job scheduling 14
TIP 15
The app key-value store 18
System requirements 18
Uses of the key-value store 18
Components of the key-value store 18
Managing key-value store collections via REST 21
EXAMPLES 22
Replication of the key-value store 24
Splunk Enterprise Security 25
Enabling HTTPS for Splunk Web 26
Enabling HTTPS for the Splunk forwarder 27
Securing a password with Splunk 28
The access control list 29
Authentication using SAML 30
NOTE 30

Session 2 – Architecture 6.3++

Splunk is known as the Google of machine log analytics. It is a very powerful, robust, and real-time big data analytics tool. In this chapter, we will study in detail how Splunk works in the backend and what is the backbone of Splunk due to which it can process big data in real time. We will also go through all the new techniques and architectural changes that have been introduced in Splunk 6.3 to make Splunk faster, better, and provide near real-time results.

The following topics will be covered in this chapter:

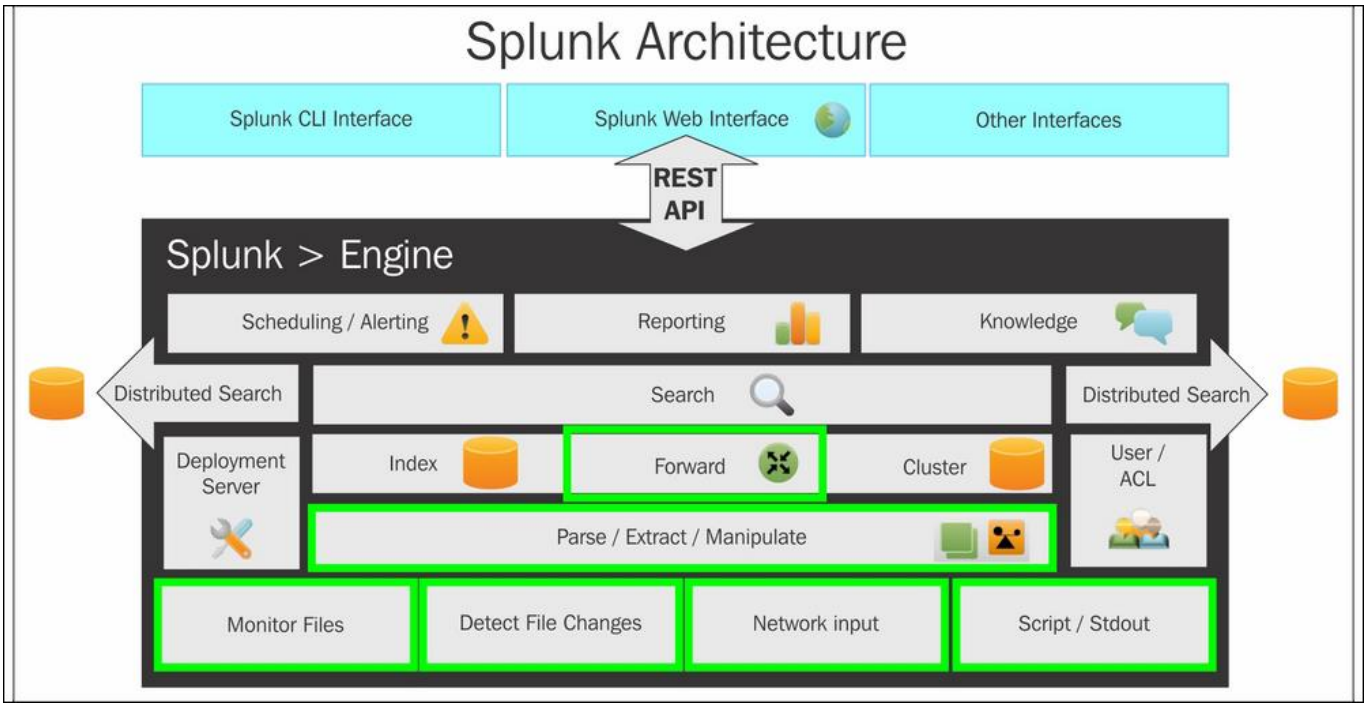
- The architecture
- Index parallelization
- Search parallelization
- Data integrity control
- Intelligent job scheduling
- The app's key-value store
- Securing Splunk Enterprise
- Single sign-on using SAML

Summary

In this chapter, we went through the architectural enhancement done by Splunk in order to speed up data ingestion and indexing to Splunk by utilizing the underutilized resources. We went through index and search parallelization and how it enhances and scales the performance of Splunk. We also went through the details of the data integrity control mechanism and intelligent job scheduling that was introduced in Splunk Enterprise 6.3. Later, we studied how the app key-value store can be used to maintain a state and other information. The last part of this chapter was concentrated on Splunk Enterprise security techniques, implementations, and configuration. We also studied in detail SSO using SAML that was introduced in Splunk 6.3. In the next chapter, we will cover how to create and manage Splunk applications and add-ons.

Splunk's architecture

Splunk's architecture comprises of components that are responsible for data ingestion and indexing and analytics.



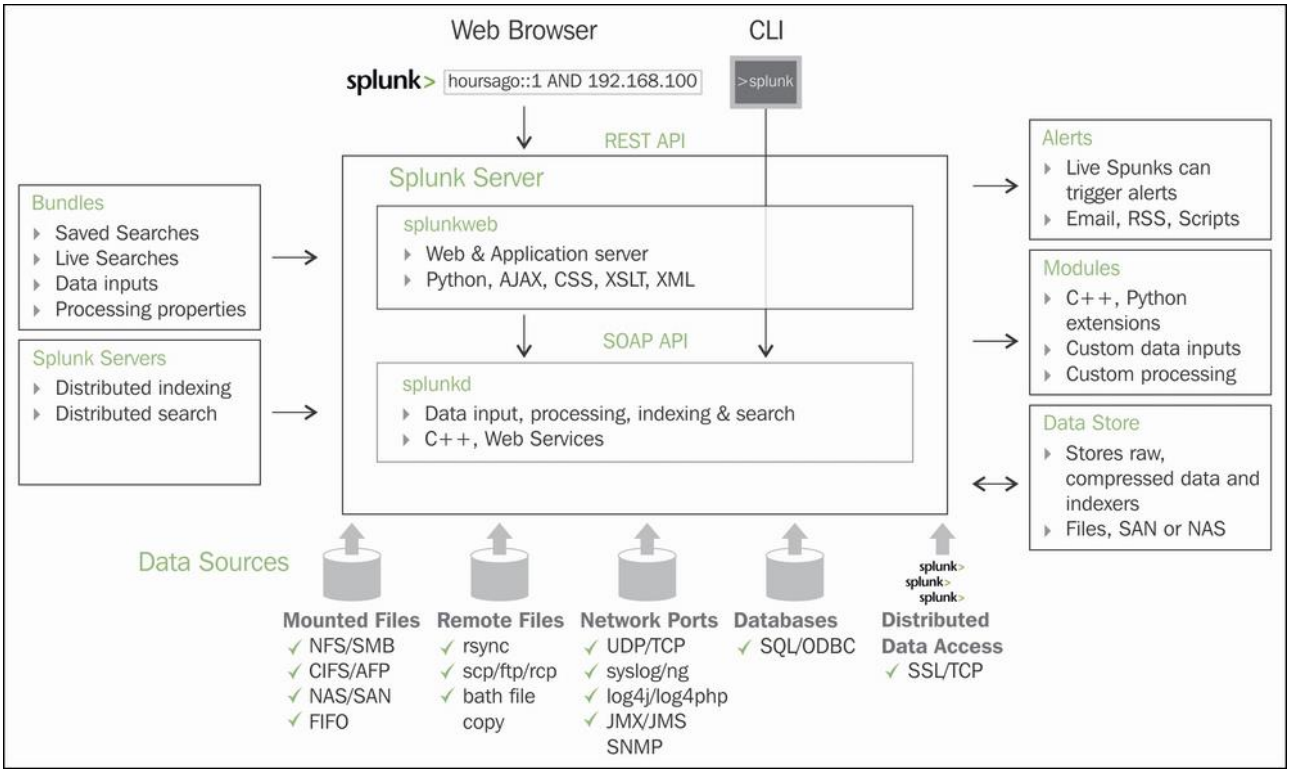
The lowest level of Splunk architecture depicts various data input methods supported by Splunk. These input methods can be configured to send data on Splunk indexers. Before the data reaches Splunk indexers, it can be parsed or manipulated, that is, data cleaning can be done if required. Once the data is indexed on Splunk, the next layer, that is, searching, comes into the picture for analytics over the log data.

Splunk supports two types of deployment: standalone deployment and distributed deployment. Depending on the deployment type, corresponding searches are performed. The Splunk engine has other additional components of knowledge manager, reporting and scheduling, and alerting. The entire Splunk engine is exposed to users via Splunk CLI, Splunk Web Interface, and Splunk SDK, which are supported by most languages.

Splunk installs a distributed server process on the host machine called **splunkd**. This process is responsible for indexing and processing a large amount of data through various sources. splunkd is capable of handling large volumes of streaming data and indexing it for real-time analytics over one or more pipelines.

Every single pipeline comprises of a series of processors, which results in faster and efficient processing of data. Listed below are the blocks of the Splunk architecture:

- **Pipeline**: This is a single-threaded configurable process residing in splunkd.
- **Processors**: They are individual reusable functions that act on incoming data passing through a pipeline. Pipelines exchange data among themselves through a queue.



splunkd allows users to search, navigate, and manage data on Splunk Enterprise through the web interface called Splunk Web. It is a web application server based on Python providing a web interface to use Splunk. In the earlier version of Splunk, splunkd and Splunk Web were two separate processes, but from Splunk 6, both the processes were

integrated in splunkd itself. It allows users to search for, analyze, and visualize data using the web interface. Splunk Web interface can be accessed using the Splunk web port, and Splunk also exposes the REST API for communication via the splunkd management port.

One of the important components of Splunk's architecture is the data store. It is responsible for compressing and storing original (raw) data. The data is stored in **Time Series Index** (**TSIDX**) files. A data store also includes storage and archiving based on the configurable retention policy.

Splunk Enterprise deployments can range from single-server deployments (which index a few gigabytes of data per day and are accessed by a few users who are searching, analyzing, and visualizing the data) to large, distributed enterprise deployments across multiple data centers, indexing hundreds of terabytes of data and searches performed by hundreds of users. Splunk supports communication with another instance of a Splunk Server via TCP to forward data from one Splunk server to another to archive data and various other clustering and data distribution requirements via Splunk-to-Splunk TCP communication.

Bundles are the components of the Splunk architecture that store the configuration of data input, user accounts, Splunk applications, add-ons, and various other environment configurations.

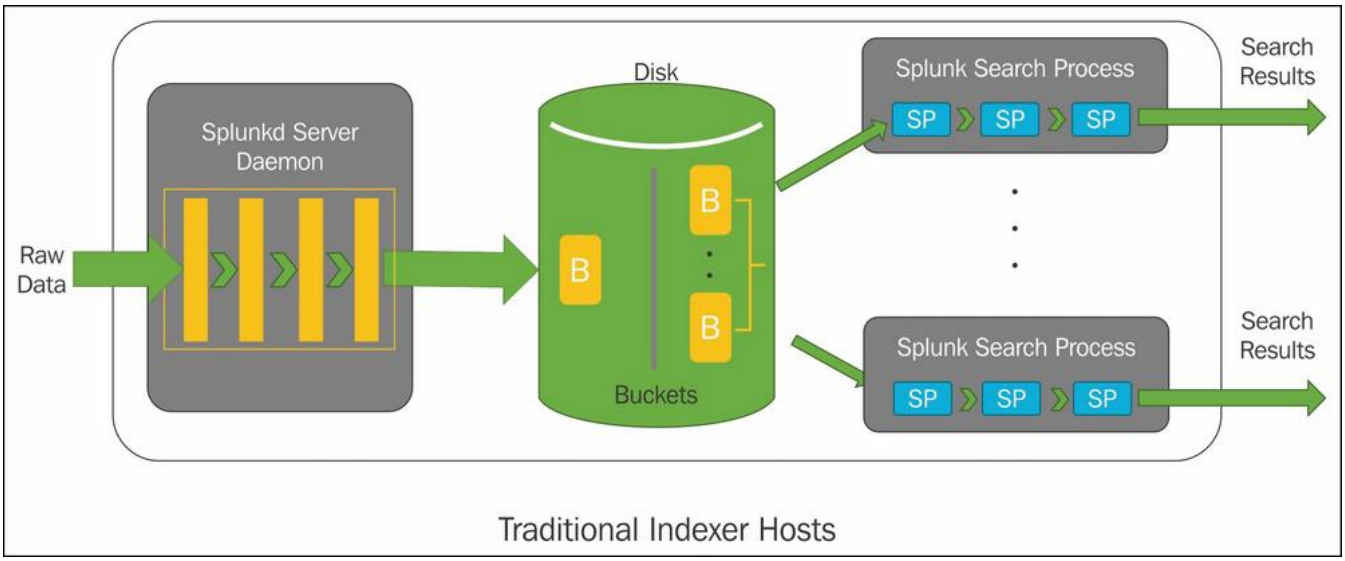
Modules are those components of the Splunk architecture that are used to add new features by modifying or creating processors and pipelines. Modules are nothing but custom scripts and data input methods or extensions that can add a new feature or modify the existing features of Splunk.

The need for parallelization

Splunk's traditional indexer had a single splunkd daemon running on a server that fetched data from different sources, which was then categorized into different indexes. Here, a traditional indexer refers to the indexers that were available in the older version of Splunk. The Splunk search queries are then processed by job queues depending on their priority. The indexer is capable of processing more searches. So, to

utilize the underutilized indexer, there is need for parallelization. Parallelization leads to full utilization of the processing power of the indexer. Expanding Splunk to meet almost any capacity requirement in order to take advantage of the scaling capability of Splunk deployment requires parallel processing of indexers and search heads.

The following figure shows a traditional indexer host, where there is no parallelization and hence the indexer is left underutilized:



Index parallelization

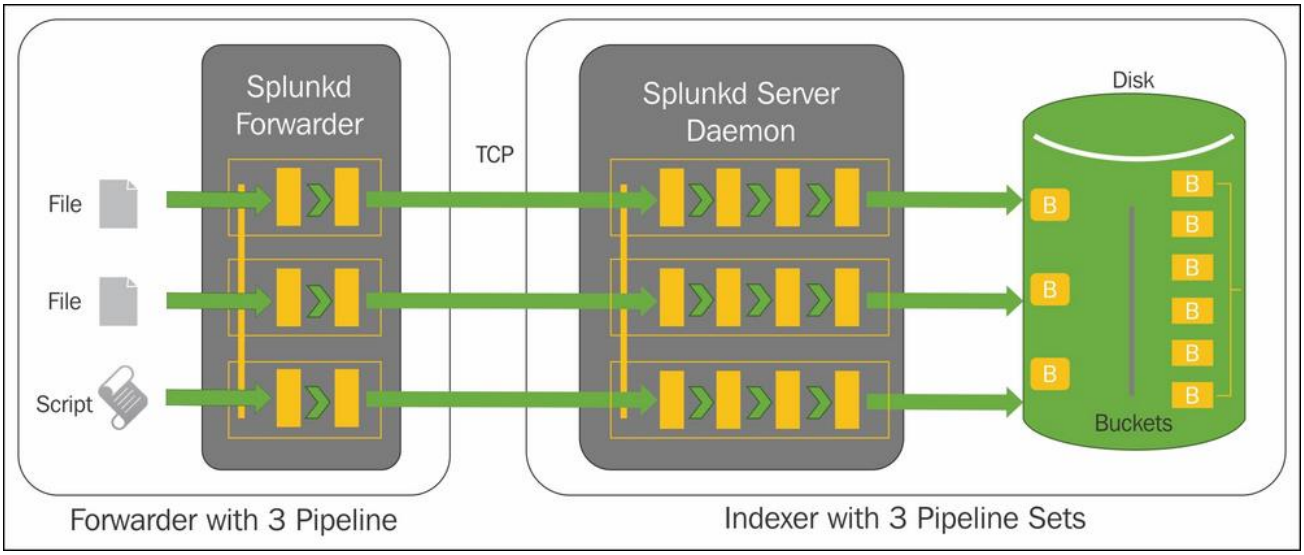
Index parallelization allows an indexer to have multiple pipeline sets. A pipeline set is responsible for processing data from ingestion of raw data, through event processing, to writing the events to a disk.

A traditional indexer runs just a single pipeline set. However, if the underlying machine is underutilized, both in terms of available cores and I/O, you can configure the indexer to run additional pipeline sets. By running multiple pipeline sets, you potentially double the indexer's indexing throughput capacity. Increasing throughput also demands disks with high **Input/output Operations Per Second** (**IOPS**). So, hardware requirements should be taken into consideration while implementing parallelization.

When you implement two pipeline sets, you have two complete processing pipelines, from the point of data ingestion to the point of

writing events to a disk. As shown in the following figure, there is a parallel process for each of the input method and each input is serviced by individual pipelines in the Splunk Server daemon. By enabling an indexer to create multiple pipelines, several data streams can be processed with additional CPU cores that were left underutilized earlier.

This implies that by implementing index parallelization, potentially more data can be indexed on a single indexer with the same set of hardware. It can accelerate parsing of data and writing to a disk up to the limit of indexers' I/O capacity. Index parallelization can double the indexing speed in case of sudden increase of data from the forwarders.



Each pipeline set has its own set of queues, pipelines, and processors. Exceptions are input pipelines that are usually singleton. No states are shared across pipelines sets, and thus, there is no dependency or a situation of deadlock. Data from a single unique source is handled by only one pipeline set at a time. Each component performs its function independently.

The following are the various components of Splunk that are enhanced in Splunk 6.3 and they function as follows to support index parallelization:

- **Monitor input**: Each pipeline set has its own set of TailReaders, BatchReaders, and archive processors. This enables parallel

reading of files and archives on forwarders. Each file/archive is assigned to one pipeline set.

- **Forwarder**: There will be one TCP output processor per pipeline set per forwarder input. This enables multiple TCP connections from forwarders to different indexers at the same time. Various rules such as load balancing rules can be applied to each pipeline set independently.
- **Indexer**: Every incoming TCP forwarder connection is bound to one pipeline set on the indexer.
- **Indexing**: Every pipeline set will independently write new data to indexes. Data is written in parallel for better utilization of underutilized resources. The buckets produced by different pipeline sets could have an overlapping time range.

Next, we'll discuss how to configure multiple ingestion pipeline sets. To do that modify `Server.conf` located at `$SPLUNK_HOME\etc\system\local`as follows for a number of ingestion pipeline sets:

```
[general]
parallelIngestionPipelines = 2 # For 2 Ingestion Pipeline sets
```

NOTE

According to Splunk documents, the default value is `1`.

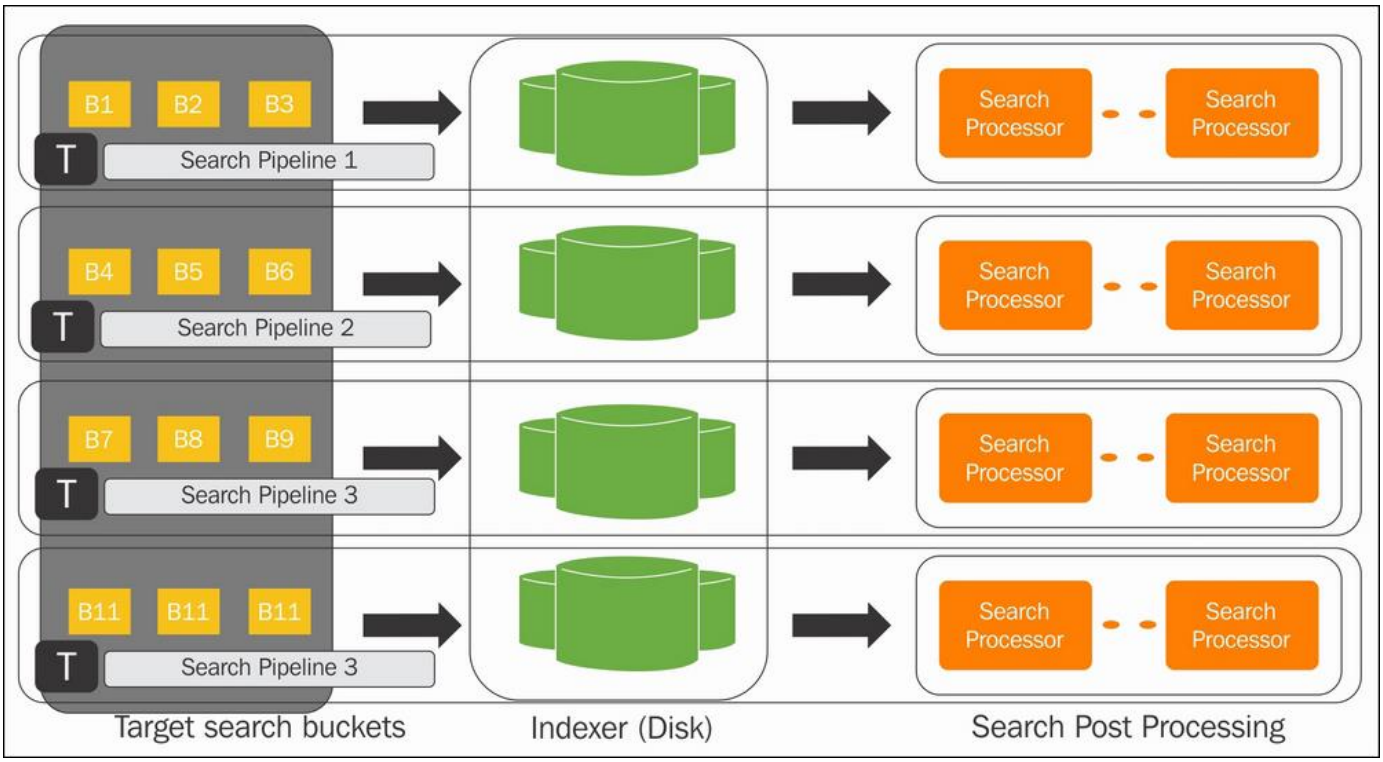
Search parallelization

Once the data is boarded on Splunk, a search is used to create analytics over the indexed data. Here, the faster the search results produced, the more the real-time results will be. Search parallelization is the easiest and most efficient way to speed up transforming searches by adding additional search pipelines on each indexer. This helps in processing of multiple buckets at the same time. Search parallelization can also enable acceleration for a transforming search when saved as a report or report-based dashboard panel.

Pipeline parallelization

Underutilized indexers and resources provide us with opportunities to execute multiple search pipelines. Since there is no sharing of states, there exists no dependency across search pipelines among each other. Though underutilized indexers are candidates for search pipeline parallelization, it is always advised not to enable pipeline parallelization if indexers are fully utilized and don't have the bandwidth to handle more processes.

The following figure depicts that search parallelization searches are designed to search and return event data by bucket instead of time. More the search pipelines added, more the search buckets are processed simultaneously, thus increasing the speed of returning the search results. The data between different pipelines is not shared at all. Each pipeline services a single target search bucket and then processes it to send out the search results.



The default value of `batch_search_max_pipeline` is `1`, and the maximum recommended value is `2`.

Now, we'll discuss how to configure batch search in a parallel mode. To configure a batch search in a parallel mode, modify the `limits.conf` file located at `$SPLUNK_HOME\etc\system\local` as:

```
[search]
batch_search_max_pipeline = 2
```

NOTE

Note that the value should be increased in multiples of `2`.

This increases the number of threads and thus improves the search performance in terms of retrieving search results.

The search scheduler

There have been tremendous improvements in the search scheduler in Splunk 6.3 to improve the search performance and for proper and

efficient resource utilization. The following two important improvements were introduced in Splunk 6.3 that reduces lags and fewer skipped searches:

- **Priority scoring**: Earlier versions of Splunk had simple, single-term priority scoring that resulted in a lag in a saved search, skipping, and could also result in starvation under CPU constraint. Thus, Splunk introduced priority scoring in Splunk 6.3 with better, multi-term priority scoring that mitigates the problem and improves performance by 25 percent.
- **Schedule window**: In earlier versions of Splunk, a scheduler was not able to distinguish between searches that should run at a specific time (such as cron) from those that don't have to. This resulted into skipping of those searches from being run. So, Splunk 6.3 was featured with a schedule window for searches that don't have to run at a specific time.

We'll learn how to configure the search scheduler next. Modify the `limits.conf` file located at `$SPLUNK_HOME\etc\system\local` as follows:

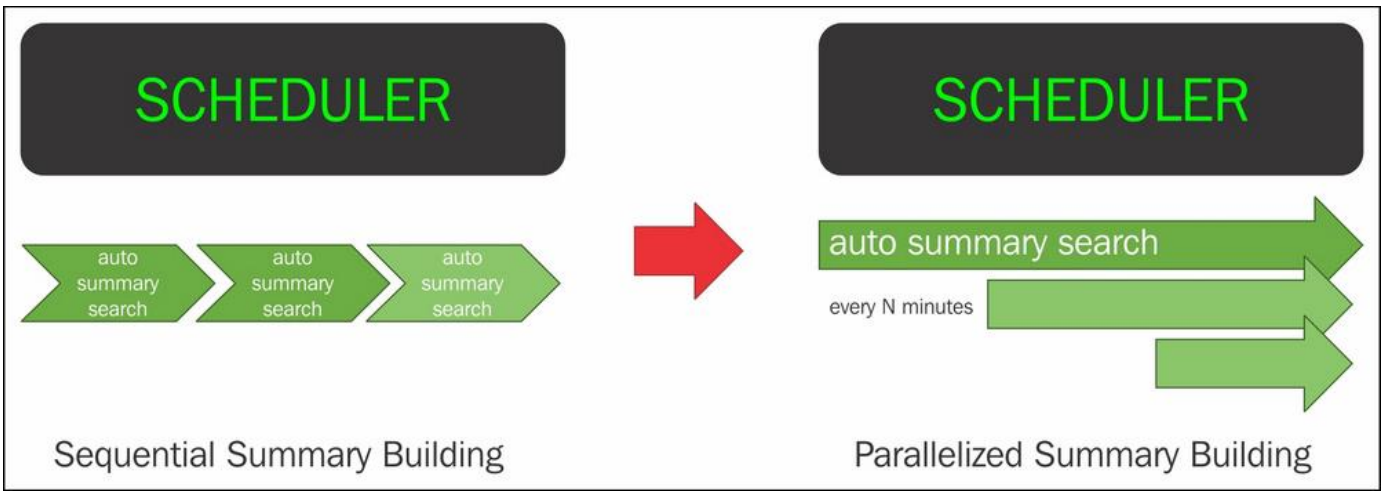
```
[scheduler]
#The ratio of jobs that scheduler can use versus the
manual/dashboard jobs. Below settings applies 50% quota for
scheduler.
Max_searches_perc = 50


# allow value to be 80 anytime on weekends.
Max_searches_perc.1 = 80
Maxx_searches_perc.1.when = ****0,6


# Allow value to be 60 between midnight and 5 am.
Max_searches_perc.2 = 60
Max_searches_perc.2.when = * 0-5 ***
```

Summary parallelization

The sequential nature of building summary data for data models and saved reports is very slow, and hence, the summary building process has been parallelized in Splunk 6.3.



As shown in the preceding figure, in the earlier versions of Splunk, the scheduler summary building was sequential. Because of this, one after the other, there was a performance bottleneck. Now, the summary building process has been parallelized, resulting into faster and efficient summary building.

Now we're going to configure summary parallelization. Modify the `savedsearches.conf` file located at `$SPLUNK_HOME\etc\system\local` as follows:

```
[default]
Auto_summarize.max_concurrent = 3
```

Then, modify the `datamodels.conf` file located at `$SPLUNK_HOME\etc\system\local` as follows:

```
[default]
Acceleration.max_concurrent = 2
```

Data integrity control

Splunk has now come up with the data integrity managing feature in its latest version 6.3. It provides a way to verify the integrity of data that is indexed over Splunk. On enabling this feature, Splunk computes hashes on every slice of uploaded data and stores those hashes so that they can be used to verify the integrity of the data. It is a very useful feature where the logs are from sources such as bank transactions and other critical data where an integrity check is necessary.

On enabling this feature, Splunk computes hashes on every slice of newly indexed raw data and writes it to an `l1Hashes` file. When the bucket rolls from one bucket to another, say from hot to warm, Splunk computes the hash of contents of the `l1Hashes` file and stores it into the `l2Hash` file.

Hash validation can be done on Splunk's data by running the following CLI command:

```
./splunk check-integrity -bucketPath [ bucket path ] [ verbose ]
./splunk check-integrity -index [ index name ] [ verbose ]
```

In case hashes are lost, they can be regenerated using the following commands:

```
./splunk generate-hash-files -bucketPath [ bucket path ]  [ verbose ]
./splunk generate-hash-files -index [ index name ] [ verbose ]
```

Let's now configure data integrity control. To configure data integrity control, modify the `indexes.conf` file located at `$SPLUNK_HOME\etc\system\local` as follows:

```
enableDataIntegrityControl=true
```

NOTE

In a clustered environment, all the clusters and peers should run Splunk 6.3 to enable accurate data integrity control.

Intelligent job scheduling

This section will explain in detail how Splunk Enterprise handles scheduled reports in order to run them concurrently. Splunk uses a report scheduler to manage scheduled alerts and reports. Depending on the configuration of the system, the scheduler sets a limit on the number of reports that can be run concurrently on the Splunk search head. Whenever the number of scheduled reports crosses the threshold limit set by the scheduler, it has to prioritize the excess reports and run them in order of their priority.

The limit is set by a scheduler so as to make sure that the system performance is not degraded and fewer or no reports get skipped disproportionally more than others. Generally, reports are skipped when slow-to-complete reports crowd out quick-to-complete reports, thus causing them to miss their scheduled runtime.

The following table shows the priority order in which Splunk runs different types of searches:

Priority	Search/report type	Description
First priority	Ad hoc historical searches	• Manually run historically searches always run first • Ad hoc search jobs are given more priority than scheduled ad hoc search reports
Second priority	Manually scheduled reports and alerts with real-time scheduling	• Reports scheduled manually use a real-time scheduling mode by default • Manually run searches are prioritized against reports to reduce skipping of manually scheduled reports and alerts

Priority	Search/report type	Description
Third priority	Manually scheduled reports with continuous scheduling	• The continuous scheduling mode is used by scheduled reports, populating summary indexes and other reports
Last priority	Automatically scheduled reports	• Scheduled reports related to report acceleration and data model acceleration fall into this category • These reports are always given last priority

TIP

Caution:

It is suggested that you do not change the settings until and unless you are aware of what you are doing.

The limit is automatically determined by Splunk on the basis of system-wide concurrent historical searches, depending upon the values of `max_searches_per_cpu`, `base_max_searches` in the `limits.conf` file located at `$SPLUNK_HOME\etc\system\local`.

The default value of `base_max_searches` is `6`.

It is calculated as follows:

*Maximum number of concurrent historical searches = (max_searches_per_cpu * number of CPU) + base_max_searches*

So, for a system with two CPUs, the value should be `8`. To get a better clarity see the following worked out example:

*Maximum number of concurrent historical searches = (1 * 2) + 6 = 8*

The *max_searches_perc* parameter can be set up so that it allows more or less concurrent scheduled reports depending on the requirement. For

a system with two CPUs, the report scheduler can safely run only four scheduled reports at a time (50 percent of the maximum number of concurrent historical searches), that is, 50 percent of 8 = 4.

For efficient and full use of the Splunk scheduler, the scheduler limit can vary by time. The scheduler limit can be set to whether to have fewer or more concurrent scheduled reports.

Now, let's configure intelligent job scheduling. Modify the `limits.conf` file located at the `$SPLUNK_HOME\etc\system\local` directory.
The `max_searches_perc.n` is to be set up with appropriate percentages for specific cron periods:

```
# The default limit, used when the periods defined below are
not in effect.
max_searches_perc = 50


#  Change the max search percentage at 5am every day when
specifically there is less load on server.
max_searches_perc.0 = 70
max_searches_perc.0.when = * 0-5 * * *


#  Change the max search percentage even more on Saturdays
and Sundays
max_searches_perc.1 = 90
max_searches_perc.1.when = * 0-5 * * 0,6
```

There are two scheduling modes of manually scheduled reports, which are as follows:

- **Real-time scheduling**: In this type of scheduling, Splunk ensures that the recent run of the report returns current data. This means that a scheduled report with real-time scheduling runs at its scheduled runtime or not at all.

  If there are longer running reports that have not finished or there are many reports with real-time scheduling set to run at the same time, then in that case, some of the real-time scheduling reports may be skipped.

A report scheduler prioritizes reports with real-time scheduling over reports with continuous scheduling.

- **Continuous scheduling**: Continuous scheduling is used in a situation where running the report is eventually required. In case a report with continuous scheduling is not able to run due to one or other reason, then it will run in future after other reports are finished.

  All the scheduled reports are, by default, set to real-time scheduling unless they are enabled for summary indexing. In case of summary indexing, the scheduling mode is set to continuous scheduling because summary indexes are not that reliable if scheduled reports that populate them are skipped.

  If there is any server failure or Splunk Enterprise is shut down for some reason, then in that case, the continuous scheduling mode's configured reports will miss scheduled runtime. The report scheduler can replace all the missed runs of continuously scheduled reports of the last 24 hours when Splunk Enterprise goes online, provided that it was at least once on its schedule before the Splunk Enterprise instance went down.

Let's configure the scheduling mode next. To configure scheduled reports so that they are in a real-time scheduling mode or in a continuous scheduling mode, the `realtime_schedule` parameter in the `savedsearches.conf`file is to be manually changed from `realtime_schedule` to `0` or `1`. Both the scheduling modes are explained as follows:

- `realtime_schedule = 0`: This mode enables scheduled reports that are to be in a continuous scheduling mode. This ensures that the scheduled reports never skip any run. If it cannot run at that moment, it will run later when other reports are over.
- `realtime_schedule = 1`: This mode enables a scheduled report to run at its scheduled start time. If it cannot start due to other reports, it skips that scheduled run. This is the default scheduling mode for new reports.

The app key-value store

The app key-value store is a feature provided by Splunk Enterprise to manage and maintain the state of the application. Using an app key-value store, users can save and retrieve data from Splunk apps.

System requirements

The app key-value store feature is only available in the 64-bit distribution of Splunk Enterprise. It is not available in the 32-bit version of Splunk. It uses the `8191` port by default, but it can be configured from `Server.conf`located at `$SPLUNK_HOME\etc\system\local` by modifying the `[kvstore]` code block.

Uses of the key-value store

The following are some of the uses of a key-value store:

- It can be used to manage the app state of the user interface by storing the session/application state information
- It creates a checkpoint of the uploaded data in case of modular inputs
- It enlists the environment variable used, accessed, or modified by users
- It is the metadata storage of the user
- It caches results from search queries

Components of the key-value store

The key-value store saves data in the collections of the key-value pair. The key-value store files are located on the search heads. The following are the various components of the key-value store:

- **Collections**: Collections are containers for data storage similar to a database table.
- **Records**: Records store the entry of data in the collection.

- **Fields**: Fields contain the value of data in the JSON format file. Fields correspond to the key name similar to columns in the database table.
- **_key**: This is the reserved field that contains a unique ID for each record. It is an autogenerated field that is not explicitly specified.
- **_user**: This is also a reserved field that is used to map the user ID of each record.
- **Accelerations**: This is used to improve search performance that contains the accelerated fields.

Let's take a look at how to create a key-value store collections via a config file. To use a key-value store, we need to create a key-value store collection using the following steps:

1. Create a `collections.conf` file in the application's `default` or `local`directory, as follows `$SPLUNK_HOME\etc\apps\APPNAME\default\collections.confor$SPLUNK_HOME\etc\apps\APPNAME\local\collections.conf`.
2. Modify `collections.conf` by specifying the name of the collection and optionally, the schema of the data. Listed in the following sublist is the description of the parameters which need to be configured in `collections.conf` file:
    - `[collection_name]`: This is the collection name
    - `enforceTypes`: This is set to `True` or `False` to enforce the data types of values when inserting them into the collection.
    - `field.name`: This is an optional field. The available data types are string, time, Boolean, and number. If the data type is not set explicitly, then it is set to JSON.

Any change in `collections.conf` needs a restart of the Splunk instance to apply the changes on the search heads. Refer to the following example for better understanding:

```
[AndroidCollections]  #collection_name
```

The screenshot that follows shows a code snippet of the sample JSON data:

```
{
"Devicename" : "Test Device",
"DeviceID" : "9661",
"DeviceBuild" : "Test build 9661C",
"DeviceAndroidVersion" : "Marshmallow 6.0",
"DeviceIMEI" : "12345678909876",
"DeviceMAC" : "AA:BB:CC:DD:EE:FF",
"DeviceDebugBuild" : "True"
}
```

The following screenshot is the code snippet of the enforce data type for the preceding JSON data:

```
[AndroidCollections]
enforceTypes = true
field.Devicename = string
field.DeviceID = number
field.DeviceBuild = string
field.DeviceAndroidVersion = string
field.DeviceIMEI = number
field.DeviceMAC = string
field.DeviceDebugBuild = Boolean
```

The following screenshot shows the sample code snippet for hierarchical JSON data:

```
{
"Devicename" : "Test Device",
"DeviceID" : 9661,
"DeviceInfo" :
    {
    "DeviceBuild" : "Test build 9661C",
    "DeviceAndroidVersion" : "Marshmallow 6.0",
    "DeviceIMEI" : 12345678909876,
    "DeviceMAC" : "AA:BB:CC:DD:EE:FF"
    },
"DeviceDebugBuild" : True
}
```

The following screenshot shows how a data type can be enforced on hierarchical data using a dot (.) notation:

```
[AndroidCollections]
enforceTypes = true
field.Devicename = string
field.DeviceID = number
field.DeviceInfo.DeviceBuild = string
field.DeviceInfo.DeviceAndroidVersion = string
field.DeviceInfo.DeviceIMEI = number
field.DeviceInfo.DeviceMAC = string
field.DeviceDebugBuild = Boolean
```

Managing key-value store collections via REST

The Splunk REST API can be used to create, read, delete, update, and manage key-value store data and collections. The Splunk REST API

accesses Splunk via the management port (by default, `8089`). The following are the REST endpoints for the key-value store:

- `storage/collections/config`:
  - `GET`: This fetches a list of collections in a specific app
  - `POST`: This creates a new collection in a specific app
- `storage/collections/config/{collection}`:
  - `GET`: This fetches information about a specific collection
  - `DELETE`: This deletes a collection
  - `POST`: This updates a collection
- `storage/collections/data/{collection}`:
  - `GET`: This fetches records from a specific collection
  - `POST`: This inserts a new record into a specific collection
  - `DELETE`: This deletes all records from a specific collection
- `storage/collections/data/{collection}/{id}`:
  - `GET`: This fetches records in a collection by a key ID
  - `POST`: This updates records in a collection by a key ID
  - `DELETE`: This deletes a record in a collection by a key ID
- `storage/collections/data/{collection}/batch_save`:
  - `POST`: This runs one or more save (insert and replace) operations in a specific collection

EXAMPLES

There are various notations used in the following examples, such as `username`, `password`, `IPAddress`, and others. Users need to replace them with their own corresponding values to execute the examples. The following are the examples:

- **Fetching a list of collections for an android app**:
  - ```
    curl -k -u username:password \
    https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config
    ```
- **Creating a new collection called AndroidCollections in the android app**:
  - ```
    curl -k -u username:password \ -d name= AndroidCollections \
    https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config
    ```

- **Defining a collection schema**:
  - `curl -k -u username:password \`
  - `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections \`
  - `-d field.Devicename = string \`
  - `-d field.DeviceID = number \`
  - `-d field.DeviceInfo.DeviceBuild = string \`

    `-d field.DeviceInfo.DeviceAndroidVersion = string`
- **Adding data of the hierarchical JSON format to a collection**:
  - `curl -k -u username:password \`
  - `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections \`
  - `-H 'Content-Type: application/json' \`

    `-d '{ "Devicename" : "Test Device", "DeviceID" : 9661, "DeviceInfo" : { "DeviceBuild" : "Test build 9661C", "DeviceAndroidVersion" : "Marshmallow 6.0", "DeviceIMEI" : 12345678909876, "DeviceMAC" : "AA:BB:CC:DD:EE:FF" }} '`
- **Getting all data from the collection**:
  - `curl -k -u username:password \`

    `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections`
- **Getting a specific range of records from collections, for example, records from `10` to `15`**:
  - `curl -k -u username:password \`

    `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections?sort=Devicename&skip=10&limit=5`
- **Getting a record of a specific key ID**:
  - `curl -k -u username:password \`

    `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections/KEYID`

  Where the key ID is the unique `_key` of collections for which the record is to be fetched.
- **Deleting the record of the specific key ID**:

    - `curl -k -u username:password –X DELETE \`

      `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections/KEYID`

- **Deleting all records of the AndroidCollections collection**:
    - `curl -k -u username:password –X DELETE \`

      `https://IPAddress:8089/servicesNS/nobody/android/storage/collections/config/ AndroidCollections`

Replication of the key-value store

In case of a distributed environment, the key-value store can be replicated to a large number of search heads by enabling replication. By default, the key-value store is not replicated to indexers in distributed deployment of Splunk.

To enable replication, the `collections.conf` file is to be modified and we need to add `replicate = true` to the file.

Splunk Enterprise Security

Splunk Enterprise is connected to various data input sources, indexers, and search heads over a network, and hence, it is very important to harden the security of Splunk Enterprise. Taking necessary steps for **Splunk Enterprise Security** (**SES**) can mitigate risk and reduce attacks from hackers.

The following are ways to secure the Splunk Enterprise deployment:

- Setting up user authentication and creating and managing user access by assigning roles. Splunk has a built-in system for user authentication and to assign roles. Along with the built-in system, it provides integration with the **Lightweight Directory Access Protocol** (**LDAP**). Splunk can be integrated with an active directory and can be used as a centralized authentication system for authentication and to assign roles. Splunk Enterprise 6.3 has been introduced with additional authentication using the **Security Assertion Markup Language** (**SAML**). Splunk Enterprise can be enabled for single sign-ons using SAML, which was explained in detail in the previous section of the chapter.
- Use **Secure Socket Layer** (**SSL**) for secure communication of Splunk deployment. Splunk provides, by default, certificates and keys that can be used to enable SSL communication to provide encryption and data compression while communicating with different components of Splunk deployment. It secures the communication between browsers, Splunk Web, and data sent from forwarders to indexers. Splunk provisions to use your own certificates and keys to secure the communication of Splunk deployment components.
- Keep Splunk installation updated with the latest security patches and updates. Splunk continuously keeps on fixing bugs and comes up with updates on Splunk Enterprise. Splunk releases the bug fix report that has a complete description about the fixes that were updated in the next release. If there are any security-related fixes, Splunk Enterprise deployment should apply that security patch/bug fix so as to make sure that Splunk Enterprise is secure from

outside threats. Continuous auditing of Splunk configuration files and Splunk audit events will result in secure Splunk deployment.

Enabling HTTPS for Splunk Web

We will see how to enable HTTPS from the Splunk Web console for all communications happening via Splunk's web channel. On enabling HTTPS, Splunk will not be able to listen over the HTTP connection, and this is the time when Splunk can be configured to either listen to HTTP or HTTPS communications only!

The following are the steps to enable HTTPS via the Splunk Web console:

1. Access the Splunk Web console via a web browser by typing the IP address followed by the port number.

   For example, `http://IPAddress:Port` or `http://localhost:8000`. Here, `8000` is a default web access port of Splunk Enterprise.
2. Go to **System Menu** | **System Settings**.
3. Click on the radio button to enable HTTPS. Splunk is configured to use default certificates when HTTPS is enabled. The default configuration is available at `$SPLUNK_HOME\etc\auth\web.conf`:

```
[settings]
enableSplunkWebSSL = true
privKeyPath = etc\auth\splunkweb\privkey.pem #Path of Default Private Key
caCertPath = etc\auth\splunkweb\cert.pem #Path of Default Certificate Path
```

We'll now configure Splunk Web with your own certificate and private key. We are talking about securing Splunk, so the default private key and default certificate provided by Splunk Enterprises should be changed for better authentication and security.

Certificates can be self-signed or can be purchased from third-part vendors. Once you have the certificate and private key, the following procedure is to be followed for the changes to take effect.

In our explanation, let's say the certificate filename is `TestCertificate.pem` and the private key is `TestPrivateKey.key`. The following are a series of steps to configure Splunk Web with a certificate and private key:

1. Copy `TestCertificate.pem` and `TestPrivateKey.key` to `$SPLUNK_HOME\etc\auth\splunkweb\`
2. Do not overwrite or delete the existing certificate located at `$SPLUNK_HOME\etc\auth\splunkweb\`, as the certificates are generated on every restart, and any changes made on this certificate and key will be reset
3. Configure `web.conf` located at `$SPLUNK_HOME\etc\system\local`as follows:

```
[settings]
enableSplunkWebSSL = true
privKeyPath = etc\auth\splunkweb\TestPrivateKey.key
caCertPath = etc\auth\splunkweb\TestCertificate.pem
```

Splunk needs to be restarted for the newer settings to take effect, and after the restart of Splunk Server, Splunk Web will be available only via HTTPS URL, that is, `https://localhost:8000`.

Enabling HTTPS for the Splunk forwarder

Configure `inputs.conf` located at `$SPLUNK_HOME\etc\system\local\` of the indexer, as mentioned in the following code block. In this example, port number `9000` is to be configured on the indexer:

```
[SSL]
rootCA = $SPLUNK_HOME\etc\auth\cacert.pem #Path of default
Key
serverCert = $SPLUNK_HOME\etc\auth\server.pem #Path of
default Certificate
password = password
[splunktcp-ssl:9000]
disabled=0
```

The Splunk forwarder needs to be configured to forward using the secure certificate and key. To configure the `outputs.conf` forwarder located at `$SPLUNK_HOME\etc\system\local`, place the following code block as

in the following mentioned code block. In this example, `192.168.1.10` is the IP address of the indexer that was configured in the previous instance:

```
[tcpout]
defaultGroup = splunkssl


[tcpout:splunkssl]
server = 192.168.1.10:9000
sslVerifyServerCert = false
sslRootCAPath = $SPLUNK_HOME\etc\auth\cacert.pem
sslCertPath = $SPLUNK_HOME\etc\auth\server.pem
sslPassword = password
```

Similar to the previous section, even in the indexer and forwarder, the certificates and private keys can be copied to their respective folders. The path of the certificate and private key can be configured in their respective config files. Splunk must be restarted for the settings to take effect.

Securing a password with Splunk

Splunk has an in built feature of encrypting configuration files via SSH. Splunk for its first start up, creates a file named `splunk.secret`, which contains a secret key that is used to encrypt authentication information in configuration files.

The following is the list of information that is encrypted via the `splunk.secret` key:

- `web.conf`: This refers to SSL passwords of every instance
- `authentication.conf`: This refers to the LDAP password; if deployment is LDAP integrated
- `inputs.conf`: This refers to SSL passwords
- `outputs.conf`: This refers to SSL passwords

When Splunk starts and if it detects a clear-text password in any of the preceding configuration files, it creates a configuration in the equivalent local folder with the encrypted password.

In a clustered and distributed environment, when Splunk is deployed on multiple servers, a secure password mechanism of encryption can be very useful to ensure consistency across the deployment.

To apply the same settings of a secret key to all the instances, users just need to configure all the changes in the configuration files and restart Splunk to ensure that the `splunk.secret` file is updated with the latest information.

Once you have the updated file, just copy the `splunk.secret` file to all the other instances and restart the instance, and you will have the same settings you applied to all the instances.

The access control list

Splunk can be configured for high security with an access control list. Using an access control list, various restrictions on the basis of IP address to various components of Splunk deployment can be applied.

The `server.conf` and `inputs.conf` can be edited or modified to specify which IP address should be allowed and which should be restricted for various communications within the Splunk deployment.

In `server.conf` and `inputs.conf`, the `[accept from]` block can be added to allow communication only from a specific IP address. For example, to instruct a node to accept communication from a specific IP address, edit the `[httpserver]` block in `server.conf`; likewise, to restrict TCP communication using SSL to a specific IP address, edit the `[tcp-ssl]`block in `inputs.conf`.

Similarly, various communications of Splunk Web, forwarder, and indexers can be restricted or allowed only from a specific IP address, and thus, security can be enhanced using the access control list features of Splunk Enterprise 6.3.

Authentication using SAML

SAML is an XML standard that allows secure web domains to exchange user authentication and authorization data. It allows one online service provider to contact an identity service provider in order to authenticate users who are trying to access the secure content.

Splunk Enterprise supports the use of SAML authentication and authorization for **Single Sign-On** (**SSO**). SSO can be enabled in Splunk with the configuration settings provided by the **Identity Provider** (**IdP**).

SSO can be configured by Splunk Web or by modifying `authentication.conf` located at `$SPLUNK_HOME\etc\system\default` directly. At present, Splunk Enterprise supports the Ping Identity product from PingFederate® for SSO.

To configure SSO with SAML, the following is the requirement list:

- An identity provider (at present, PingIdentity) is a tested identity provider, and others can also be integrated on similar lines.
- Configuration that uses an on-premise search head.
- A user with an admin role and `change_authentication` Splunk capability. This permission allows us to enable SAML and edit authentication settings on the Splunk search head.

NOTE

SSO must be configured on all the search heads in the Splunk deployment for it to function properly.

We'll now learn how to set up SSO using SAML. Let's get acquainted with the steps of setting up SSO:

1. The following information will be required from IdP to configure Splunk in order to authenticate the user:
    - `role`
    - `realName`
    - `mail`

2. The groups returned by IdP are mapped to Splunk roles. A single Splunk role can be assigned to multiple groups.

Let's configure SSO using SAML via Splunk Web. The following are the steps to configure SSO on Splunk Web:

1. Access Splunk Web by going to `localhost:8000` from the deployment server machine or via `IPAaddress:PortNo` from a machine in the same network.
2. Go to **Settings** | **Access Controls** | **Authentication Method**.
3. Choose **SAML** as the **External Authentication Method** and click on **Configure Splunk** to use SAML.
4. In the **SAML Groups** page, click on **SAML Configuration**.
5. Browse and select the XML file provided by the IdP provider and fill in all the details and click on **Save**.

If all the settings are correct, the **SAML Groups** page will be populated with all the users and groups where specific groups and Splunk roles can be assigned.